Proceedings XII GEOINFO, November 27-29, 2011, Campos do Jordão, Brazil. p 73-84.

The Spatial Star Schema Benchmark

Samara Martins do Nascimento[1], Renata Miwa Tsuruda[2], Thiago Luís Lopes Siqueira[2,3], Valéria Cesário Times[1], Ricardo Rodrigues Ciferri[2], Cristina Dutra de Aguiar Ciferri[4]

[1]Informatics Center, Federal University of Pernambuco, UFPE, 50.670-901, Recife, PE, Brazil, +55 81 2126-8430

[2]Department of Computer Science, Federal University of São Carlos, UFSCar, 13.565-905, São Carlos, SP, Brazil, +55 16 3351-8573

[3]São Paulo Federal Institute of Education, Science and Technology, IFSP, 13.565-905, São Carlos, SP, Brazil, +55 16 3351-9608

[4]Department of Computer Science, University of São Paulo at São Carlos, USP, 13.560-970, São Carlos, SP, Brazil, +55 16 3373-8172

{smn, vct}@cin.ufpe.br, {renata_tsuruda, ricardo}@dc.ufscar.br, prof.thiago@ifsp.edu.br, cdac@icmc.usp.br

Abstract. *Spatial Data Warehouses (SDWs) enable the simultaneous processing of multidimensional queries and spatial analysis. In the literature, little attention has been devoted to the development of benchmarks for analyzing the performance of query processing over SDWs. In this paper, we propose a novel benchmark, called Spatial SSB, designed specifically to perform controlled experimental performance evaluation of SDWs environments. The Spatial SSB proposes a non-redundant SDW schema and controls: the generation of data, the query selectivity and the data distribution in the extent. In addition, the Spatial SSB provides the increase of the data volume, varies the complexity of spatial objects' geometries, and generates a certain number of objects that intersect an ad hoc spatial query window.*

1. Introduction

The experimental performance evaluation of databases systems is carried out mainly using benchmarks to provide a supervised generation of synthetic data and a controlled execution of queries over the synthetic datasets [Barbosa, Manolescu and Yu, 2009]. According to Gray (1993), the research on conventional data warehouse (DW) reached maturity enough to motivate the creation of benchmarks focused in its analysis, such as the *TPC-H* benchmark [Poess, M. et al., 2000] and the *Star Schema Benchmark* (*SSB*) [O'Neil, P. et al., 2009]. Conventional DW stores strictly numeric and alphanumeric data that can be represented by standard data types of SQL. On the other hand, a spatial data warehouse (SDW) consists of a DW that stores spatial data in one or more dimensions or in at least one measure of a fact table [Stefanovic, N. et al., 2000] [Malinowski, E. et al., 2008]. In this sense, the storage of spatial data in DWs allows SOLAP (Spatial On-Line Analytical Processing) query processing, which are based on

predicates that refer to data stored as vector geometries and then enable the simultaneous processing of multidimensional queries and spatial analysis [Rigaux et al., 2002].

However, existing benchmarks for DW do not consider spatial predicates, and the single benchmark in the literature for SDW, called *Spadawan* (*Spatial Data Warehouse Benchmark*) [Siqueira et al., 2010], has some drawbacks, such as: (i) it does not support one-dimensional vector objects (e.g. lines); (ii) it does not enable the adjustment of the complexity of the spatial objects (i.e. number of points that compose the geometry of each spatial object), such as an increase in the number of vertices of polygons; (iii) it does not allow a controlled distribution of spatial data in the extent; (iv) it does not enable queries to retrieve spatial objects based on a given percentage of the extent and therefore on a given selectivity; and (v) it does not define a specific scale factor to generate increasing volumes of spatial data. These are important issues that are tackled by our proposed benchmark.

In this paper, we propose the *Spatial Star Schema Benchmark* (*Spatial SSB*) to evaluate the performance of SOLAP queries over SDWs. Our benchmark extends the *SSB* to enable the storage and the processing of spatial data in dimension tables. The *Spatial SSB* manipulates only synthetic data, ensuring an accurate control over the selectivity of both conventional and spatial data. Also, it defines specific characteristics that can significantly degrade the performance, e.g. the increase of data volume and the increase of the complexity of polygons. Furthermore, aiming at generating synthetic data, we developed a data generator called *Spatial Geometry Generator*, used to produce the location and distribution of *regions*, *nations*, *cities*, *streets* and *addresses*, which are represented respectively by the following spatial data types: polygons, lines and points. Also, the *Spatial SSB* provides predefined spatial hierarchies, e.g. *region_geo* $\preceq$ *nation_geo* $\preceq$ *city_geo* $\preceq$ *street_geo* $\preceq$ c_*address_point_geo*, with the granularity level of *region* being the highest and the granularity level of *address* being the lowest. Regarding the workload, the proposed SOLAP queries of the *Spatial SSB* were obtained by modifying the existing *SSB* queries, reusing the complex operations regarding conventional data and additionally including spatial predicates in each query, to allow the evaluation of topological relationships among spatial attributes.

In order to investigate the impact of different properties for generating synthetic data, we conducted two experiments. Firstly, we investigated the effects of increasing the complexity of spatial data, i.e. the number of points that compose the geometry of each spatial object. Secondly, we tackled the increase of the number of spatial objects according to a given scale factor. The test configurations included the *Spatial SSB*'s SDW schema and a workload composed of spatial and multidimensional queries with controlled query selectivity as well.

This paper is organized as follows. Section 2 surveys related work, Section 3 presents the proposed benchmark *Spatial SSB*, Section 4 details the workload, Section 5 discusses the spatial data generation process, Section 6 describes the experiments using the *Spatial SSB* and finally, Section 7 concludes the paper.

2. Related Work

TPC-H [Poess, M. et al., 2000] and *SSB* [O'Neil, P. et al., 2009] are well-known benchmarks for conventional DWs. *TPC-H* is a decision support benchmark that consists in a suite of business oriented analytical queries and a voluminous fact constellation DW. It represents historical data from orders and sales of a company. *SSB* is based on *TPC-H*, but provides a simpler star

schema [Kimball and Ross, 2002] that was designed by applying several modifications on the original *TPC-H* schema. However, both the *TPC-H* and *SSB* benchmarks cannot be used for SDWs, since they do not allow the generation and storage of spatial data and do not provide the evaluation of spatial predicates.

The *Spadawan*, on the other hand, is a benchmark aimed at performance analysis of SDWs. However, this benchmark only supports point and polygon geometries to represent spatial data. Also, it does not allow varying the amount of points of polygons, thus making it impossible a further analysis over the complexity of spatial objects. *Spadawan* proposes the growth of the spatial data volume by the replication of the geographic objects. However, the increase of the spatial data volume is not based on the scale factor. Another limitation of *Spadawan* is that it does not include changes to all *SSB* queries and does not provide combinations of query windows that refer to different spatial granularity levels in the same query.

In this paper, we propose the *Spatial SSB*, a benchmark for SDWs based on a star schema that allows performance evaluation of queries involving spatial predicates. *Spatial SSB* advances in the state of the art overcoming all the aforementioned limitations, since it considers other spatial data types such as lines to represent *street* networks, ensures a greater control over the selectivity of both conventional and spatial data and generates multidimensional and spatial data automatically. In addition to ensuring the automatic generation of data using its own data generator, the *Spatial SSB* allows the investigation of how increasing data volumes impair query processing performance, by providing a means of varying the number of points denoting the shape of each spatial object (i.e. points of the geometry) or by selecting a database scale factor. Furthermore, the proposed set of *Spatial SSB's* queries also includes innovative aspects since they range from simple queries, with only one level of spatial granularity, to complex queries based on more than one query window that are related to different levels of granularity.

3. The *Spatial SSB*

The *Spatial SSB* schema is shown in Figure 1 and was adapted from the *SSB* schema to include spatial data. It is composed of a fact table *Lineorder*, two dimension tables to store conventional data (i.e. *Part* and *Date*) and six spatial dimension tables to store geometries (i.e. *Customer*, *Supplier*, *Region*, *Nation*, *City* and *Street*). The tables *Customer* and *Supplier* reference the spatial dimension tables *Region*, *Nation*, *City* and *Street* through foreign keys, maintain conventional attributes and the spatial attributes *c_address_point_geo* and *s_address_point_geo* that store the geometries of *Customer* and *Supplier addresses*, respectively. The spatial dimension tables were created following a predefined spatial hierarchy (e.g. *region_geo* $\preceq$ *nation_geo* $\preceq$ *city_geo* $\preceq$ *street_geo* $\preceq$ s_*address_point_geo*) according to the granularity levels. This hierarchy is defined in terms of the *containment* spatial relationship [Malinowski, E. et al., 2008].

The *Spatial SSB* schema is considered hybrid since it eliminates any redundancy in the storage of geometries. As the separate storage of spatial and conventional attributes has been recommended in SDW [Siqueira et al., 2009], we have designed a non-redundant schema based on the claim that computing additional joins is less costly than storing a large amount of redundant spatial data in the spatial dimension table and processing them to answer SOLAP queries. However, we have not created a spatial dimension table for *Customer* and *Supplier addresses* because all spatial objects that represent them are distinct, are points and have a 1:1 association with the dimension table primary key values. For this case, the joint storage of spatial and conventional data does not impair the performance of SOLAP queries [Mateus et al., 2010]. For

each spatial dimension table, a specific spatial data type was used to represent the spatial attribute. Polygons were used in *regions*, *nations* and *cities*, while lines modeled *streets* and points represented *addresses*.

The cardinality of the Spatial SSB schema depends on the *conventional* scale factor (CSF) that corresponds to the *SSB*'s scale factor and on the introduced *spatial* scale factor (SSF). The CSF and the SSF may vary independently for conventional and spatial data. However, SSF must be equal or less than CSF to guarantee a 1:1 association among addresses (i.e. addresses of suppliers and customers) considering conventional and spatial data. For greater CSF or SSF values, larger data volumes will be generated (e.g., SSF = 10 generates ten times more spatial data than SSF = 1). For instance, it is possible to increase the number of spatial objects, to assess how an increasing number of geometries impact the query processing performance.

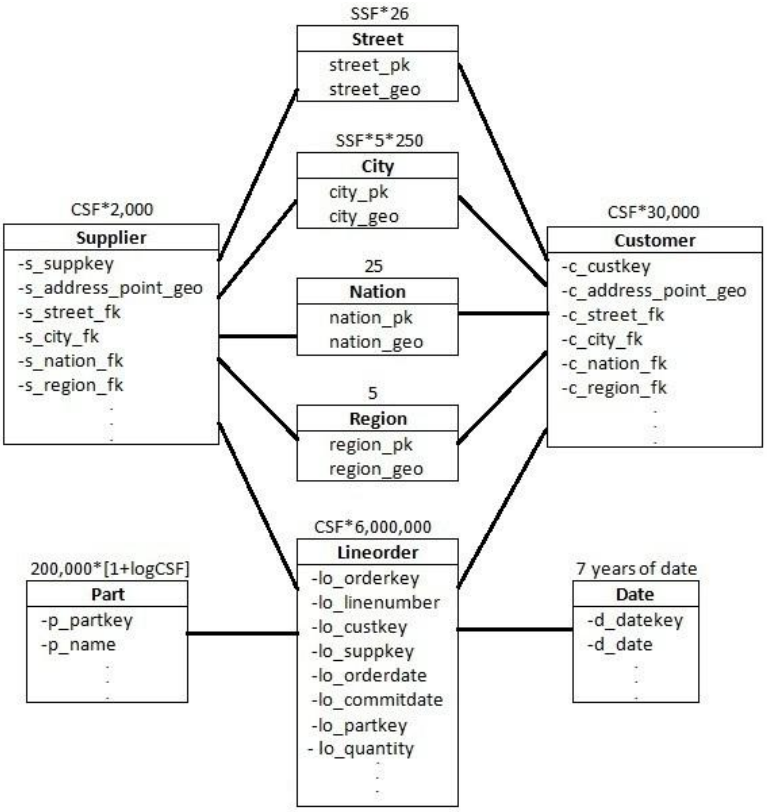


Figure 1. *Spatial SSB* schema

4. Workload

The *Spatial SSB* extends all the *SSB* queries by including spatial predicates based on different query windows (QWs), according to the granularity of the spatial dimension tables and also considering an empty area. The empty area represents oceans, aiming at identifying how an area without intersection with the spatial objects can impact the query processing cost and the query selectivity. The QWs can be (i) predefined, or (ii) can be generated and placed on the extent to comply with a given query selectivity. Each QW overlaps a specific area of the *extent*, retrieving a number of spatial objects and evaluating the spatial relationship *intersection*.

For the predefined QWs, five of them correspond to a different granularity level (*i.e. region*, *nation*, *city*, *street* and *address*), and the last QW intersects the empty area. These six QWs are quadratic, have a correlated distribution with spatial data and their sizes are proportional to the spatial granularity. Aiming at controlling the query selectivity, the *Spatial SSB* enables the retrieval of spatial objects based on a given percentage of the spatial data volume. Thus, the *Spatial Geometry Generator* computes the QW that will retrieve a given number of spatial objects. As a result, the acquisition of a number of objects through the use of an ad hoc query window is not defined a priori, but can vary according to user requests.

The *Spatial SSB* queries assess the performance of conventional and spatial predicates. Regarding the selectivity of a query, it is given by multiplying the Filter Factor (FF) and the cardinality of the table, then obtaining the number of required tuples from the fact table. FF is calculated from the conventional and spatial predicates chosen, which determine the conventional filter factor (CFF) and the spatial filter factor (SFF), respectively. As a result, FF = CFF * SFF. Note that the predefined query windows produce fixed values for the SFF, while the query windows generated to comply with a given selectivity vary the values of the SFF and they can reduce or increase the value of the FF. The *Spatial SSB* queries have the additional properties: use of one or two query windows and the definition of queries for each spatial data type that enable the query selectivity variation.

The queries are shown in Figures 2 to 6 and described in Table 1. The query selectivity values for the six predefined QWs are available at http://gbd.dc.ufscar.br/spatialssb/. Figure 7 shows an example of predefined QW that intersects 5 regions (i.e. R1 to R5), but does not intersect the empty area (i.e. EA). In this example, there are five objects distributed in the extent. The replacement of the conventional predicate of the original queries Q1.1, Q2.1, Q3.1 and Q4.1 of the SSB produced different levels of granularities for the new queries of the Spatial SSB (i.e. *region*, *nation*, *city*, *street*, *address* and *empty area*), obtaining different selectivity results shown in Tables 2, 3, 4 and 5. The results given on the variation of selectivity are based on the modified conventional predicate, and for all examples, the region granularity level was used, without considering the empty area of the extent.

```
SELECT SUM (LO_EXTENDEDPRICE*LO_DISCOUNT) AS REVENUE
FROM LINEORDER, DATE, CUSTOMER, SUPPLIER
WHERE LO_ORDERDATE = D_DATEKEY
AND LO_SUPPKEY = S_SUPPKEY
AND LO_CUSTKEY = C_CUSTKEY
AND D_YEAR = 1993
AND LO_DISCOUNT BETWEEN 1 AND 3
AND LO_QUANTITY < 25
    AND INTERSECTS (REGION, QW1)
    AND INTERSECTS (NATION, QW2)
    AND INTERSECTS (CITY, QW3)
    AND INTERSECTS (STREET, QW4)
    AND INTERSECTS (ADDRESS, QW5)
    AND INTERSECTS (EMPTY_AREA, QW6)
```

Figure 2. Query Q1.1 of *Spatial SSB*

```
SELECT SUM (LO_REVENUE), D_YEAR, P_BRAND1
FROM LINEORDER, DATE, PART, SUPPLIER
WHERE LO_ORDERDATE = D_DATEKEY
AND LO_PARTKEY = P_PARTKEY
AND LO_SUPPKEY = S_SUPPKEY
AND P_CATEGORY = 'MFGR#12'
    AND INTERSECTS (C_REGION, QW1)
    AND INTERSECTS (C_NATION, QW2)
    AND INTERSECTS (C_CITY, QW3)
    AND INTERSECTS (C_STREET, QW4)
    AND INTERSECTS (C_ADDRESS, QW5)
    AND INTERSECTS (EMPTY_AREA, QW6)
GROUP BY D_YEAR, P_BRAND1
ORDER BY D_YEAR, P_BRAND1
```

Figure 3. Query Q2.1 of *Spatial SSB*

```
SELECT C_NATION, S_NATION, D_YEAR, SUM (LO_REVENUE)
AS REVENUE
FROM CUSTOMER, LINEORDER, DATE, SUPPLIER
WHERE LO_ORDERDATE = D_DATEKEY
AND LO_CUSTKEY = C_CUSTKEY
AND LO_SUPPKEY = S_SUPPKEY
    AND INTERSECTS (C_REGION, QW1) AND INTERSECTS (S_REGION, QW1)
    AND INTERSECTS (C_NATION, QW2) AND INTERSECTS (S_NATION, QW2)
    AND INTERSECTS (C_CITY, QW3) AND INTERSECTS (S_CITY, QW3)
    AND INTERSECTS (C_STREET, QW4) AND INTERSECTS (S_STREET, QW4)
    AND INTERSECTS (C_ADDRESS, QW5) AND INTERSECTS (S_ADDRESS, QW5)
    AND INTERSECTS (EMPTY_AREA, QW6)
AND D_YEAR >= 1992 AND D_YEAR <= 1997
GROUP BY C_NATION, S_NATION, D_YEAR
ORDER BY D_YEAR ASC, REVENUE DESC
```

```
SELECT D_YEAR, S_CITY, P_BRAND1,
SUM (LO_REVENUE – LO_SUPPLYCOST) AS PROFIT
FROM DATE, CUSTOMER, LINEORDER, PART, SUPPLIER
WHERE LO_CUSTKEY = C_CUSTKEY
AND LO_SUPPKEY = S_SUPPKEY
AND LO_PARTKEY = P_PARTKEY
AND LO_ORDERDATE = D_DATEKEY
    AND INTERSECTS (C_REGION, QW1) AND INTERSECTS (S_NATION, QW1)
    AND INTERSECTS (C_NATION, QW2) AND INTERSECTS (S_CITY, QW2)
    AND INTERSECTS (C_CITY, QW3) AND INTERSECTS (S_STREET, QW3)
    AND INTERSECTS (C_STREET, QW4) AND INTERSECTS (S_ADDRESS, QW4)
    AND INTERSECTS (EMPTY_AREA, QW6)
AND (D_YEAR = 1997 OR D_YEAR = 1998)
AND P_CATEGORY = 'MFGR#14'
GROUP BY D_YEAR, P_BRAND1
ORDER BY D_YEAR, P_BRAND1
```

Figure 4. Query Q3.1 of *Spatial SSB*

```
SELECT D_YEAR, C_NATION, SUM (LO_REVENUE – LO_SUPPLYCOST) AS PROFIT
FROM DATE, CUSTOMER, LINEORDER, PART, SUPPLIER
WHERE LO_CUSTKEY = C_CUSTKEY
AND LO_SUPPKEY = S_SUPPKEY
AND LO_PARTKEY = P_PARTKEY
AND LO_ORDERDATE = D_DATEKEY
    AND INTERSECTS (C_REGION, QW1) AND INTERSECTS (S_REGION, QW1)
    AND INTERSECTS (C_NATION, QW2) AND INTERSECTS (S_NATION, QW2)
    AND INTERSECTS (C_CITY, QW3) AND INTERSECTS (S_CITY, QW3)
    AND INTERSECTS (C_STREET, QW4) AND INTERSECTS (S_STREET, QW4)
    AND INTERSECTS (C_ADDRESS, QW5) AND INTERSECTS (S_ADDRESS, QW5)
    AND INTERSECTS (EMPTY_AREA, QW6)
AND (P_MFGR = 'MFGR#1' OR P_MFGR = 'MFGR#2')
GROUP BY D_YEAR
ORDER BY D_YEAR
```

Figure 6. Query Q4.3 of *Spatial SSB*

Figure 5. Query 4.1 of *Spatial SSB*

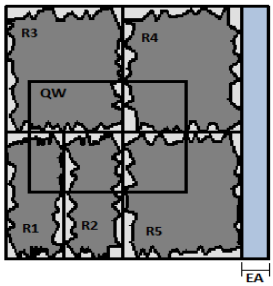


Figure 7. QW on the *extent of Regions*

Table 1. Queries of *The Spatial Star Schema Benchmark*

Query	Characteristics
Q1.1	Returns the increased revenue that is resulted from the elimination of discounts in the company, at a certain scale of percentage for products shipped in a given year, intersects for the different levels of granularity, as shown in Figure 2.
Q1.2	Changes made to the conventional predicates of the query Q1.1, as follows: (i) *d_yearmonthnum* = 199401, (ii) *lo_quantity* is between 26 and 35 and (iii) *lo_discount* is between 4 and 6.
Q1.3	Modifies the conventional predicates of *Spatial SSB* query of type Q1.1, are used: (i) *d_weeknuminyear* = 6; (ii) *d_year* = 1994; (iii) *lo_quantity* is between 36 and 40, and (iv) *lo_discount* is between 5 and 7.
Q2.1	Compares the revenues for some brands of products, grouped by years of orders, intersects for the different levels of granularity, as shown in Figure 3.
Q2.2	Changes made to the previous query type by using *p_brand1* between 'MFGR#2221' and 'MFGR#2228'.
Q2.3	This is obtained by changing the traditional predicate of query Q2.1 of *Spatial SSB*. The modification is done to use *p_brand1* = 'MFGR # 2221'.
Q3.1	Provide the revenues associated with sales and order transactions for a certain period of time. This query uses two QWs for each granularity level, with the spatial predicate intersects, as shown in Figure 4.
Q3.2	Changes were made at conventional predicate that was before analyzed in a certain range of years in order to be computed according to months per year to vary the selectivity.
Q4.1	Query Q4.1 aims to measure the profit from the subtraction of costs from revenues. It is illustrated in Figure 5.
Q4.2	It is an extension of Q4.1, changing the conventional predicate to consider the calculation of profits within a period of time, i.e. in 1997 or 1998.
Q4.3	The conventional and spatial predicates were changed of query Q4.1, to obtaining different selectivity results, as shown in Figure 6.

Table 2. Variation of Selectivity of Query Q1

Query Q1	Selectivity
Q1.1	0.39% to 1.95%
Q1.2	0013% to 0065%
Q1.3	0.0015% to 0.0075%.

Table 3. Variation of Selectivity of Query Q2

Query Q2	Selectivity
Q2.1	0.16% to 0.80%.
Q2.2	0032% to 0.16%
Q2.3	0.02% to 0.1%

Table 4. Variation of Selectivity of Query Q3

Query Q3	Selectivity
Q3.1	3.43% to 85.71%
Q3.2	0.048% to 1.19%

Table 5. Variation of Selectivity of Query Q4

Query Q4	Selectivity
Q4.1	1.6% to 40%
Q4.2	0.046% to 11.42%
Q4.3	0.05% to 1.14%.

5. Data Generation

Aiming at automating the conventional data loading process of the *Spatial SSB*, we implemented a component called *VisualTPCH+SSB*, which is responsible for creating the schemas and loading data from the *SSB* data generator. This component is described in Section 5.1. In addition, for the generation of spatial data, we implemented the *Spatial Geometry Generator* to produce the location and distribution of geometries for *regions*, *nations*, *cities*, *streets* and *addresses*, which is detailed in Section 5.2. Together, these components give rise to *VisualSpatialSSB* tool that is available at http://gbd.dc.ufscar.br/spatialssb/.

5.1 The *VisualTPCH+SSB* Component

The *VisualTPCH+SSB* component manages the storage, generation and load of data for *SSB* schema. The schema is graphically displayed and significant features are available: deleting attributes of a table, renaming tables and deleting tables. The component also offers a graphical visualization of aggregation levels of the generated schema and interactive features, as highlighting the direct ancestral or descendent of the graph of materialized views, deleting vertices and visualizing the SQL command that generates a given vertex (i.e. materialized view) are available. Finally, the data generator of *VisualTPCH+SSB* loads a dataset for each of the considered *benchmarks*.

5.2 The Spatial Geometry Generator

The *Spatial Geometry Generator* component of the *Spatial SSB* benchmark generates rectangles to guide the creation of spatial geometries. Each rectangle is a MBR (*Minimum Bounding Rectangle*) as described as follows and shown in Figure 8. The generation of spatial data contained in the MBRs is based on the *quadtree* space-partitioning model that considers quadrants that are formed from a recursive partitioning of the *extent* [Ghazel, M. et al., 2000]. The space is recursively decomposed into four sub-regions, called “quadrants”, which may have different sizes, but are similar in its shape. Figure 8 also illustrates the empty area in a dark color.

The *extent* size is 0 to 1 in both horizontal and vertical axes. The *extent* is partitioned according to a given predefined amount of *regions*, *nations* and *cities*. The spatial data are generated according to a predefined spatial hierarchy, e.g. $region\_geo \preceq nation\_geo \preceq city\_geo \preceq street\_geo \preceq$ c_*address_point_geo*. The number of *regions* is not limited, enabling to partition the *extent* area in *n* disjoint *regions*. Initially, the partition occurs in the *x* axis, dividing the *extent* in two sub-regions (Figure 8b) and after that, one partition in the *y* axis, forming three sub-regions (Figure 8c), and finally, another partition in the *y* axis, forming four sub-regions (Figure 8d). If necessary, a new partition occurs in the southwest sub-region, dividing it into two sub-regions (Figure 8e), and following this, new partitioning may occur in this southwest sub-region and so on (Figure 8f e 8g). This partitioning is always done clockwise.

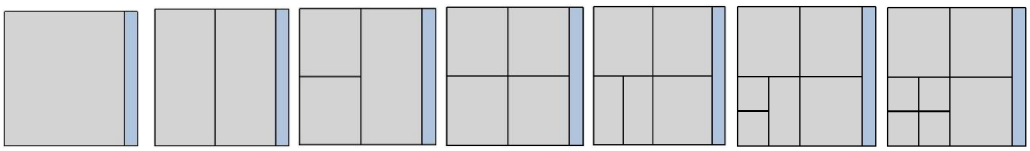


Figure 8. *Extent* partitioning

The distribution of spatial data in the MBR initially considers a margin of M = 0.1% in all sides of the MBR, and thus generates the points inside the margin limit M (Figure 9). Then, the total amount of points (i.e. the complexity of the polygon) is divided by the number of sides of the MBR (i.e. 4 sides), and the points are evenly distributed among the sides, as shown in Figure 10. This distribution of spatial data ensures that the generated geometries will be polygons. The points were generated applying a random function to one of the axes. For the *x* axis, we consider this coordinate growing and continuous and generate the *y* axis from the random function; and, when dealing with the *y* axis, we consider, now, this coordinate growing and continuous and vary the *x* axis from the random function, as can be seen in Figure 10. Figure 11 shows the algorithm to generate points.

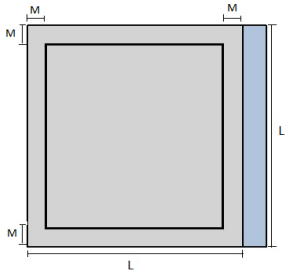


Figure 9. Margin of MBR, Where M = 0.01*L

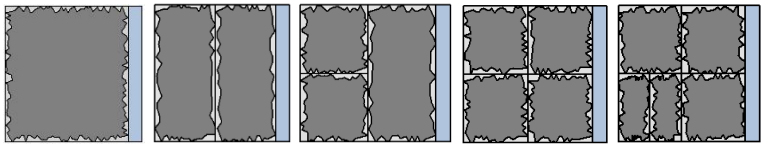


Figure 10. Generation of polygons on MBR

Algorithm 1: CreatePoint (n, x1, y1, x2, y2)

Input: n number of points (the polygon complexity); x1, y1, x2, y2 coordinates of points that form the *extent*.
Output: A file that stores the coordinates of points that compose the polygons.
1 *Read* the number of regions.
2 Margin ← 0.1%; // the margin in x and y axes are defined here.
3 Auxiliary ← n / 4
4 *Write* in the file the first point of the geometry.
5 *While* it does not reach the end of the last calculated point within the range of margin.
6 Uniformly and randomly *increment* a coordinate in an axis and *Generate* the points in the other axis.
7 *FinalResult* ← write in the file the generated points.

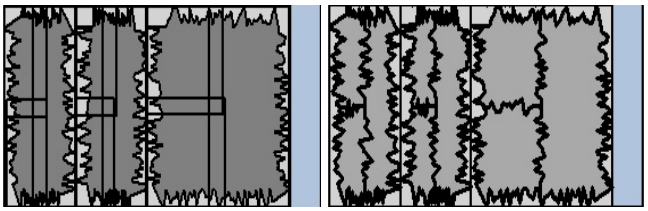
Figure 11. *Extent* partitioning algorithm

In Algorithm 1 of Figure 11, in line 1, the number of regions that are generated is read. In line 2, a margin of 0.1% in the MBR is created to accommodate the generation of points representing the spatial object's geometry. Therefore, margin sizes in the x and y axes are calculated and the ranges of the coordinates, in both *x* and *y* axes, are generated. In line 3, an auxiliary variable divides the total number of points by the number of sides (always considering a MBR) and each side will now have nearly the same number of generated points. In line 4, the first point of the geometry will be written to the file, as well as the *x* position (after the computation of the margin) and the *y* position. A loop is done to check if the generated points are still within the range referenced in this margin and if the points are being generated in a growing and continuous way in both axes.

The number of *nations* is not limited, enabling the partition of a given region in *n* disjoint *nations*. For generating the data distribution of *regions*, we slightly modified the algorithm to represent data distribution in MBRs (Figure 11), as follows. The subdivision of the MBRs is performed within the geometries for *regions*. A margin for *x* axis is obtained from the subtraction of the coordinates x_n and x_0 (last and first point in the axis x of the region), and then multiplying this result by 0.02. Similarly, a margin for *y* axis is obtained from the subtraction of the

coordinates y_n and y_0 (last and first point in the axis y of the region), and then multiplying this result by 0.02. With each new subdivision, new margins are generated and it is within this space that data distribution are generated. An example of this subdivision can be seen in Figure 12a, in which a MBR is shown with *regions* and within each *region* there are the MBRs of *nations*, in this example, three *nations* were generated by *region*. The points distributed into the margin to compose the boundaries of the polygons were generated using a random function that is similar to the random function of *regions*. The polygons of generated *nations* can be seen in Figure 12b.

The algorithm for generating *city* geometries that also were represented by polygons follows the generation of *nations*. The coordinates of *cities* use the points referring to the sides of *nation* geometries added to the generated points within the margin. The amount of points is previously known and is uniformly distributed to all sides of the polygon. Each *city* geometry contains lines to represent *streets*. We generated streets as a rectangular grid of lines that intersect each other, with the same number of lines in each axes. We consider a margin in both *x* and *y* axes to take it as starting points to the generation of lines. We also consider five thousand points to represent a *street*. The generation of lines can be seen in Figure 13. The generation of *addresses* (i.e. point geometries) considers a distribution of one address per street and no address is generated at the intersection of *streets*, as shown in Figure 14.



a. MBR b. Points

Figure 12. Generation of *nations*

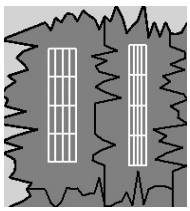


Figure 13. Street

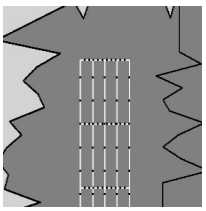


Figure 14. Address

6. Experimental Evaluation

The *Spatial Geometry Generator*, which is the spatial data generator of the *Spatial SSB*, allows that a varying number of spatial objects be distributed in the *extent* and each spatial object be represented by a varying set of points that represents the complexity of the spatial object. Thus, in addition to increasing the data volume by considering the scale factors CSF and SSF defined for the SDW schema of the *Spatial SSB* benchmark, this enlarged volume of data can also be achieved by varying the complexity of geometries of spatial objects.

The characteristics of the proposed *Spatial SSB* benchmark were investigated through two sets of experimental tests, considering three types of *Spatial SSB* queries: (i) The Query Q2.2 was chosen because it uses one QW for each level of granularity, and this QW is predefined; (ii) The Query Q3.1 was selected because two predefined QWs for a given level of granularity were used; and (iii) The Query Q4.3 was chosen because it uses two QWs related to different levels of granularity, and these ad hoc QWs were specified by an user defined percentage of intersection with the *extent*, which retrieved 5% of the spatial objects stored in the SDW. The goal of performing experiments based on the query Q4.3 is to increase the complexity of processing queries with low selectivity by using two QWs related to two different levels of spatial granularity, but that retrieve a small percentage of the spatial objects. The tests illustrate the flexibility of the proposed *Spatial SSB* benchmark for building ad hoc QWs, using predefined QWs and controling selectivity.

The first set of tests aimed at checking the impact of increasing the complexity of spatial objects on the query processsing of SDWs, i.e. increasing the number of points of each geometry of the spatial objects stored in SDWs. These test results are discussed in Section 6.1. The second set of tests investigated the impact of increasing the number of spatial objects. These test results are detailed in Section 6.2. Finally, all the experiments were conducted on a computer with 2.66 GHZ Intel Core i5 processor, 3GB of main memory, 5400 RPM SATA 320 GB hard disk, operating system Linux Ubuntu 9.10, PostgreSQL 8.2.5 and PostGIS 1.3.3.

6.1 Increasing the Complexity of Objects

In this section, we verify the impact of SOLAP query processing performance caused by an increasing number of points that represent each spatial object. We used a database with CSF and SSF equal to 1. The generation of spatial data created 5 disjoint *regions*, 5 *nations* per *region*, totaling 25 *nations*, 1,250 *cities*, 26 *streets*, 2,000 *addresses* for *Supplier* and 30,000 *addresses* for *Customer*. We investigated three configurations that varied from each other according to the number of points generated per region and per nation: 200 points, 20,000 points and 200,000 points. For each dataset, we collected the elapsed time in milliseconds.

The Query Q2.2 was issued against the nation granularity level, with a total of 25 *nations* in the *extent*. The results are shown in Table 6, indicating an increase in query processing cost of 289%, when this time is compared between the smallest number of points and the largest number of points. The greater the number of points used for representing spatial objects, the greater the data volume that impaired the query processing cost. Therefore, the *Spatial SSB* can be used to generate datasets storing spatial objects with distinct complexities, and this can introduce increasing query processing costs.

Another test was conducted, using the Query Q3.1 and two QWs for the same level of granularity. We considered the level of granularity of *regions*, obtaining a total of 5 spatial objects. Table 6 shows the performance results. An increase of 5,814% was obtained when the spatial object representation was changed from 200 to 200,000 points. Therefore we also can conclude that increasing the data volume through varying the complexity of spatial objects was directly related to processing performance losses for the Query Q3.1.

The Query Q4.3 proposes the use of two QWs for different levels of spatial granularity. For this test, it was considered the level of granularity of: (i) customer *region*, with a total of 5 spatial objects, and supplier *nation*, with a total of 25 spatial objects. Table 6 depicts the performance results, showing that there was an increase of 10,063% in the elapsed time of Q4.3 when the representation of an object changed from 200 points to 200,000 points.

Table 6. Elapsed time to process each SOLAP query (milliseconds)

Number of Points	Q2.2	Q3.1	Q4.3
200	363,060	53,299	18,362
20,000	494,942	56,812	24,621
200,000	1,414,542	3,152,581	1,866,190

6.2 Increasing the Number of Spatial Objects

In this section, we verify the impact of SOLAP query processing performance caused by an increasing number of spatial objects. We used the same workbench described in Section 6.1. However, we generated data with CSF and SSF equal to 2, which produced twice the data volume of the datasets described in Section 6.1. Therefore, the generation of spatial data created 5

distinct *regions*, 5 *nations* per *region*, totaling 25 *nations*, 2,500 *cities*, 52 *streets*, 4,000 *addresses* for *Supplier* and 60,000 *addresses* for *Customer*. Besides, we considered the level of granularity city, with 2,500 spatial objects, except for the query Q4.3 that used two levels of granularity: customers' *nation* and suppliers' *city*. For each dataset, we collected the elapsed time in milliseconds.

The performance results described in Table 7 and Figure 15 show that a significant increase in query processing costs was obtained for the CSF and SSF equal to 2, when compared to the same spatial complexity with a CSF and SSF equal to 1 (i.e. Table 6). For instance, the Query 2.1 spent 363,060 milliseconds for handling spatial objects composed of 200 points considering CSF and SSF equal to 1, while the same query spent 3,764,143 milliseconds for processing spatial objects composed of 200 points considering CSF and SSF equal to 2. Considering each query individually, Table 7 indicates an increase of 54% for the Query Q2.2 with regard to query processing costs, when comparing the smallest set of points, i.e. 200 points, with the largest, 200,000 points. For the queries Q3.1 and Q4.3, the increase was of 113% and 94%, respectively.

Table 7. Elapsed time to process each query (in milliseconds)

Number of Points	Q2.2	Q3.1	Q4.3
200	3,764,143	1,944,788	2,636,336
20,000	4,310,462	2,144,696	3,438,380
200,000	5,813,175	4,145,552	5,133,934

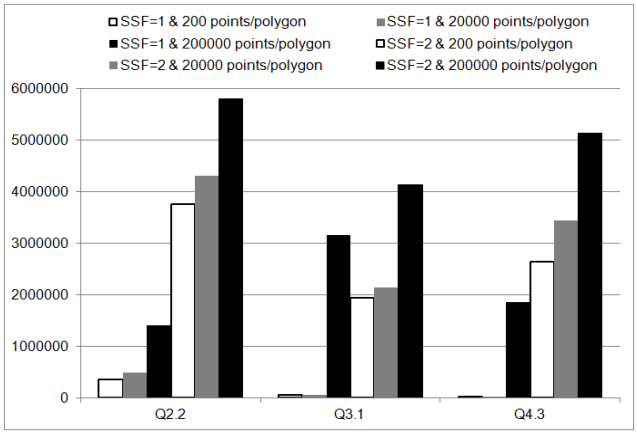


Figure 15. Performance comparison: datasets with scale factor 1 vs scale factor 2

It is noted that the increased volume of data, whether caused by the complexity of the spatial objects or caused by the number of spatial objects, highly impaired the SOLAP query processing performance. Therefore, the *Spatial SSB* can be used to generate datasets storing an increasing number of spatial objects, and this property introduces increasing query processing costs.

7. Conclusion

In this paper we proposed a new spatial data warehouse benchmark called *Spatial SSB* (Spatial Star Schema Benchmark). It is composed by a set of SOLAP queries that was derived from changes made to SSB workload to incorporate spatial predicates for different spatial granularity levels. The dataset is synthetic and is created by a specific data generator, called *Spatial Geometry*, to generate points, lines and polygons. The proposed benchmark allows controlling spa-

tial distribution, the geometric shapes, the data volume, the data complexity and the data selectivity encompassing the main spatial data types.

As future work, we aim to add other types of spatial hierarchies such as those described in [Malinowski, E. et al., 2005] [Malinowski, E. et al., 2008] [Stefanovic, N. et al., 2000]. Another indication of additional research is to incorporate different spatial data types such as complex polygons and vague spatial objects in data generation and in spatial and multidimensional query processing as well [Viswanathan and Schneider, 2011]. Also, an interesting investigation concerns the verification of how data distribution and some SOLAP query issues can affect the performance of a spatial data warehouse.

References

Barbosa, D., Manolescu, I, Yu, J. (2009) "Application Benchmark". Encyclopedia of Database Systems, Springer, p. 99-100.

Ghazel, M., Freeman, G.H. and Vrscay, E.R. (2000) "An effective hybrid fractal-wavelet image coder using quadtree partitioning and pruning". In: IEEE CCECE. p. 416-420.

Gray, J. (1993) "Database and Transaction Processing Performance Handbook". The Benchmark Handbook for Database and Transaction Systems, Morgan Kaufmann, 2nd Edition. p. 99-100.

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

Malinowski, E. and Zimányi, E. (2005) "Spatial Hierarchies and Topological Relationships in the Spatial MultiDimER Model". In: BNCDB. p. 17-28.

Malinowski, E. and Zimányi, E. (2008) "Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications (Data-Centric Systems and Applications)". Springer.

Mateus, R.C., Siqueira, T.L.L., Times, V.C., Ciferri, R.R. and Ciferri, C.D. (2010) "How does the spatial data redundancy affect query performance in geographic data warehouses?". In JIDM, v.1, n.3, p. 519-534.

O'Neil, P., O'Neil, E., Chen, X. and Revilak, S. (2009) "The Star Schema Benchmark and Augmented Fact Table Indexing". In: TPCTC. p. 237-252.

Poess, M. and Floyd, C. (2000) "New TPC benchmarks for decision support and web commerce". In SIGMOD Record, v.29, p. 64-71.

Rigaux, P., Scholl, M. and Voisard, A. (2002) "Spatial Databases with Application to GIS". Morgan Kauffman.

Siqueira, T.L.L., Ciferri, C.D., Times, V.C., Oliveira, A.G. and Ciferri, R.R. (2009) "The impact of spatial data redundancy on SOLAP query performance". In JBCS, v. 15, n. 2, p. 19-34.

Siqueira, T.L.L., Ciferri, C.D., Times, V.C. and Ciferri, R.R. (2010) "Benchmarking Spatial Data Warehouses". In: DaWaK. p. 40-51.

Stefanovic, N., Han, J. and Koperski, K. (2000) "Object-BasedSelective Materialization for Efficient Implementation of Spatial Data Cubes". In IEEE TKDE, v. 12, n. 6, p. 938-958.

Viswanathan, G., Schneider, M. (2011) "OLAP Formulations for Supporting Complex Spatial Objects in Data Warehouses". In: DaWaK. p. 39-50.