applied sciences

MDPI

Article

Decision-Tree-Based Horizontal Fragmentation Method for Data Warehouses

Nidia Rodríguez-Mazahua [1], Lisbeth Rodríguez-Mazahua [1,*], Asdrúbal López-Chau [2], Giner Alor-Hernández [1] and Isaac Machorro-Cano [3]

[1] Tecnológico Nacional de México/I. T., Av. Oriente 9 852, Col. Emiliano Zapata, Orizaba C.P. 94320, Veracruz, Mexico

[2] Universidad Autónoma del Estado de México, Centro Universitario UAEM Zumpango, Camino Viejo a Jilotzingo Continuación Calle Rayón, Valle Hermoso, Zumpango C.P. 55600, Estado de México, Mexico

[3] Universidad del Papaloapan, Calle Circuito Central #200, Col. Parque Industrial, Tuxtepec C.P. 68301, Oaxaca, Mexico

* Correspondence: lrodriguezm@orizaba.tecnm.mx; Tel.: +52-01-272-188-3228

Abstract: Data warehousing gives frameworks and means for enterprise administrators to methodically prepare, comprehend, and utilize the data to improve strategic decision-making skills. One of the principal challenges to data warehouse designers is fragmentation. Currently, several fragmentation approaches for data warehouses have been developed since this technique can decrease the OLAP (online analytical processing) query response time and it provides considerable benefits in table loading and maintenance tasks. In this paper, a horizontal fragmentation method, called FTree, that uses decision trees to fragment data warehouses is presented to take advantage of the effectiveness that this technique provides in classification. FTree determines the OLAP queries with major relevance, evaluates the predicates found in the workload, and according to this, builds the decision tree to select the horizontal fragmentation scheme. To verify that the design is correct, the SSB (star schema benchmark) was used in the first instance; later, a tourist data warehouse was built, and the fragmentation method was tested on it. The results of the experiments proved the efficacy of the method.

Keywords: horizontal fragmentation; decision tree; data warehouse; cost model; data mining

Citation: Rodríguez-Mazahua, N.; Rodríguez-Mazahua, L.; López-Chau, A.; Alor-Hernández, G.; Machorro-Cano, I. Decision-Tree-Based Horizontal Fragmentation Method for Data Warehouses. *Appl. Sci.* **2022**, *12*, 10942. https://doi.org/10.3390/app122110942

Academic Editor: Yosoon Choi

Received: 13 September 2022
Accepted: 25 October 2022
Published: 28 October 2022

1. Introduction

Fragmentation is a design technique in distributed databases that divides database tables into fragments and deals with them like separate database objects; three alternatives exist for this purpose: horizontal, vertical, and hybrid fragmentation. The foundation of the first type of partitioning is the select operator because the selection predicates define the horizontal partitioning; in contrast, the vertical one is carried out by applying the project operator. Hybrid fragmentation involves combining horizontal and vertical fragmentation [1].

A data warehouse (DW) is an integrated, nonvolatile, time-changeable, and subject-oriented data repository that aids in the decision-making activity of the administration. Data warehousing gives frameworks and means for enterprise administrators to methodically prepare, comprehend, and utilize the data to make strategic decisions. DW systems are beneficial assets in the current dynamic and competitive world. Therefore, recently, several corporations (e.g., the Assistance Publique-Hôpitaux de Paris (AP-HP) [2], the National Aeronautics and Space Administration (NASA) Ames Aviation Systems Division [3], John Deere, and DowDuPont [4]) have invested a lot of money in building company-wide data warehouses. The most accepted data model for the development of data warehouses is the multidimensional model, commonly presented as a star, a fact constellation, or a snowflake schema. The star schema is the most conventional modeling paradigm, where the DW is principally composed of a considerably central table (fact table) that contains the volume of the data with no redundancy, and a group of smaller auxiliary tables (dimension tables), one for every single dimension [5]. The main features of data warehouses are (1) their data complexity because of the presence of hierarchy attributes, (2) a high amount of data, and (3) the query complexity due to the presence of joins and aggregations [6].

For a few decades, various fragmentation methods for DW have been presented since this technique has the advantage of minimizing OLAP (online analytical processing) query response time as well as having important benefits in the operations of table loading and maintenance [7]. DW fragmentation is more complex and challenging compared to object-oriented and relational database partitioning due to the variety of options for fragmenting a star schema. In a DW, either the dimension tables, the fact table, or both can be fragmented. Most of the proposed work focuses on horizontal fragmentation because fact tables typically have a high data volume, i.e., they are about millions of tuples [8], and queries usually only require a subset of them. Horizontal fragmentation typically divides DW tables into partitions of tuples according to the workload.

The main advantages of decision trees are that their construction does not require domain expertise or configuration, they deal with multidimensional data, the steps of learning and classification involved in decision tree induction are uncomplicated and quick, and they achieve adequate performance. Furthermore, decision trees are important in classification because they permit obtaining pure partitions (tuple subsets) in a data set utilizing measures including the Gini index, information gain, and gain ratio. Most decision tree induction algorithms have a top-down approach; the decision tree is recursively built by dividing the training data into smaller subsets. The representation of knowledge learned in the shape of a tree is intuitive and usually easy for humans to assimilate; in addition, decision tree classifiers have good precision [9].

Until the time of analyzing the state of the art for this paper, the development of a horizontal fragmentation algorithm for data warehouses that uses decision trees has not been published. The use of a decision tree to horizontally fragment a relational database considering OLTP (online transaction processing) workloads has been reported in [10]; however, [10] does not take into account the characteristics of a DW such as the multidimensional model and OLAP queries. This article describes a horizontal fragmentation method, called FTree, which is focused on obtaining fragmentation schemes that decrease the OLAP query execution cost. To achieve this, first, a comprehensive analysis of the literature about existing horizontal fragmentation methods for DW was realized, and later, FTree was introduced. FTree begins selecting the relevant decision support queries by examining the predicates employed by the workload, and then according to these, it builds the decision tree to choose the horizontal fragmentation scheme. To verify the effectiveness of FTree, the SSB (star schema benchmark) was used in the first instance; later, a tourist data warehouse was developed, and the fragmentation method was tested on this, achieving results that prove the benefit obtained by FTree. Subsequently, some experiments were carried out to compare the horizontal fragmentation scheme (HFS) obtained by FTree with J48 versus the scheme selected by other classifiers; for this comparison, a cost model was used considering that the cost of a decision support query q_i consists of the amount of irrelevant data accessed (ITC) as well as the volume of relevant data located in other fragments (RTC).

Therefore, the contributions of this paper are two-fold: first, a new horizontal fragmentation method for DW is proposed, and second, a cost model to evaluate horizontal fragmentation schemes is introduced.

This paper is made up of the following parts: Section 2 provides the analysis of different works about horizontal fragmentation techniques. Section 3 introduces FTree. Section 4 relates the findings obtained in the development of the fragmentation method and discussion. Finally, Section 5 addresses future work and gives the conclusion.

2. State of the Art of Horizontal Fragmentation Methods

In the literature, some algorithms have been developed to address the challenge of choosing a horizontal fragmentation scheme from a given DW. They are classified as minterm generation algorithms [1], affinity-based algorithms, cost-model-driven algorithms, and data-mining-based algorithms, to mention a few [11]. This section presents some related horizontal fragmentation (HF) techniques and their main characteristics.

The approach proposed in [12] presented a method that uses linear programming to resolve the NP-hard problem of finding an HF scheme in a relational DW. The problem was determined in two concurrent objectives, named: the I/O amount required to answer the queries, and the number of partitions produced to recognize the optimal solutions matched to the Pareto dominance concept.

According to [13], although many techniques have been proposed to speed up data access in the DW, they cannot be applied directly to the object-relational DW (ORDW), since it involves abstract data types for complex data stored in an object format. Therefore, the authors proposed a partition-based approach to efficiently access data in an ORDW. The approach has three steps: (1) obtaining minterm predicates, (2) partitioning the object-relational scheme by derived horizontal partitioning, and (3) fragment selection based on DW properties and query trends.

In contrast, in [14], to obtain fewer fragments and improve the performance of OLAP queries, Kechar and Nait-Bahloul fragmented horizontally just the fact table considering the selectivity of the predicates, their occurrence amount, and their access frequencies. They proposed to split only the fact relation without using the primary HF of the dimension relations. Later, in [15], the authors introduced an improved version of their approach presented in [14].

A technique established on frequent itemset mining was proposed in [16] towards partition, bucket, and sort tables (PBSTs) into a big DW with the more common predicate attributes in the workload. This method considered the quantity of the relation attributes, data skew, and the physical features of the cluster nodes. The authors employed a hash-partitioning approach, which horizontally splits all the tables of a big relational DW, utilizing workload-based PBST methods.

Parchas et al. [17] focused on the “Dist-Key” which is a horizontal partitioning type widely used in DW systems on the market. This kind of horizontal partitioning divides the tuples of a table by using a hash function on the values of a particular attribute identified as the distribution key. Consequently, the proposed strategy tries to reduce the query network cost by choosing the optimal attribute to hash-distribute every DW relation. They suggested BaW (best of all worlds), a mixed method that merges heuristic and exact techniques to select the best distribution key in favor of some tables.

Nevertheless, Barkhoradari and Niamanesh [18] developed a technique called Chabok that involves a two-phase Map–Reduce to resolve DW issues incorporating big data. Chabok is utilized in repositories with the star schema and allows the calculation of distributive measures. This approach deals efficiently with big dimensions; in these dimensions, data size is larger than the bulk of a node. Chabok partitions horizontally the fact table. The Fact Mapper nodes store an identical quantity of tuples in the case of similar nodes. Hadoop and the TPC-DS benchmark were utilized to implement Chabok. The query response time on Chabok overcame significant big data tools for data warehousing, such as Hive and Spark-SQL.

Another approach based on affinity is [19], where the authors considered the horizontal fragmentation of the DW through a simplified decision problem that takes into account the fragmentation derived from the fact table according to the partitioning scheme of a dimension table making use of solely one attribute. The corresponding optimization problem involves calculating a fact table partitioning by limiting the number of fragments by a constant B (quantity of fragments established by the DW administrator) and minimizing the number of I/O operations. A hill-climbing method was developed to choose a near-optimal solution for selecting an optimal fragmentation scheme in a relational DW.

The hill-climbing algorithm has two steps: (1) it finds an initial HFS using an affinity algorithm, and (2) it iteratively improves the initial solution to minimize query execution cost and satisfy B.

Likewise, the problem of selecting an HFS that is supported by the ant-colony-based approach was modeled in [20]; the variables were: the non-fragmented DW scheme, the workload commonly executed, and the largest quantity of partitions necessary for the DW administrator. The HFS that minimizes the total cost of request loading is the resulting output. The approach taken was based on cost.

In [9,21], all the DW tables were fragmented into several buckets (NB) using a hash-partitioning approach and then they were distributed evenly among the cluster nodes to optimize star join queries. The buckets were built in three stages: (1) finding out the near-best NB and the partitioning key by employing the balanced k-means algorithm; (2) obtaining the fact table buckets; and (3) building dimension buckets.

Ettaoufik and Ouzzif [22] implemented an assisted incremental horizontal fragmentation technique with temporary materialized views by a web service. This approach has its basis in managing temporary materialized views for optimizing frequent new queries before proceeding with the fragmentation implementation. The authors used a web service to automatically monitor and improve DW performance and reduce both manual efforts and implementation costs. The web service selects and implements the HFS, and then monitors the performance of the DW to avoid any degradation.

However, to adjust decision-making systems with big data management, in [23], the authors proposed the merger of NoSQL graph-oriented data into the DW; the work details a new method named Big Parallel-ETL which has the aim of adjusting the traditional ETL process (extract–transform–load) through big data tools to speed up data manipulation with the basis of the well-known MapReduce concept distinguished by its suitable parallel-processing feature. The system architecture is comprised of two main layers; one of them has the data sources of the system (Neo4j) and the other layer is comprised of the ETL process that works with the MapReduce paradigm. The main idea of the work consists of operating on a JSON file containing Neo4j data mixed with DW metadata to arrange the multidimensional schema utilizing the MapReduce paradigm beginning with the horizontal partitioning operation and then transforming the sub-JSON files into column-oriented tables to obtain the corresponding multidimensional layout to be unified into the target DW.

Likewise, Munerman et al. [24] proposed an allocation approach of "objects" to "storages" where the nature of the former is defined by the subject area. Later, they considered the speed up of the join operation while processing big data in a specific area. To decrease the execution time of this activity, it is required to optimally distribute the operand tables. Therefore, parallel implementation of the join execution needs an equal data distribution among the DW processors in the cluster. To efficiently resolve the optimal big data distribution, the authors formalized a heuristic method with polynomial computational complexity presenting the objective function and the system of restrictions. The study of the established technique considering different data bulks and diverse data warehouses was realized. A report of the experiments to validate the heuristic greedy optimal distribution proposal was given. The executing time of the approach, even considering massive data, is brief; thus, it can be efficiently applied to distribute data for parallel resolving issues with enormous computational complexity.

On the other hand, [25] established that improving OLAP query efficiency in a distributed system of the kind of Hadoop and Spark is a challenge since the star join operation is the most costly operation involved in these queries and the one requiring significant communication cost. A popular technique that reduces the network flow of the star join operation is to co-partition several DW relations on their join key. Nevertheless, this operation still needs multiple MapReduce cycles in established DW partitioning schemes. For that, the authors proposed two approaches called "FKey" and "NewKey" which have their basis in a data-mining technique to lead their physical design. The partitioning and distribution scheme assists the query optimizer to obtain a significant query-processing plan, to perform the star join execution in one Spark phase minus the shuffle stage. The authors performed experiments on a computer cluster to evaluate their approach using the TPC-DS benchmark.

Finally, Ramdane et al. [26] proposed a dynamic method to optimize group-by operation. This proposal applies partitioning and bucketing approaches to distribute a big DW among the nodes of a Hadoop cluster. This new schema aids the system omit accessing irrelevant data blocks, executes a star join operation in a single Spark phase excluding a shuffle stage, and accelerates group-by aggregation. Experimental findings demonstrated that the technique overcomes related methods considering OLAP query response time.

Table 1 establishes the main characteristics of each horizontal fragmentation method described before. Although several DW fragmentation methods are based on data mining, they use clustering or association techniques. This paper introduces a horizontal fragmentation technique, called FTree, which uses decision trees to select the horizontal fragmentation scheme of data warehouses.

Table 1. Main characteristics of horizontal fragmentation methods.

Work	Classification	Validation
[9]	Data mining (clustering)	TPC-DS benchmark employing Scala language applied in a cluster of similar nodes, a Hadoop-YARN platform, a Spark engine, and Hive.
[12]	Metaheuristic	APB-1 benchmark.
[13]	Minterm predicates	TPC-H benchmark.
[14]	Cost	APB-1 benchmark.
[15]	Cost	SSB (star schema benchmark).
[16]	Data mining (association)	TPC-DS benchmark.
[17]	Graph	Real1 and Real2 join graphs randomly extracted from real-life guests of Redshift with different volumes and frequencies as well as the TPC-DS benchmark.
[18]	Other (Map–Reduce)	TPC-DS benchmark.
[19]	Affinity	APB-1 benchmark.
[20]	Cost	APB-1 benchmark.
[21]	Data mining (clustering)	TPC-DS benchmark.
[22]	Cost	APB-1 benchmark.
[23]	Other (Map–Reduce)	JSON file which contains Neo4j vast data combined with DW metadata.
[24]	Metaheuristic	Experiments in a workstation to identify dependencies such as the execution time of the method compared to the number of data warehouses and the distribution quality.
[25]	Data mining (clustering)	Experiments on a computer cluster utilizing the TPC-DS benchmark.
[26]	Partitioning (hash)	Experiments with the TPC-DS benchmark through a cluster of homogeneous nodes, a Spark engine, a Hadoop-YARN platform, a Hive system, a Ray system, and a Redis database.

3. Materials and Methods

This section introduces the FTree method. First, its four steps are explained. Then, the web application to apply FTree in a DW and the elements of its schema are detailed.

3.1. FTree Method

FTree has four steps, as depicted in Figure 1. It takes as the input the DW to be horizontally fragmented and W, i.e., the upper limit of partitions required by the DW administrator (DWA).

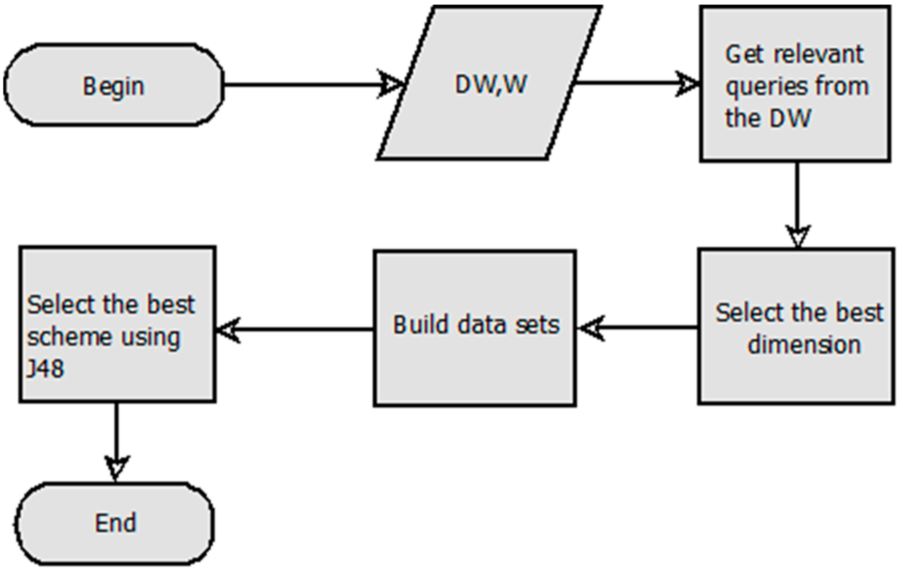

Begin
DW,W
Get relevant queries from the DW
Select the best dimension
Build data sets
Select the best scheme using J48
End

Figure 1. Horizontal fragmentation method for data warehouses based on decision trees.

To explain FTree, we use the SSB (star schema benchmark) [27]; this is demonstrated in Table 2. The SSB consists of a data warehouse with the *lineorder* fact table as well as the *customer*, *supplier*, *part*, and *date* dimension tables.

Table 2. Queries executed in SSB.

Query	Frequency
q_1: SELECT d_month, sum(lo_quantity) from date, lineorder WHERE d_datekey=lo_orderdate AND d_year=1992 GROUP BY d_month;	3
q_2: SELECT d_month, sum(lo_quantity) FROM date, lineorder WHERE d_datekey=lo_orderdate AND d_year=1992 AND d_sellingseason='Summer' GROUP BY d_month;	5
q_3: SELECT d_month, avg(lo_ordtotalprice) from date, lineorder WHERE d_datekey=lo_orderdate AND d_year=1993 GROUP BY d_month;	4
q_4: SELECT d_month, avg(lo_ordtotalprice) FROM date, lineorder WHERE d_datekey=lo_orderdate AND d_year=1993 AND d_sellingseason='Christmas' GROUP BY d_month;	3
q_5: SELECT d_year, sum(lo_quantity) FROM date, lineorder WHERE d_datekey=lo_orderdate AND d_month='January' GROUP BY d_year;	3
q_6: SELECT d_month, sum(lo_revenue) FROM lineorder, date WHERE d_datekey=lo_orderdate and d_year=1995 GROUP BY d_month;	2
q_7: SELECT d_year, sum(lo_quantity) FROM date, lineorder WHERE d_datekey=lo_orderdate AND d_sellingseason='Winter' GROUP BY d_year;	5
q_8: 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 s_region='AMERICA' GROUP BY d_year, p_brand1 ORDER BY d_year, p_brand1;	2
q_9: 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_brand1='MFGR#2221' AND s_region='EUROPE' GROUP BY d_year, p_brand1 ORDER BY d_year, p_brand1;	2

The data warehouse (DW) is made up of several dimension tables $D = \{d_1, d_2, \ldots, d_m\}$ and a fact table F. To carry out the fragmentation, the following steps are performed:

1. ***Obtain relevant queries from DW.*** For this step, it is necessary to keep only the queries performed in the DW that imply joins of a dimension table d_i with F; every di-

mension table d_i will have a set of queries $Q_i = \{q_{i1}, q_{i2}, \ldots, q_{in}\}$. Additionally, the frequency of queries $F_i = \{f_{i1}, f_{i2}, \ldots, f_{in}\}$ is obtained, each query q_{ij} has a set of predicates $Pr_{\mathrm{ij}} = \{p_{ij1}, p_{ij2}, \ldots, p_{ijr}\}$, and finally, the selectivity of predicates $S_{ij} = \{sel_{ij1}, sel_{ij2}, \ldots, sel_{ijr}\}$ is required. The relevant queries presented in Table 2 are considered. Table 3 depicts the predicates used by the queries and their selectivity.

Table 3. Predicates used by the queries.

Pr	Description	S
p_1	*d_year = 1992*	sel_1 = *907,987*
p_2	*d_sellingseason = 'Summer'*	sel_2 = *2,076,040*
p_3	*d_year = 1993*	sel_3 = *908,288*
p_4	*d_sellingseason = 'Christmas'*	sel_4 = *912,008*
p_5	*Dumont = 'January'*	sel_5 = *541,483*
p_6	*d_year = 1995*	sel_6 = *909,991*
p_7	*d_sellingseason = 'Winter '*	sel_7 = *1,577,160*
p_8	*p_category = 'MFGR#12'*	sel_8 = *236,816*
p_9	*s_region = 'AMERICA"*	sel_9 = *1,134,371*
p_{10}	*p_brand1 = 'MFGR#2221',*	sel_{10} = *5848*
p_{11}	*s_region = 'EUROPE'*	sel_{11} = *1,140,311*

2. ***Select the best dimension.*** The most important dimension is selected in this step. Equation (1) is used to calculate the importance of each dimension, where n is the number of queries executed against the dimension d_i and r is the number of predicates for the dimension d_i involved in each query. For this, the query frequency is multiplied by the selectivity of its predicates. The selected table is called *dim*. Table 4 shows the importance of dimensions for the scenario. For example, the importance of dimension *part* is 485,328 because we multiply $f_8 * sel_8 + f_9 * sel_{10} = 2 * 236{,}816 + 2 * 5848$.

$$I_i = \sum_{j=1}^{n} f_{ij} \left(\sum_{k=1}^{r} sel_{ijk} \right) \tag{1}$$

Table 4. Dimension importance for the scenario.

Dimension	Importance
date	*38,076,239*
supplier	*4,549,634*
part	*485,328*

3. ***Build data sets.*** This step consists of building $W - 1$ data sets used to train the J48 algorithm to select the best horizontal fragmentation scheme (HFS). J48 was used because it obtained better performance than other decision tree algorithms according to [28], where it was demonstrated that J48 built decision trees with better results considering four evaluation metrics: precision, recall, F-measure, and ROC area. Algorithm 1 shows the steps to build the data sets. According to lines 1–3, it receives as an input W and a predicate usage matrix (PUM) of the queries executed in both the fact table and *dim*. Table 5 shows the PUM for the example. The data sets in ARFF (attribute-relation file format) are the output of Algorithm 1 (line 4). In line 5, c is set to two because this is the minimum number of fragments of the scheme. Then, a partition tree (PT) is generated [29], where initially, every predicate is considered to be in a single partition (see Table 6, where each column corresponds to a fragment and the * indicates an empty partition); from step 1 to step $r - 1$, where r is the number of predicates of *dim*, a merging profit matrix (MPM) is obtained (lines 6 and 7), which measures the benefit obtained by merging a pair of predicates. When a pair of predicates is merged, the number of remote tuples accessed is reduced, as some queries that previously required tuples from multiple fragments will now only need to access one fragment. For example, the PUM of Table 5 shows that the query q_2 uses the predicates p_1 and p_2; when they are combined in a partition in the second step of the PT depicted in Table 6, q_2 only accesses one fragment, while the number of irrelevant tuples retrieved is increased because some queries will now have to retrieve all tuples from the new fragment even if they only require those from one of the merged fragments. For example, in Table 5, the query q_1 only needs the tuples that satisfy the predicate p_1, but when the two predicates (p_1 and p_2) are merged, it also has to access the tuples from predicate p_2. The MPM is obtained as in [30], which takes the PUM as the input and calculates the decreased volume of remote tuples retrieved (DRT) and the increased number of irrelevant tuples obtained (IIT). The merging benefit of two predicates will be equal to the difference between DRT and IIT. The predicate pair with the highest merging profit will be merged (lines 8–9). When step r-(W-c) is reached (line 10), the first data set ($data\_set_c$) is obtained (line 11). With each merger, a new data set will be generated with one more fragment (line 12), until finding the last data set consisting of W minus one, because the last fragment would be the complement. In the example, W = 4; then, $data\_set_2$ and $data\_set_3$ were obtained for steps 6 and 5, respectively. Figure 2 shows *data_set2.arff*, which is one of the data sets obtained as a result of Algorithm 1. This data set includes the workload given as input to J48 to build the decision tree.

Algorithm 1. Generate data sets.

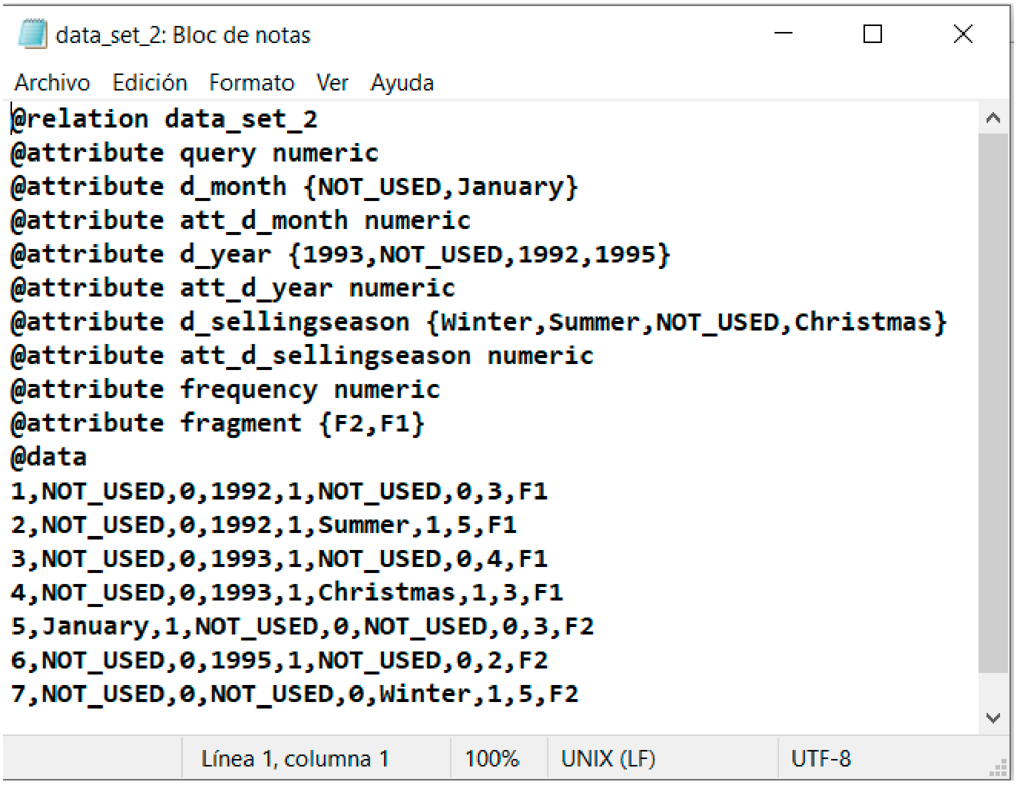
```
1   Data: PUM of the fact table $F$ and $dim$ (a set of queries $Q = \{q_1, q_2, \ldots, q_n\}$,
2   the frequency $f_i$ of every query $q_i$, a set of predicates $Pr = \{p_1, p_2, \ldots, p_r\}$, the
3   selectivity $sel_j$ of every predicate $p_j$), $W$
4   Result: data sets $D = \{data\_set_2, data\_set_3, \ldots, data\_set_{w-1}\}$
5   $c = 2$
6   for each $step_i \in PT \mid 1 \leq i \leq r-1$ do
7       getMPM(PUM, MPM);
8       choose two fragments with the greatest merging profit;
9       fuse the fragments;
10      if $i > r$-($W$-$c$)
11          generate $data\_set_c$
12          $c = c + 1$;
13      end;
14  end;
```

Table 5. Predicate Usage Matrix for the scenario.

Q/Pr	p_1	p_2	p_3	p_4	p_5	p_6	p_7	F_i
q_1	1	0	0	0	0	0	0	3
q_2	1	1	0	0	0	0	0	5
q_3	0	0	1	0	0	0	0	4
q_4	0	0	1	1	0	0	0	3
q_5	0	0	0	0	1	0	0	3
q_6	0	0	0	0	0	1	0	2
q_7	0	0	0	0	0	0	1	5
sel_j	907,987	2,076,040	908,288	912,008	541,483	913,927	1,577,160	

Table 6. Partition Tree for the scenario.

Step	Predicates						
1	p_1	p_2	p_3	p_4	p_5	p_6	p_7
2	p_1, p_2	*	p_3	p_4	p_5	p_6	p_7
3	p_1, p_2	*	p_3, p_4	*	p_5	p_6	p_7
4	p_1, p_2	*	p_3, p_4	*	p_5, p_6	*	p_7
5	p_1, p_2	*	p_3, p_4	*	p_5, p_6, p_7	*	*
6	p_1, p_2, p_3, p_4	*	*	*	p_5, p_6, p_7	*	*



```
@relation data_set_2
@attribute query numeric
@attribute d_month {NOT_USED,January}
@attribute att_d_month numeric
@attribute d_year {1993,NOT_USED,1992,1995}
@attribute att_d_year numeric
@attribute d_sellingseason {Winter,Summer,NOT_USED,Christmas}
@attribute att_d_sellingseason numeric
@attribute frequency numeric
@attribute fragment {F2,F1}
@data
1,NOT_USED,0,1992,1,NOT_USED,0,3,F1
2,NOT_USED,0,1992,1,Summer,1,5,F1
3,NOT_USED,0,1993,1,NOT_USED,0,4,F1
4,NOT_USED,0,1993,1,Christmas,1,3,F1
5,January,1,NOT_USED,0,NOT_USED,0,3,F2
6,NOT_USED,0,1995,1,NOT_USED,0,2,F2
7,NOT_USED,0,NOT_USED,0,Winter,1,5,F2
```

Figure 2. Data set for step 6 of the PT.

Figure 2 shows that the instances of the data set represent the queries and the attributes correspond to the predicates; a numeric variable is added for each attribute involved in the predicates. Additionally, the query frequency is considered in the data sets. The class label attribute is *fragment*; the value for this attribute is assigned according to the partition tree and when a query uses predicates located in different fragments, the selectivity is considered. For example, according to Table 5, q_4 uses the predicates p_3 and p_4. The selectivity of these predicates is sel_3 = 908,288 and sel_4 = 912,008. In the second step of the partition tree of Table 6, they are in different fragments; p_3 is in the second fragment (F2) and p_4 is in the third one (F3). Therefore, the label F3 is assigned to q_4, because $sel_4 > sel_3$.

4. Apply J48 to data sets. The Weka API (application program interface) is used to apply the J48 algorithm to the W-1 data sets. The data set with the best results achieved for four performance metrics (precision, recall, F-measure, and ROC area) will be chosen. Table 7 shows the best HFS of the example, i.e., $data\_set_2$. We used 5-fold cross-validation. Figure 3 depicts the decision tree for $data\_set_2$. Therefore, the fragmentation scheme is $fr_1 = \{p_1, p_2, p_3, p_4\}, fr_2 = \{p_5, p_6, p_7\}, fr_3 = \{\neg(p_1, p_2, p_3, p_4), \neg(p_5, p_6, p_7)\}$.

Table 7. Comparative table of fragmentation schemes.

Scheme	Precision	Recall	F-Measure	ROC Area
$data\_set_2$	**0.286**	**0.429**	**0.343**	**0.208**
$data\_set_3$	-	0.429	-	0.244

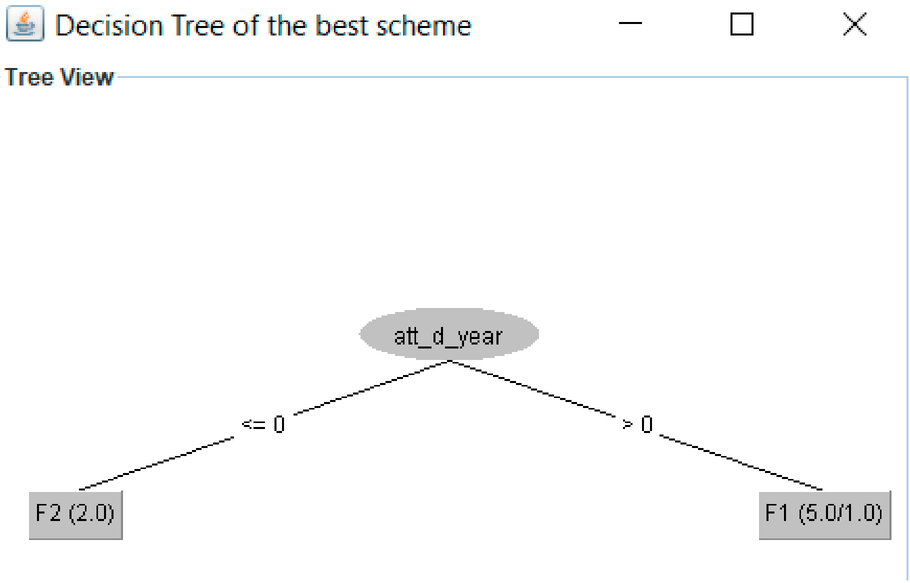
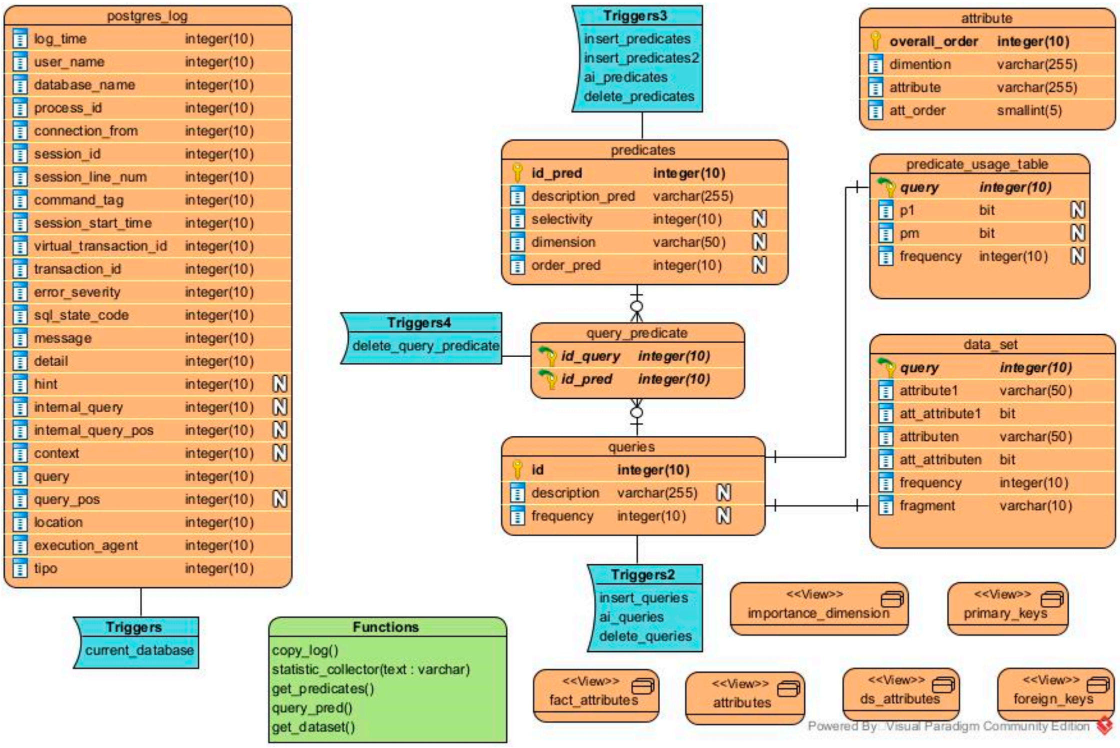


Figure 3. Decision tree of the best scheme (with better results for the evaluation metrics).

3.2. FTree Web Application

We also developed a web application using the Java programming language, the API of Weka, and the database management system (DBMS) PostgreSQL to apply FTree in a DW. First, the DWA must provide the IP of the server where the data warehouse is located, the port, as well as his/her username and password. Then, the DW to be fragmented and the fact table are selected and the upper limit of partitions is entered (W). When these data are provided and the fragmentation proceeds, the FTree schema of Figure 4 is created in the DW. Table 8 describes every element of this schema. Figure 6 shows a partition tree in the web application, while Figure 8 depicts a PUM, Figure 10 presents the comparative table of two fragmentation schemes, and Figure 3 exhibits a decision tree in the web application.



postgres_log		
log_time	integer(10)	
user_name	integer(10)	
database_name	integer(10)	
process_id	integer(10)	
connection_from	integer(10)	
session_id	integer(10)	
session_line_num	integer(10)	
command_tag	integer(10)	
session_start_time	integer(10)	
virtual_transaction_id	integer(10)	
transaction_id	integer(10)	
error_severity	integer(10)	
sql_state_code	integer(10)	
message	integer(10)	
detail	integer(10)	
hint	integer(10)	N
internal_query	integer(10)	N
internal_query_pos	integer(10)	N
context	integer(10)	N
query	integer(10)	
query_pos	integer(10)	N
location	integer(10)	
execution_agent	integer(10)	
tipo	integer(10)	

Triggers
current_database

Functions
copy_log()
statistic_collector(text : varchar)
get_predicates()
query_pred()
get_dataset()

Triggers3
insert_predicates
insert_predicates2
ai_predicates
delete_predicates

predicates		
id_pred	**integer(10)**	
description_pred	varchar(255)	
selectivity	integer(10)	N
dimension	varchar(50)	N
order_pred	integer(10)	N

Triggers4
delete_query_predicate

query_predicate	
id_query	***integer(10)***
id_pred	***integer(10)***

queries		
id	**integer(10)**	
description	varchar(255)	N
frequency	integer(10)	N

Triggers2
insert_queries
ai_queries
delete_queries

attribute	
overall_order	**integer(10)**
dimention	varchar(255)
attribute	varchar(255)
att_order	smallint(5)

predicate_usage_table		
query	***integer(10)***	
p1	bit	N
pm	bit	N
frequency	integer(10)	N

data_set	
query	***integer(10)***
attribute1	varchar(50)
att_attribute1	bit
attributen	varchar(50)
att_attributen	bit
frequency	integer(10)
fragment	varchar(10)

Figure 4. Physical model of the FTree schema.

Table 8. Elements of the FTree schema created in the DW.

Name	Type	Description
postgres_log	Table	It stores all the queries found in the log file.
current_database	Trigger	Before inserting a query into the *postgres_log* table, it verifies that only the relevant queries were saved.
copy_log()	Function	It copies the contents of the current day's transaction log to the *postgres_log* table.
queries	Table	It stores the identifier, the description, and the frequency of the queries.
delete_queries	Trigger	After removing a query on the *queries* table, it reduces the value of the identifier by 1.
insert_queries	Trigger	Before inserting a query into the *queries* table, if the query is already stored in this table, then it is not inserted and its frequency is increased by 1.
ai_queries	Trigger	After inserting a query into the *queries* table, it updates its identifier if it is bigger than the number of tuples.
statistic_collector(text)	Function	It analyzes the queries stored in *postgres_log* to insert into the table *queries* only those executed in the table to be fragmented. Its parameter is the fact table.
attributes	View	It stores the names of the dimensions as well as the attributes of each dimension and the order in which they appear in the dimension table.
attribute	Table	It saves the data from the attributes view and adds a global order attribute.
fact_attributes	View	It stores the name of the fact table as well as the names and order of each of its attributes.
predicates	Table	It saves the identifier, description, and selectivity of the predicates, as well as their dimensions and order in that dimension.
insert_predicates	Trigger	Before inserting predicates into the table, it analyzes that only those from queries involving the fact table with the dimension table are taken.
insert_predicates2	Trigger	Before a predicate is inserted into the table, it validates that only different predicates are recorded.
ai_predicates	Trigger	After inserting a predicate into the *predicates* table, it updates its identifier if it is greater than the maximum number of rows.
delete_predicates	Trigger	After removing a predicate in the predicates table, it reduces the value of the identifier by 1.
get_predicates()	Function	It obtains the predicates of the queries stored in *queries* and stores them in the *predicates* table.
query_predicate	Table	It saves the relationship between queries and predicates.
delete_query_predicate	Trigger	After deleting from the *query_predicate* table, it sets to 1 the sequences for queries and predicates.
predicate_usage_table	Table	It stores the PUM that Algorithm 1 takes as input.
data_set	Table	It contains the data_set obtained by Algorithm 1.
query_pred()	Function	It analyzes which predicates appear in which queries and keeps this relation in *query_predicate*.
get_dataset()	Function	It creates the data sets that are used to build the decision tree.
importance_dimension	View	It stores the importance of the dimensions.
ds_attributes	View	The attributes of the data set table and their order are in this view.
primary_keys	View	It stores the name and the attributes of primary keys for all the DW tables.
foreign_keys	View	The name of the foreign keys for each dimension and the attribute in the fact table involved in this constraint are in this view.

In the following section, we compare the horizontal fragmentation schemes obtained by FTree with J48 versus FTree with other classification techniques such as naïve Bayes and multilayer perceptron. These algorithms were used in the comparison analysis because they are two classification techniques widely known for their effectiveness [31,32].

Naïve Bayes is a classification algorithm that applies the probability theory to indicate the relationship among variables and class labels; this model is one of the easiest and most known probabilistic classification models. Naïve Bayes realizes a simplifying supposition concerning the class-conditional probabilities, understood as naïve Bayes presumption; using this, it is possible to obtain confident estimates of class-conditional probabilities, still with a large number of attributes [33].

A multilayer perceptron is a kind of artificial neural network (ANN) and is considered a powerful classification approach that can find considerably complex and nonlinear decision boundaries purely coming from the data. In a multilayer perceptron, the basic concept of a perceptron is generalized to more elaborated architectures of nodes that can learn nonlinear decision boundaries [33]. In the next section, the main findings of the evaluation of FTree are presented.

4. Results and Discussion

This section includes the findings obtained with FTree using a real DW and the comparison of FTree with J48 versus other classifiers using a cost model.

4.1. Description of the Second Scenario: Tourist Data Warehouse

To evaluate FTree with real data, a tourist DW was built following the methodology proposed by [34]. We used information from the main organizations that regulate tourist activity in Mexico: SECTUR (Secretaría de Turismo), SEGOB (Secretaría de Gobernación), UWTO (United Nations World Tourism Organization), and INEGI (Instituto Nacional de Estadística y Geografía). The type of multidimensional scheme to be used in this case corresponds to a constellation of facts. Under PostgreSQL selected as the DBMS and Pentaho as the ETL (extraction/transformation/loading) tool, dimensional modeling of the touristic DW was carried out, which can be seen in Figure 5. Table 9 shows that the DW has two fact tables (*Hotel_activity* and *International_visitors*) and four dimensions (*Location*, *Time*, *Tourism*, and *Tourist*).

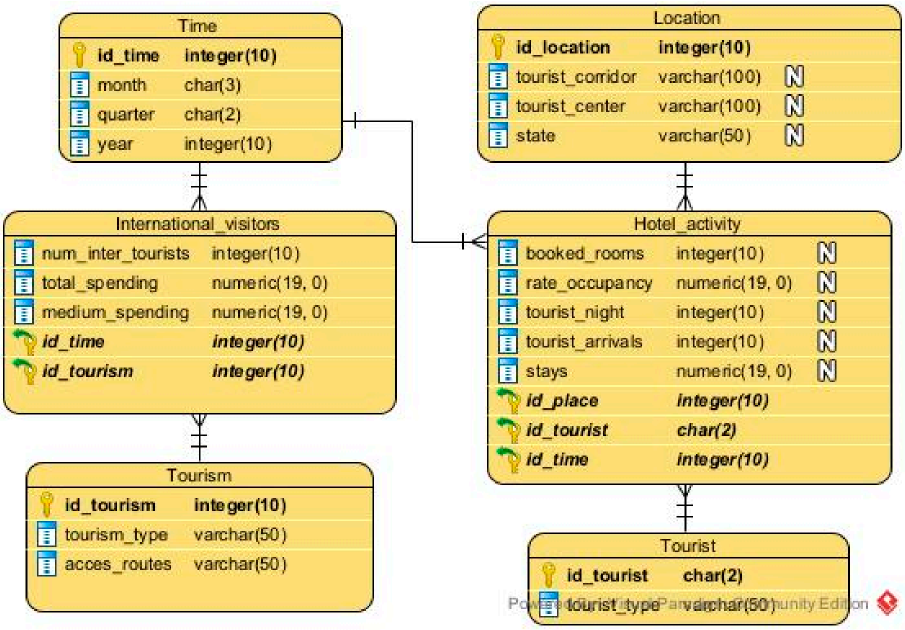


Figure 5. The multidimensional model of the tourist DW.

Table 9. Tables of a tourist DW.

Name	Type	Attributes	Tuples
Hotel_activity	Facts	8	35,424
Internationals_visitors	Facts	5	576
Location	Dimension	4	123
Time	Dimension	4	144
Tourism	Dimension	3	4
Tourist	Dimension	2	2

4.2. Application of FTree in the Tourist Data Warehouse

FTree was applied and tested in the tourist DW. The fact table fragmented was *Hotel_activity*. We performed the first experiment using the predicate use matrix (PUM) of Table 10.

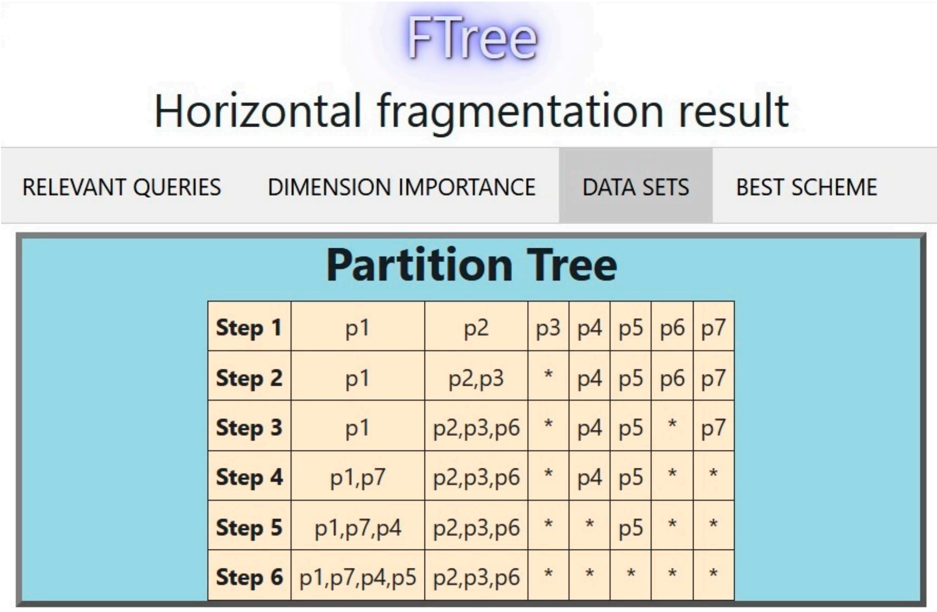
Table 10. Predicate usage matrix for the first experiment.

Q/Pr	p_1	p_2	p_3	p_4	p_5	p_6	p_7	F_i
q_1	1	1	1	0	0	0	0	20
q_2	1	0	0	1	0	0	0	15
q_3	1	0	0	0	1	0	0	15
q_4	0	1	0	0	0	1	0	35
q_5	0	1	1	0	0	0	0	35
q_6	1	0	0	0	0	0	1	30
q_7	0	0	0	1	0	0	0	10
q_8	0	0	0	0	0	0	1	10
q_9	0	0	0	0	1	0	0	10
q_{10}	0	0	1	0	0	0	0	10
sel_{j}	2952	8856	2952	2952	2952	2952	2952	

In this case, the upper limit of partitions (W) allowed by the DWA is 4. The predicates considered are presented in Table 11. Using the PUM of Table 10, the partition tree of Figure 6 was obtained by FTree (every column represents a fragment and the * indicates an empty partition). Table 12 depicts the comparison between $data\_set_2$ with fragments $fr_1 = \{p_1, p_4, p_5, p_7\}, fr_2 = \{p_2, p_3, p_6\}$ and $data\_set_3$ with fragments $fr_1 = \{p_1, p_4, p_7\}$, $fr_2 = \{p_2, p_3, p_6\}, fr_3 = \{p_5\}$. We used 8-fold cross-validation. The best scheme is $data\_set_2$. The decision tree obtained by FTree is visualized in Figure 7.

Table 11. Predicates for the first experiment.

Pr	**Pr** **Text**
p_1	*month* = 'feb'
p_2	*quarter* = 'T3'
p_3	year = 2018
p_4	year = 2014
p_5	*year* = 2017
p_6	*year* = 2015
p_7	year = 2016



Partition Tree

Step 1	p1	p2	p3	p4	p5	p6	p7
Step 2	p1	p2,p3	*	p4	p5	p6	p7
Step 3	p1	p2,p3,p6	*	p4	p5	*	p7
Step 4	p1,p7	p2,p3,p6	*	p4	p5	*	*
Step 5	p1,p7,p4	p2,p3,p6	*	*	p5	*	*
Step 6	p1,p7,p4,p5	p2,p3,p6	*	*	*	*	*

Figure 6. PT for the first experiment.

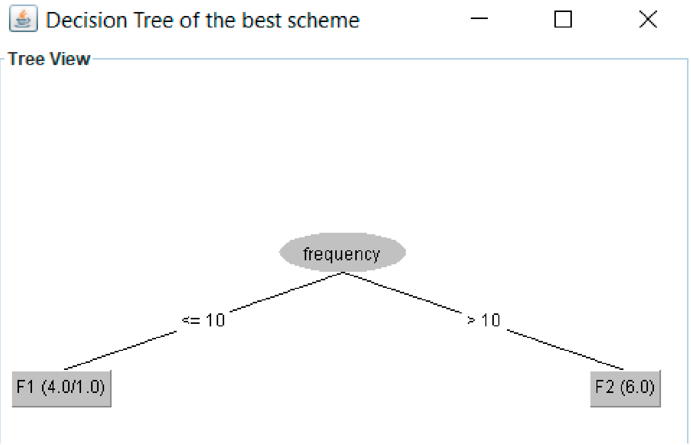


Figure 7. Decision tree for 2 fragments found by FTree for the first experiment.

Other classification algorithms were used to select the fragmentation scheme. Table 12 presents the results of the evaluation metrics precision (P), recall (R), F-measure (F), and ROC area (ROC) with naïve Bayes and multilayer perceptron with one hidden layer with six units, sigmoid activation function, learning rate = 0.3, momentum = 0.2, and number of epochs = 500.

Table 12. Comparison of the two schemes with J48, Naïve Bayes, and Multi-layer Perceptron.

Scheme	J48				Naïve Bayes				Multi-Layer Perceptron			
	P	R	F-M	ROC	P	R	F-M	ROC	P	R	F-M	ROC
data_set$_2$	**0.925**	**0.900**	**0.903**	**0.881**	**0.925**	**0.900**	**0.903**	**0.952**	**0.880**	**0.800**	**0.808**	**0.857**
data_set$_3$	-	0.600	-	0.439	-	0.800	-	0.774	0.750	0.600	0.650	0.694

As seen in Table 12, all the algorithms agree that the two-fragment scheme is the most suitable. In addition, the model obtained by FTree (Figure 7) is easier to interpret, which allows the fragmentation of the data warehouse based on the attribute *frequency*.

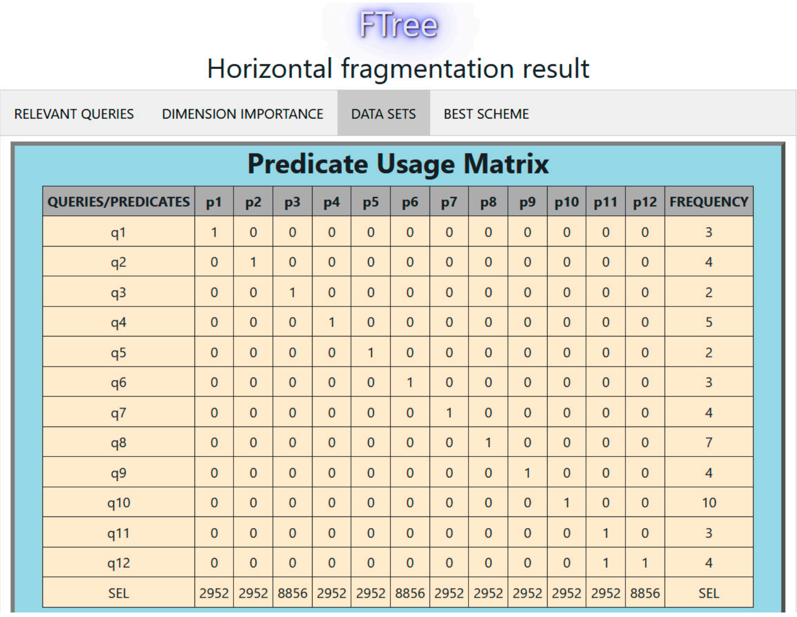
We performed a second experiment. Table 13 presents the OLAP queries executed in the tourist DW. The PUM for this workload is shown in Figure 8.

Table 13. OLAP queries executed in the tourist DW in the second experiment.

Query	Frequency
q_1: SELECT month, sum(booked_rooms) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.year=2007 GROUP BY month;	3
q_2: SELECT month, sum(booked_rooms) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.year=2008 GROUP BY month;	4
q_3: SELECT month, sum(booked_rooms) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.quarter='T1′ GROUP BY month;	2
q_4: SELECT month, sum(tourist_night) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.year=2009 GROUP BY month;	5
q_5: SELECT quarter, sum(tourist_night) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.year=2010 GROUP BY quarter;	2
q_6: SELECT year, sum(tourist_night) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.quarter='T2′ group by year;	3
q_7: SELECT month, sum(tourist_arrivals) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.year=2011 group by month;	4
q_8: SELECT year, sum(tourist_arrivals) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.month='dic′ group by year;	7
q_9: SELECT quarter, sum(tourist_arrivals) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.year=2012 group by quarter;	4

Table 13. *Cont.*

Query	Frequency
q_{10}: SELECT year, avg(stays) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.month='jul' group by year;	10
q_{11}: SELECT quarter, avg(stays) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.year=2013 group by quarter;	3
q_{12}: SELECT month, avg(stays) FROM time, hotel_activity WHERE time.id_time=hotel_activity.id_time AND time.year=2013 AND time.quarter='T4' group by month;	4

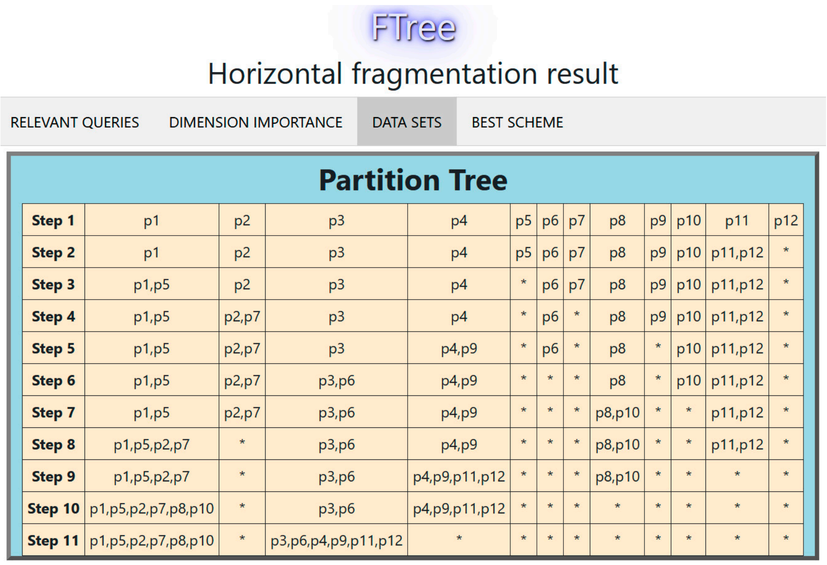


Predicate Usage Matrix

QUERIES/PREDICATES	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	FREQUENCY
q1	1	0	0	0	0	0	0	0	0	0	0	0	3
q2	0	1	0	0	0	0	0	0	0	0	0	0	4
q3	0	0	1	0	0	0	0	0	0	0	0	0	2
q4	0	0	0	1	0	0	0	0	0	0	0	0	5
q5	0	0	0	0	1	0	0	0	0	0	0	0	2
q6	0	0	0	0	0	1	0	0	0	0	0	0	3
q7	0	0	0	0	0	0	1	0	0	0	0	0	4
q8	0	0	0	0	0	0	0	1	0	0	0	0	7
q9	0	0	0	0	0	0	0	0	1	0	0	0	4
q10	0	0	0	0	0	0	0	0	0	1	0	0	10
q11	0	0	0	0	0	0	0	0	0	0	1	0	3
q12	0	0	0	0	0	0	0	0	0	0	1	1	4
SEL	2952	2952	8856	2952	2952	8856	2952	2952	2952	2952	2952	8856	SEL

Figure 8. PUM for the second experiment with the tourist DW.

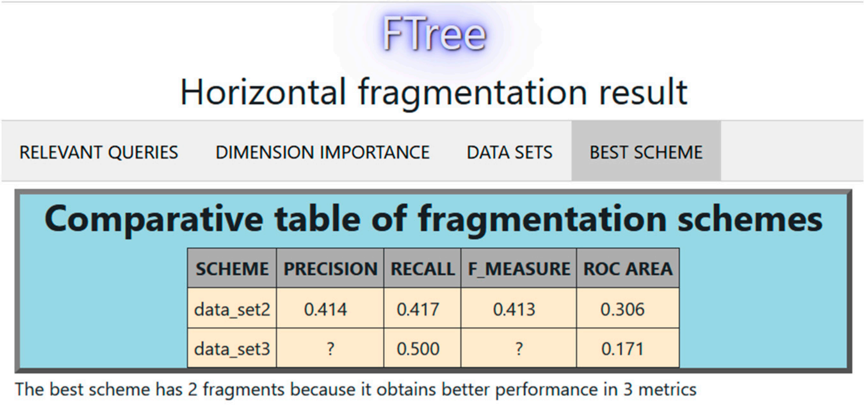
The partition tree for this experiment is shown in Figure 9 (each column represents a fragment, the * indicates an empty partition). In this case, W = 4; therefore, two data sets were obtained. Figure 10 compares the performance of these data sets using 10-fold cross-validation. Figure 11 displays the decision tree for the best fragmentation scheme. It has two fragments $fr_1 = \{p_1, p_2, p_5, p_7, p_8, p_{10}\}$, $fr_2 = \{p_3, p_4, p_6, p_9, p_{11}, p_{12}\}$.



Partition Tree

Step 1	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12
Step 2	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11,p12	*
Step 3	p1,p5	p2	p3	p4	*	p6	p7	p8	p9	p10	p11,p12	*
Step 4	p1,p5	p2,p7	p3	p4	*	p6	*	p8	p9	p10	p11,p12	*
Step 5	p1,p5	p2,p7	p3	p4,p9	*	p6	*	p8	*	p10	p11,p12	*
Step 6	p1,p5	p2,p7	p3,p6	p4,p9	*	*	*	p8	*	p10	p11,p12	*
Step 7	p1,p5	p2,p7	p3,p6	p4,p9	*	*	*	p8,p10	*	*	p11,p12	*
Step 8	p1,p5,p2,p7	*	p3,p6	p4,p9	*	*	*	p8,p10	*	*	p11,p12	*
Step 9	p1,p5,p2,p7	*	p3,p6	p4,p9,p11,p12	*	*	*	p8,p10	*	*	*	*
Step 10	p1,p5,p2,p7,p8,p10	*	p3,p6	p4,p9,p11,p12	*	*	*	*	*	*	*	*
Step 11	p1,p5,p2,p7,p8,p10	*	p3,p6,p4,p9,p11,p12	*	*	*	*	*	*	*	*	*

Figure 9. Partition tree for the second experiment with the tourist DW.



FTree

Horizontal fragmentation result

RELEVANT QUERIES | DIMENSION IMPORTANCE | DATA SETS | BEST SCHEME

Comparative table of fragmentation schemes

SCHEME	PRECISION	RECALL	F_MEASURE	ROC AREA
data_set2	0.414	0.417	0.413	0.306
data_set3	?	0.500	?	0.171

The best scheme has 2 fragments because it obtains better performance in 3 metrics

Figure 10. Comparison of fragmentation schemes with 2 and 3 fragments.

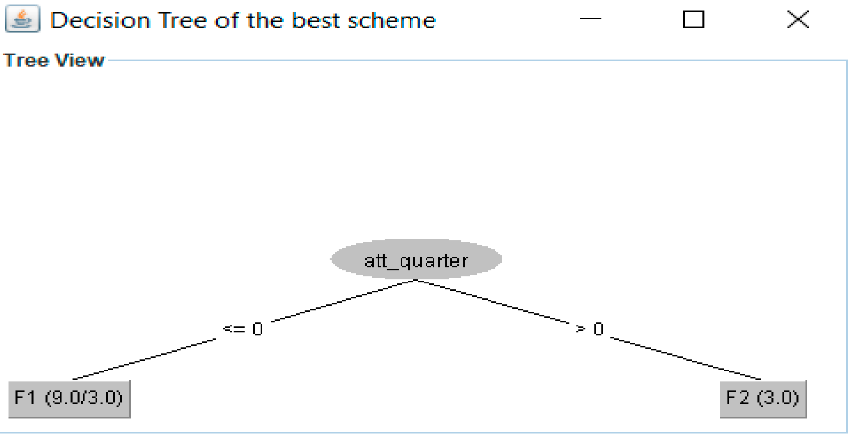


Figure 11. Decision tree for 2 fragments found by FTree for the second experiment.

Other classification techniques were used to select the fragmentation scheme. Table 14 presents the results of the evaluation metrics with naïve Bayes and multilayer perceptron with one hidden layer with six units, sigmoid activation function, learning rate = 0.3, momentum = 0.2, and number of epochs = 500.

Table 14. Comparison of the two schemes with Naïve Bayes and Multi-layer Perceptron.

Scheme	Naïve Bayes				Multi-Layer Perceptron			
	Precision	Recall	F-Measure	ROC Area	Precision	Recall	F-Measure	ROC Area
data_set$_2$	0.800	0.667	0.625	0.708	**0.688**	**0.667**	**0.657**	0.694
data_set$_3$	**0.857**	**0.750**	**0.742**	**0.750**	0.563	0.583	0.570	**0.733**

The multilayer perceptron selected the same fragmentation scheme as FTree, while naïve Bayes chose the scheme with three fragments. We compared the schemes using a cost model where the cost of a decision support query q_i is given by the irrelevant tuple cost (ITC) and the relevant tuple cost (RTC), as shown in Equation (2).

$$Cost(q_i) = ITC(q_i) + RTC(q_i) \tag{2}$$

The amount of irrelevant data accessed from a query is calculated by multiplying the number of irrelevant tuples accessed by the query in every fragment fr_k by the query frequency (f_i). Equation (3) is used to obtain ITC where $card(fr_k)$ is the cardinality of the fragment fr_k, sel_j is the selectivity of the predicate p_j, and rt_j is the number of tuples from the predicate p_j located in other fragments required to answer the query.

$$ITC(q_i) = \sum_{k=1}^{m}(card(fr_k) - \sum_{p_j|PUM(q_i,p_j)=1 \wedge p_j \in fr_k}(sel_j - rt_j)) * f_i \tag{3}$$

The amount of relevant data located in other fragments is obtained by Equation (4), where the number of relevant tuples located in other fragments is multiplied by the frequency of the query squared and by the number of fragments (nf_i) required to answer the query.

$$RTC(q_i) = \sum_{p_j|PUM(q_i,\ p_j)=1 \wedge rt_j>0}^{n} rt_j * f_i^2 * nf_i \tag{4}$$

Table 15 shows the cost of the queries in both fragmentation schemes, the one selected by FTree with J48 and the multilayer perceptron, and the scheme chosen by naïve Bayes. In the case of the first scheme, the cardinalities of the fragments are $card(fr_1)$ = 15,744, $card(fr_2)$ = 17,220, and $card(fr_3)$ = 2460. The third fragment is the complement. The ITC of q_3 is calculated as follows:

$$ITC(q_3) = (card(fr_2) - (sel_3 - rt_3)) * f_3 = (17,220 - (8856 - 2952)) * 2 = 22,632$$

The RTC of the same query is obtained as shown below:

$$RTC(q_3) = rt_3 * f_3^2 * nf_3 = 2952 * 2^2 * 2 = 23,616.$$

For the scheme chosen by naïve Bayes, the cardinalities of the fragments are $card(fr_1)$ = 15,744, $card(fr_2)$ = 11,808, $card(fr_3)$ = 5412, and $card(fr_4)$ = 2460. The fourth fragment is the complement. Therefore, the costs of the third query in this scheme are:

$$ITC(q_3) = (card(fr_2) - (sel_3 - rt_3)) * f_3 = (11,808 - (8856 - 2952)) * 2 = 11,808.$$
$$RTC(q_3) = rt_3 * f_3^2 * nf_3 = 2952 * 2^2 * 2 = 23,616.$$

Table 15. Query cost comparison of the fragmentation schemes.

Query	HFS of *data_set*$_2$			HFS of *data_set*$_3$		
	ITC	**RTC**	**Cost**	**ITC**	**RTC**	**Cost**
q_1	38,376	0	38,376	38,376	0	38,376
q_2	51,168	0	51,168	51,168	0	51,168
q_3	22,632	23,616	46,248	11,808	23,616	**35,424**
q_4	73,800	24,600	**98,400**	22,140	147,600	169,740
q_5	25,584	0	25,584	25,584	0	25,584
q_6	33,948	53,136	87,084	17,712	53,136	**70,848**
q_7	51,168	0	51,168	51,168	0	51,168
q_8	89,544	0	89,544	89,544	0	89,544
q_9	59,040	15,744	**74,784**	17,712	94,464	112,176
q_{10}	127,920	0	127,920	127,920	0	127,920
q_{11}	44,280	8856	**53,136**	13,284	53,136	66,420
q_{12}	43,296	173,184	**216,480**	1968	330,624	332,592

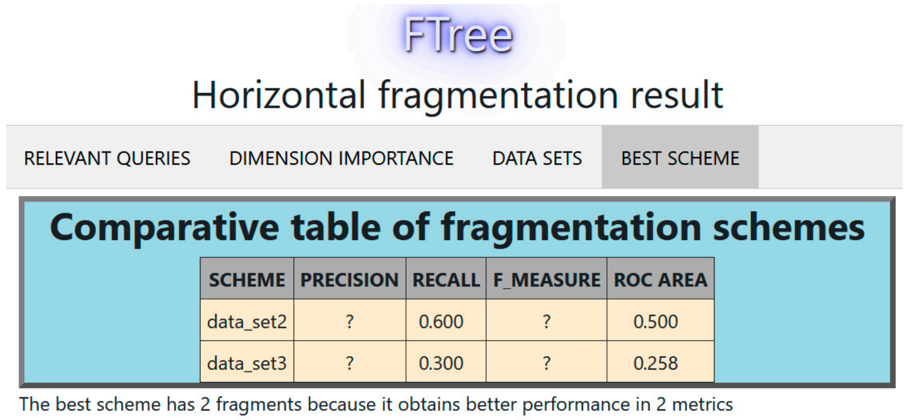
Table 15 shows that the HFS selected by FTree with J48 and the multilayer perceptron is better in most cases. In only two queries, the HFS chosen by naïve Bayes obtained a lower query execution cost. Additionally, the decision tree generated by J48 is easier to interpret than the model of the multilayer perceptron. Therefore, the model obtained by J48 outperformed those of naive Bayes and the multilayer perceptron.

Additionally, to evaluate FTree, it was compared with [14]; for this, the predicate use matrix of Table 16 taken from [14] was used.

Table 16. Predicate use matrix for the third experiment.

Q/P	p_1	p_2	p_3	p_4	p_5	p_6	p_7	F_i
q_1	0	1	0	0	1	0	1	20
q_2	1	1	0	0	0	1	0	15
q_3	0	0	1	0	1	0	1	20
q_4	0	1	0	1	0	1	0	15
q_5	0	0	1	0	1	0	1	15
q_6	1	0	0	1	0	1	1	10
q_7	0	1	0	1	0	1	0	15
sel_j	30	50	80	65	20	35	40	

The best fragmentation scheme according to [14] is that of the three fragments $fr_1 = \{p_1, p_2, p_3, p_4, p_5\}$, $fr_2 = \{p_6\}$, and $fr_3 = \{p_7\}$. With FTree, in this case, the maximum number of fragments allowed by the data warehouse manager (W) is four. Using the PUM, the partition tree generated by FTree was obtained. Figure 12 shows the comparison between data_set$_2$ with the fragments $fr_1 = \{p_1, p_2, p_4, p_6\}$, $fr_2 = \{p_3, p_5, p_7\}$, and $data\_set_3$ with the fragments $fr_1 = \{p_1, p_2, p_4\}$, $fr_2 = \{p_3, p_5, p_7\}$, $fr_3 = \{p_6\}$. We used 8-fold cross-validation. The best scheme is $data\_set_2$. The decision tree obtained with FTree is visualized in Figure 13.



Comparative table of fragmentation schemes

SCHEME	PRECISION	RECALL	F_MEASURE	ROC AREA
data_set2	?	0.600	?	0.500
data_set3	?	0.300	?	0.258

The best scheme has 2 fragments because it obtains better performance in 2 metrics

Figure 12. Comparison of fragmentation schemes with 2 and 3 fragments for the third experiment.

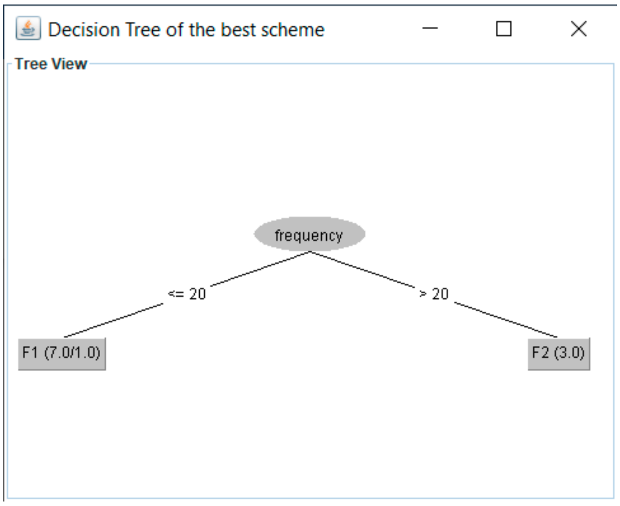


Figure 13. Decision tree for 2 fragments found by FTree in the third experiment.

Table 17 shows the cost of the queries in two fragmentation schemes, that obtained by [14] and that of FTree. In the case of the schema generated by FTree, the cardinalities of the first and second fragments are $card(fr_1) = 180$ and $card(fr_2) = 140$. For the scheme produced by [14], the cardinalities of the fragments are $card(fr_1) = 245$, $card(fr_2) = 35$, and $card(fr_3) = 40$.

Table 17. Comparison of query costs for the two fragmentation schemes.

Query		FTree			Kechar & Nait-Bahloul [14]	
	ITC	RTC	Cost	ITC	RTC	Cost
q_1	4200	0	4200	3500	0	**3500**
q_2	975	0	**975**	2475	0	2475
q_3	0	0	**0**	2900	0	2900
q_4	450	0	**450**	1950	0	1950
q_5	0	0	**0**	2175	0	2175
q_6	1500	0	1500	1500	0	1500
q_7	450	0	**450**	1950	0	1950

As can be seen in Table 17, although both fragmentation methods obtained schemes without RTC, the scheme generated by FTree had lower ITC in most cases. Only the first query was more efficient in the scheme found by [14]. These experiments demonstrated the effectiveness of FTree.

5. Conclusions and Future Work

Technological advances have made it possible to collect large amounts of data in different fields; therefore, there is a constant need to develop tools that help the process of extracting precise information from the enormous volume of data that are converted into knowledge. The DWs have proven to be efficient for the analysis of these massive data; for this reason, they are an excellent option to be implemented in the monitoring of tourist activity, in which, to stay at a competitive level, it is necessary to know the client preferences. As mentioned throughout this article, DW fragmentation achieves optimized response time and execution costs of OLAP queries. Several techniques for both vertical and horizontal fragmentation have been developed; however, to the best of our knowledge, a decision tree has not been used to fragment the DW, which represents an unexplored area within DW research. In this work, the classification capability of decision trees is exploited to use them in DW horizontal fragmentation. A horizontal fragmentation method, called FTree, has been developed that uses decision trees to obtain fragmentation schemes that achieve the optimization of OLAP queries in a DW. FTree was applied in a tourist DW, taking advantage of a large amount of data available in this area. The results will benefit both designers and users of data warehouses as well as researchers from the computational area.

In the future, we will extend FTree to provide dynamic fragmentation of data warehouses, monitoring the access patterns of the DW to detect when to modify the horizontal fragmentation scheme to avoid the reduction in query performance. Additionally, the fragmentation of data lakes will be considered.

Author Contributions: Conceptualization, L.R.-M., N.R.-M. and A.L.-C.; formal analysis, N.R.-M.; investigation, N.R.-M. and L.R.-M.; methodology, L.R.-M., G.A.-H. and N.R.-M.; data curation, N.R.-M.; visualization, N.R.-M.; funding acquisition, L.R.-M. and G.A.-H.; resources, L.R.-M. and I.M.-C.; writing—original draft, N.R.-M.; writing—review & editing, L.R.-M., G.A.-H., I.M.-C. and G.A.-H.; project administration, L.R.-M., A.L.-C. and I.M.-C.; supervision, L.R.-M. and A.L.-C.; validation, L.R.-M. All authors have read and agreed to the published version of the manuscript.

Funding: The National Council of Science and Technology (CONACYT) as well as the Public Secretariat of Education (SEP) financed this project through the Sectorial Fund of Research for Education, grant number A1-S-51808.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study is available on request from the corresponding author.

Acknowledgments: The authors thank the National Technological of Mexico (TecNM) for supporting this research. In addition, the National Council of Science and Technology (CONACYT) and the Public Secretariat of Education (SEP) patronized this project.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ozsu, M.T.; Valduriez, P. *Principles of Distributed Database Systems, 4th ed*; Springer Nature Switzerland AG: Cham, Switzerland, 2020; p. 768.
2. Daniel, C.; Salamanca, E.; Nordlinger, B. Hospital Databases: AP-HP Clinical Data Warehouse. In *Healthcare and Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 57–67.
3. Melton, J.E.; Go, S.; Zilliac, G.G.; Zhang, B.Z. *Greenhouse Gas Emission Estimations for 2016–2020 using the Sherlock Air Traffic Data Warehouse*; Report NASA/TM-202220007609; NASA: Washington, DC, USA, 2022.
4. Janzen, T.J.; Ristino, L. *USDA and Agriculture Data: Improving Productivity while Protecting Privacy*; SSRN: Washington, DC, USA, 2018.
5. Han, J.; Kamber, M.; Pei, J. *Data Mining Concepts and Techniques, 3rd ed*; Morgan Kaufmann Publishers: Burlington, MA, USA, 2012; p. 703.
6. Furtado, P. Experimental Evidence on Partitioning in Parallel Data Warehouses. In Proceedings of the 7th ACM International Workshop on Data Warehousing and OLAP, Washington, DC, USA, 12–13 November 2004; pp. 23–30.
7. Kimball, R.; Ross, M.; Thornthwaite, W.; Mundy, J.; Becker, B. *The Data Warehouse Lifecycle Toolkit*, 2nd ed.; Wiley Publishing, Inc.: Indianapolis, Indiana, 2008; p. 636.
8. Noaman, A.Y.; Barker, K. A Horizontal Fragmentation Algorithm for the Fact Relation in a Distributed Data Warehouse. In Proceedings of the Eighth International Conference on Information and Knowledge Management, CIKM '99, Kansas City, MI, USA, 2–6 November 1999; pp. 154–161.
9. Ramdane, Y.; Kabachi, N.; Boussaid, O.; Bentayeb, F. SDWP: A New Data Placement Strategy for Distributed Big Data Warehouses in Hadoop. In *Big Data Analytics and Knowledge Discovery*; Ordonez, C., Song, I.Y., Anderst-Kotsis, G., Tjoa, A.M., Khalil, I., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 189–205.
10. Curino, C.; Jones, E.; Zhang, Y.; Madden, S. Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.* **2010**, *3*, 48–57. [CrossRef]
11. Mahboubi, H.; Darmont, J. Data mining-based fragmentation of XML data warehouses. In Proceedings of the ACM 11th international workshop on Data warehousing and OLAP-DOLAP '08, Napa Valley, CA, USA, 30 October 2008; ACM Press: New York, NY, USA, 2008; p. 9. Available online: http://portal.acm.org/citation.cfm?doid=1458432.1458435 (accessed on 12 September 2022).
12. Barr, M.; Boukhalfa, K.; Bouibede, K. Bi-Objective Optimization Method for Horizontal Fragmentation Problem in Relational Data Warehouses as a Linear Programming Problem. *Appl. Artif. Intell.* **2018**, *32*, 907–923. [CrossRef]
13. Liu, J.Y.C.; Wang, C.W.; Chan, C.Y. An Efficient Partitioning for Object-Relational Data Warehouses. *Appl. Mech. Mater.* **2013**, *284–287*, 3320–3324. [CrossRef]
14. Kechar, M.; Nait-Bahloul, S. Performance optimisation of the decision-support queries by the horizontal fragmentation of the data warehouse. *Int. J. Bus. Inf. Syst.* **2017**, *26*, 506. [CrossRef]
15. Kechar, M.; Nait-Bahloul, S. Bringing Together Physical Design and Fast Querying of Large Data Warehouses: A New Data Partitioning Strategy. In Proceedings of the 4th International Conference on Big Data and Internet of Things, Rabat Morocco, 23–24 October 2019; Association for Computing Machinery: New York, NY, USA, 2019.
16. Ramdane, Y.; Boussaid, O.; Kabachi, N.; Bentayeb, F. Partitioning and Bucketing Techniques to Speed up Query Processing in Spark-SQL. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 142–151. Available online: https://ieeexplore.ieee.org/document/8644891/ (accessed on 12 September 2022).
17. Parchas, P.; Naamad, Y.; Bouwel, P.V.; Faloutsos, C.; Petropoulos, M. Fast and effective distribution-key recommendation for amazon redshift. *Proc. VLDB Endow.* **2020**, *13*, 2411–2423. [CrossRef]
18. Barkhordari, M.; Niamanesh, M. Chabok: A Map-Reduce based method to solve data warehouse problems. *J. Big. Data.* **2018**, *5*, 1–25. [CrossRef]
19. Bellatreche, L.; Boukhalfa, K.; Richard, P. Data Partitioning in Data Warehouses: Hardness Study, Heuristics and ORACLE Validation. In *Data Warehousing and Knowledge Discovery*; Song, I.Y., Eder, J., Nguyen, T.M., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 87–96. Available online: http://link.springer.com/10.1007/978-3-540-85836-2_9 (accessed on 12 September 2022).
20. Barr, M.; Bellatreche, L. A New Approach Based on Ants for Solving the Problem of Horizontal Fragmentation in Relational Data Warehouses. In Proceedings of the 2010 International Conference on Machine and Web Intelligence, Algiers, Algeria, 3–5 October 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 411–415. Available online: http://ieeexplore.ieee.org/document/5648104/ (accessed on 12 September 2022).

21. Ramdane, Y.; Kabachi, N.; Boussaid, O.; Bentayeb, F. SkipSJoin: A New Physical Design for Distributed Big Data Warehouses in Hadoop. In *Conceptual Modeling*; Laender, A.H.F., Pernici, B., Lim, E.P., de Oliveira, J.P.M., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 255–263.
22. Ettaoufik, A.; Ouzzif, M. Web Service for Incremental and Automatic Data Warehouses Fragmentation. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 1–10.
23. Soussi, N. Big-Parallel-ETL: New ETL for Multidimensional NoSQL Graph Oriented Data. *J. Phys. Conf. Ser.* **2021**, *1743*, 012037. [CrossRef]
24. Munerman, V.; Munerman, D.; Samoilova, T. The Heuristic Algorithm for Symmetric Horizontal Data Distribution. In Proceedings of the 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). St. Petersburg, Moscow, Russia, 26–29 January 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 2161–2165.
25. Ramdane, Y.; Kabachi, N.; Boussaid, O.; Bentayeb, F. A Data Mining Approach to Guide the Physical Design of Distributed Big Data Warehouses. In *Advances in Knowledge Discovery and Management: Volume 9*; Jaziri, R., Martin, A., Rousset, M.C., Boudjeloud-Assala, L., Guillet, F., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 107–125. [CrossRef]
26. Ramdane, Y.; Boussaid, O.; Boukraà, D.; Kabachi, N.; Bentayeb, F. Building a novel physical design of a distributed big data warehouse over a Hadoop cluster to enhance OLAP cube query performance. *Parallel. Comput.* **2022**, *111*, 102918. [CrossRef]
27. O'neil, P.; O'neil, B.; Chen, X. *The Star Schema Benchmark (SSB)*; UMass: Boston, MA, USA, 2009.
28. Rodríguez-Mazahua, N.; Rodríguez-Mazahua, L.; López-Chau, A.; Alor-Hernández, G.; Peláez-Camarena, S.G. Comparative Analysis of Decision Tree Algorithms for Data Warehouse Fragmentation. In *New Perspectives on Enterprise Decision-Making Applying Artificial Intelligence Techniques*; Zapata-Cortes, J.A., Alor-Hernández, G., Sánchez-Ramírez, C., García-Alcaraz, J.L., Eds.; Springer International Publishing: Cham, Switzerland, 2021; Volume 966, pp. 337–363. [CrossRef]
29. Son, J.H.; Kim, M.H. An adaptable vertical partitioning method in distributed systems. *J. Syst. Softw.* **2004**, *73*, 551–561. [CrossRef]
30. Rodríguez, L.; Alor-Hernández, G.; Abud-Figueroa, M.A.; Peláez-Camarena, S.G. Horizontal Partitioning of Multimedia Databases Using Hierarchical Agglomerative Clustering. In Proceedings of the Mexican International Conference on Artificial Intelligence, MICAI 2014: Nature-Inspired Computation and Machine Learning, Tuxtla, Mexico, 16–22 November 2014; Springer: Cham, Switzerland, 2014; pp. 296–309.
31. Anitha, S.; Vanitha, M. Classification of VASA Dataset Using J48, Random Forest, and Naive Bayes. In *Ntelligent Data Engineering and Analytics Smart Innovation, Systems, and Technologies*; Satapathy, S.C., Ed.; Springer: Berlin/Heidelberg, Germany, 2022. [CrossRef]
32. Razdan, S.; Gupta, H.; Seth, A. Performance Analysis of Network Intrusion Systems using J48 and Naive Bayes Algorithm. In Proceedings of the 6th International Conference for Convergence in Technology (I2CT), Maharashtra, India, 2–4 April 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–7.
33. Tan, P.N.; Steinbach, M.; Karpatne, A.; Kumar, V. *Introduction to Data Mining*, 2nd ed.; Pearson: New York, NY, USA, 2019; p. 839.
34. Kimball, R.; Ross, M. *The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence*, 2nd ed.; John Wiley & Sons, Inc.: Indianapolis, Indiana, 2016; p. 744.