Mathware & Soft Computing 11 (2004) 51-66

Data Mining Techniques Using Decision Tree Model in Materialised Projection and Selection View

Y.W. Teh
Fac. of Computer Science and Information Technology,
University of Malaya, 50603, Kuala Lumpur, Malaysia
tehyw@um.edu.my

Abstract

With the availability of very large data storage today, redundant data structures are no longer a big issue. However, an intelligent way of managing materialised projection and selection views that can lead to fast access of data is the central issue dealt with in this paper. A set of implementation steps for the data warehouse administrators or decision makers to improve the response time of queries is also defined. The study concludes that both attributes and tuples, are important factors to be considered to improve the response time of a query. The adoption of data mining techniques in the physical design of data warehouses has been shown to be useful in practice.

1 Introduction

For business organisations, going digital is no longer an option but a necessity. Large business organisations need huge amount of digital data storage to run their operations.

Decision makers or data warehouse administrators handle very large data storage. They have to address four main issues:

1. Data warehouse performance tuning is an essential part of the management and administration of a decision-support system. The commercial databases (e.g. Oracle, DB2, Microsoft SQL Server 7, etc.) use data warehouse performance tuning process to improve the response time of a query. Data warehouse performance tuning process is a built-in feature of most of the commercial databases. Inexperienced decision makers and data warehouse administrators have difficulties in understanding the process of data warehouse performance tuning.

2. Only a few decision makers or data warehouse administrators can do justice to the task of picking the right redundant data structures for the data warehouse [10]. The decision makers or data warehouse administrators lack knowledge in understanding the query processing techniques.

3. The proprietary database does not have open-source database architecture, and this makes it impossible for the decision makers or data warehouse administrators to optimise the performance of the query processing techniques directly. There is a strong need to have a tool to interact with the current proprietary database.

4. Most decision makers or data warehouse administrators lack knowledge in data mining tools. It is difficult for them to make use of current technologies to improve the response time of queries. As such, the decision makers or data warehouse administrators need data mining tool with a simple user interface to interact with the data warehouse.

2 Literature Review

Decision-support queries typically involve several joins, a grouping with aggregation, and/or sorting of the resulting tuples of relations. Figure 1 shows a simple star scheme that resulted from multidimensional modelling. Facts correspond to SALES data, and CUSTOMER, TIME, PRODUCT are dimensions of SALES. The data stored in specialised relational tables provide a multidimensional view of the data by using the relational technology as an underlying data model. The specialised relational tables stored in the star scheme have two major advantages:

1. they can be used and easily incorporated into other existing relational database systems, and

2. relational data can be stored more efficiently than multidimensional data [11].

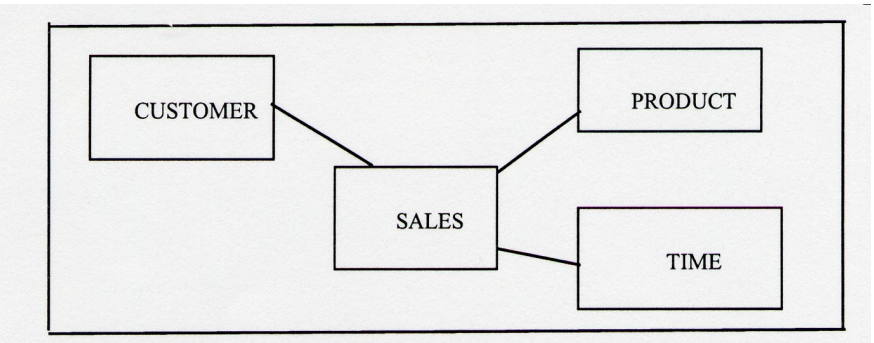


Fig. 1 A Simple Star Schema with Multidimensional Modelling

Data warehouse administrators are responsible for data warehouse performance tuning. There are many tables in a data warehouse. A significant aspect of this research looks at how commercial data warehouses reduce the workload of the data warehouse administrators.

Claussen et al., [9] proposed two classes of query evaluation of such queries. The algorithms are based on:

(1) early sorting, and

(2) early partitioning – or a combination of both.

The idea is to push the sorting and /or the partitioning to the leaves, i.e., the base relations of the query evaluation plans (QEPs) and thereby avoids sorting or partitioning large intermediate results generated by the joins. Ross and Li [8] described two new join algorithms that are based on join indices. One of the algorithms is sorted-based and the other is partitioned-based. Both algorithms benefit from not completely materialising the joined result; rather they store the temporary result on disk in two ordered files to be merged to obtain the joined result.

One of the techniques employed in a data warehouse to improve the response time of a query is the creation of a set of materialised views and indexes [1]. The materialised view and index selection are two problems often studied independently in data warehouses. Studying the problem of views and index separately causes an inefficient distribution of resources (space, computation time, maintenance time, etc.) between materialised views and indexes [2].

The right materialised views can significantly improve performance, particularly for decision-support applications [1]. Both indexes and materialised views are physical structures that can significantly accelerate performance. An effective physical database design tool must therefore take into account the interaction between indexes and materialised views by considering them together to optimise the physical design for the workload on the system [1].

2.1 Microsoft SQL V7 Index Selection Tool - AutoAdmin

Chaudhuri and Narasayya [10] started the AutoAdmin research project at Microsoft Corporation to introduce new techniques to self-tune and self-administer database systems. One of the important duties of physical database design is selecting indexes. Chaudhuri and Narasayya introduced the original technique that they developed in the AutoAdmin project to automate the job of selecting a set of indexes. They used the optimiser's cost estimates to build indexes. Their goal is to select a set of indexes that is appropriate for a given database and workload. A workload consists of a set of SQL data manipulation statements, such as Select, Insert, Delete and Update. The term *configuration* is used to imply a set of indexes. To select the configuration, they must be intelligent to evaluate the relative goodness of any two configurations. Given a configuration and a workload, they used the total of the optimiser's estimated costs for all the SQL statements in the workload as the metrics of goodness [10]. The aim of an index selection tool is to choose a configuration that is best or as close to optimum, as possible. The index selection process is subject to limitations, for example, an upper bound on the number of indexes or storage space. They explained their technique to deal with the problem of selecting a configuration subject to an upper bound on the number of indexes. The strength of AutoAdmin is that it helps the data warehouse administrators to select optimal indexes automatically. Its weakness is that it is a

black box to the data warehouse administrators as automatic tools do everything by themselves without involving a data warehouse administrator. AutoAdmin uses the optimiser's estimated costs for all the SQL statements (Select, Insert, Delete and Update). In a data warehouse, most of the queries are read-only statements (select) rather than update statements (Insert, Delete and Update). If we let the index selection tool (AutoAdmin) to build the index, it might not be suitable for a data warehouse.

2.2 Microsoft SQL 2000's Tuning Wizard

Agrawal, Chaudhuri and Narasayya [1] contended that an effective physical design database tool must take into account the interaction between indexes and materialised views by considering them together to optimise the physical design for the workload on the system. In 2000, they established that, in order to pick a physical design consisting of indexes and materialised views, it is vital to explore over the combined space of indexes and materialised views. The tuning wizard has been implemented in Microsoft SQL Server 2000. The strength of the tuning wizard is that it helps the data warehouse administrators to select optimal indexes and materialised views automatically. Its weakness is that it is a black box to the data warehouse administrators as automatic tools do everything without the involvement of a data warehouse administrator. The tuning wizard uses a workload to recommend the indexes and materialised views. The workload is the logging capability of database systems to capture a trace of queries and updates faced by the system. In a data warehouse, most of the queries are read-only statements (select) rather than update statements (Insert, Delete and Update). Microsoft SQL 2000's tuning wizard and AutoAdmin face the same problem. If the tuning wizard is allowed to build the index or materialised view, it might not be suitable for a data warehouse.

2.3 Current Research in Query Processing Techniques

Current research in query processing techniques comprises either the automatic or non-automatic selection of query processing techniques (Table 1). Both approaches, however, are not suitable for a data warehouse. There are too many parameters to select in data warehouse performance tuning. Microsoft's AutoAdmin and Microsoft SQL 2000's tuning wizard use the optimiser estimated cost for all the SQL statement (Insert, Delete, Update and Select).

Microsoft SQL 2000's tuning wizard is not an open-source software, thus, it is impossible to change the existing codes. Therefore, data mining techniques are proposed as intelligent ways to handle the query processing techniques in this research.

Most researchers apply data mining at the application level of data warehouse [4], [5], [6]. From their work [7], [8], it is known that it is important to start optimisation from the base relation to reduce large, immediate results.

An association-rule is one of data mining techniques. An association-rule involves certain association relationships among a set of objects in a database. In

Non-automatic or automatic selection of indexes and/or materialised views	**Comments**	**Actions**
Data warehouse performance tuning	There are too many parameters to be selected. It is not suitable for inexperienced data warehouse administrators	Microsoft's AutoAdmin
Microsoft's AutoAdmin (index selection tool)	It uses the optimiser's estimated cost for all the SQL statements. It might not be suitable for a data warehouse (read-only statements)	To incorporate the materialised view selection (Microsoft SQL 2000' tuning wizard) as well and modify the estimated cost for selection statements only.
Microsoft SQL 2000's tuning wizard (index and materialised view)	It uses the optimiser's estimated cost for all the SQL statements. It might not be suitable for a data warehouse (read-only statements)	To modify the estimated cost for selection statements only.

Table 1. Current Research in Query Processing Techniques.

this context, sets of association-rules are found at various levels of abstraction from the relevant set(s) of data in a database. The weakness of the association-rule algorithms is that they use the transaction log which relies only on the frequently accessed items. Redundant data structures are built based on multiple criteria (number of bytes, the frequently accessed attributes and multiple-attribute access) rather than only frequently accessed items/attributes. Therefore, the association rule might not be suitable for selection based on multiple criteria to form redundant data structures. Decision tree model is more appropriate for this approach.

In this research, data mining techniques using decision tree model will be applied in the physical design of data warehouses to optimise the base relation rather than data cubes. The physical design highlights how the data is physically stored, and what data structures are used to speed up the response to a query. Fig. 2.shows an architectural overview of the approach used in this research to handle redundant data structures (both materialised projection and selection views). The Select statements constitute the major workload of this architecture; therefore, the architecture used generates redundant data structures that are more suitable for a data warehouse.

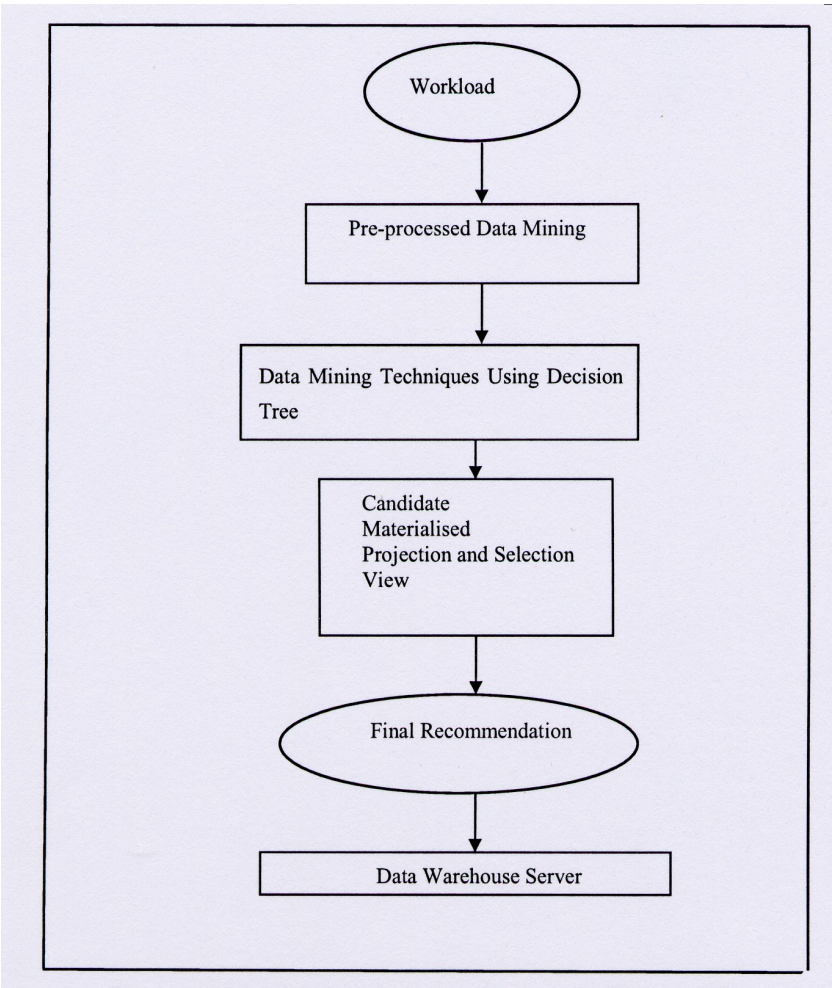


Fig. 2: Architecture of both Materialised Projection and Selection View

3 Data Mining Techniques Using Decision Tree in MPSV

A high priority user's (such as a manager) access log file keeps track of the high priority user decision-support queries from time t_1 to time t_{35}, as shown in Fig. 3.

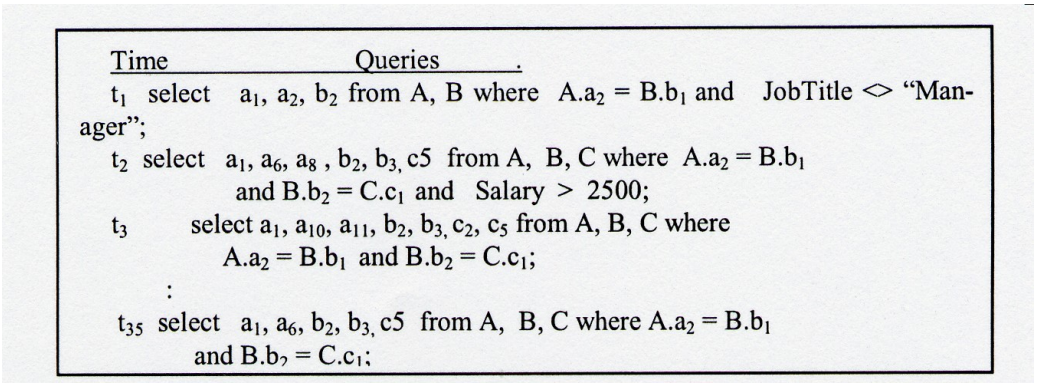


Fig. 3: A High Priority User's Access Log File

Assume that the predicates in the access log file above have the data distribution, as shown in Table 2.

Predicate	Data Distribution
City <> "KL"	60 %
ForeignStaff="No"	90 %
Gender = "Female"	40 %
JobTitle <> "Manager"	90 %
NoOfSkill > 3	70 %
Salary > 2500	20 %
NoOfYearExperience > 2	50 %
NoOfYearJoin > 3	30%

Table 2: Predicates and Data Distribution

The algorithm in Fig. 4 illustrates how to process the access log file and generate the table of frequently accessed attributes. Table 3 is the output generated by the algorithm given in Fig. 4.

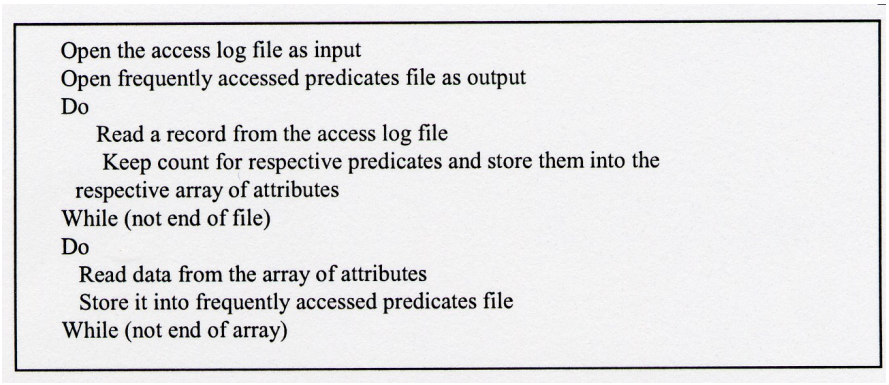


Fig. 4: Algorithm to Generate a Table of Frequently Accessed Predicates

Attribute Names	Frequency of Access F(x)
a_1	7
a_2	9
a_3	4
a_4	6
:	:
:	:
a_{12}	4
a_{13}	4
a_{14}	5
City	3
ForeignStaff	1
Gender	1
JobTitle	1
NoOfSkill	1
YExperience	2
Yjoin	3
Salary	2

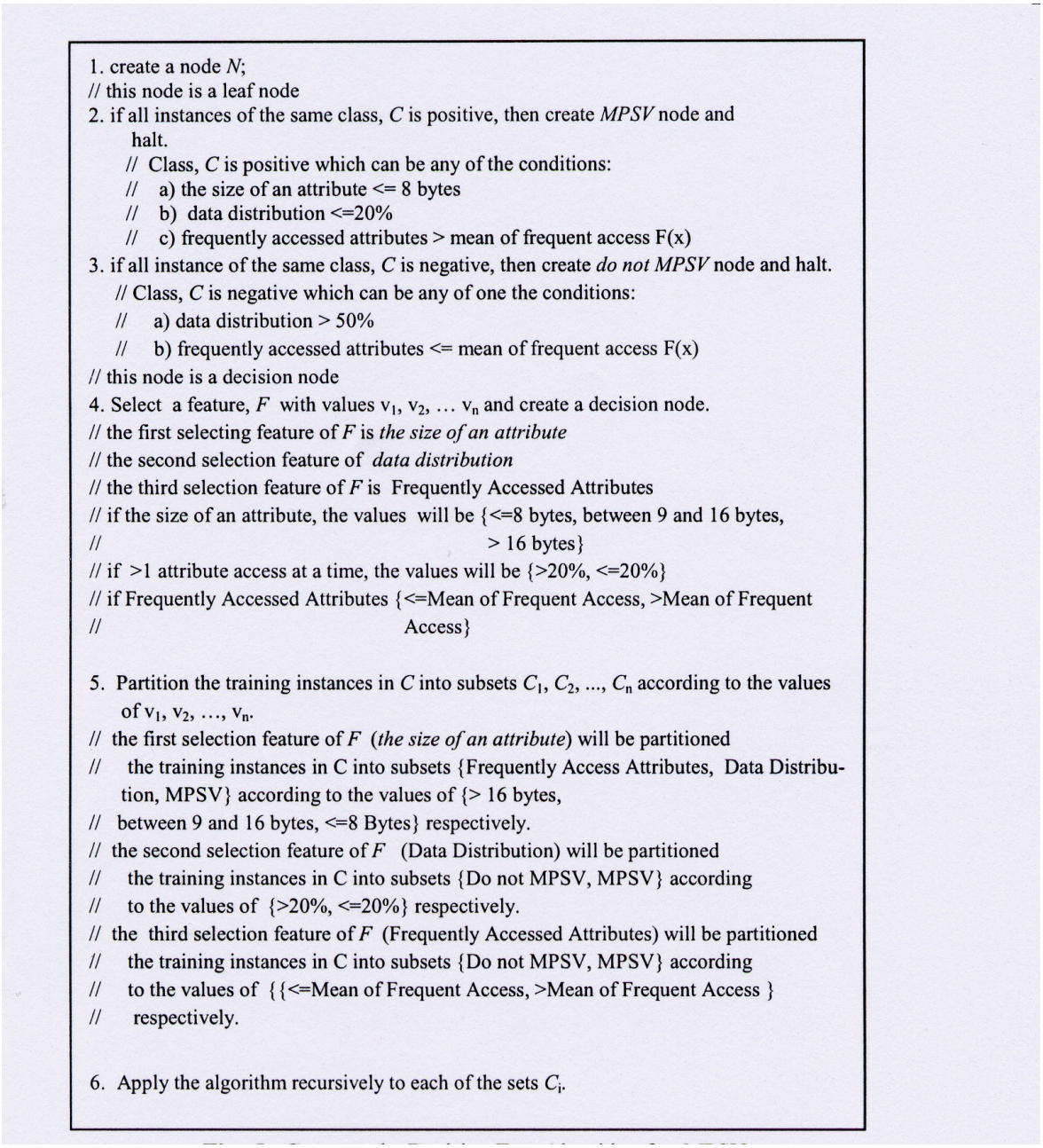
Table 3: Table of Frequently Accessed Attributes

Attribute Names	Len(x)	F(x)	Data Distribution	Class
a_1	30	7	Null	MPSV
a_2	45	9	Null	MPSV
a_3	18	4	Null	MPSV
a_4	24	6	Null	MPSV
a_5	21	1	Null	Do not MPSV
:	:	:	:	:
:	:	:	:	:
a_{13}	11	4	Null	Do not MPSV
a_{14}	16	5	Null	Do not MPSV
City	10	3	60	Do not MPSV
ForeignStaff	3	1	90	MPSV
Gender	1	14	40	MPSV
JobTitle	10	1	90	Do not MPSV
NoOfSkill	2	1	70	MPSV
Salary	4	2	20	MPSV
YExperience	2	3	50	MPSV
YJoin	2	3	30	MPSV

Table 4: MPSV With a Small Training Data Set

Table 4 shows a training data set with four data attributes and two classes.

The Hunt's method for generating an MPSV can be elaborated further as shown in Fig. 5.



1. create a node *N*;
// this node is a leaf node
2. if all instances of the same class, *C* is positive, then create *MPSV* node and halt.
   // Class, *C* is positive which can be any of the conditions:
   // a) the size of an attribute <= 8 bytes
   // b) data distribution <=20%
   // c) frequently accessed attributes > mean of frequent access F(x)
3. if all instance of the same class, *C* is negative, then create *do not MPSV* node and halt.
   // Class, *C* is negative which can be any of one the conditions:
   // a) data distribution > 50%
   // b) frequently accessed attributes <= mean of frequent access F(x)
// this node is a decision node
4. Select a feature, *F* with values $v_1, v_2, \ldots v_n$ and create a decision node.
// the first selecting feature of *F* is *the size of an attribute*
// the second selection feature of *data distribution*
// the third selection feature of *F* is Frequently Accessed Attributes
// if the size of an attribute, the values will be {<=8 bytes, between 9 and 16 bytes,
// > 16 bytes}
// if >1 attribute access at a time, the values will be {>20%, <=20%}
// if Frequently Accessed Attributes {<=Mean of Frequent Access, >Mean of Frequent
// Access}

5. Partition the training instances in *C* into subsets $C_1, C_2, \ldots, C_n$ according to the values of $v_1, v_2, \ldots, v_n$.
// the first selection feature of *F* (*the size of an attribute*) will be partitioned
// the training instances in C into subsets {Frequently Access Attributes, Data Distribution, MPSV} according to the values of {> 16 bytes,
// between 9 and 16 bytes, <=8 Bytes} respectively.
// the second selection feature of *F* (Data Distribution) will be partitioned
// the training instances in C into subsets {Do not MPSV, MPSV} according
// to the values of {>20%, <=20%} respectively.
// the third selection feature of *F* (Frequently Accessed Attributes) will be partitioned
// the training instances in C into subsets {Do not MPSV, MPSV} according
// to the values of {{<=Mean of Frequent Access, >Mean of Frequent Access }
// respectively.

6. Apply the algorithm recursively to each of the sets C_i.

Fig 5: Generate the Decision Tree Algorithm for MPSV

Fig. 6 can be generated by the algorithm in Fig. 5.

Fig. 6 shows how the Hunt's method [12] works with the training data set. A test based on a single attribute is chosen to expand the current node. The choice of an attribute is normally based its size. If the number of bytes in an attribute is less than 8 bytes, it will be partitioned, otherwise, it checks the size of an attribute of between 9 and 16 bytes and the data distribution of the attribute is less than or equal to 20%. If the found attributes are less than or equal to 20%, they qualify to be MPSV. This is followed by checking those frequently accessed attributes which are higher than the mean of total frequently accessed attributes. It qualifies to be MPSV if this is verified.

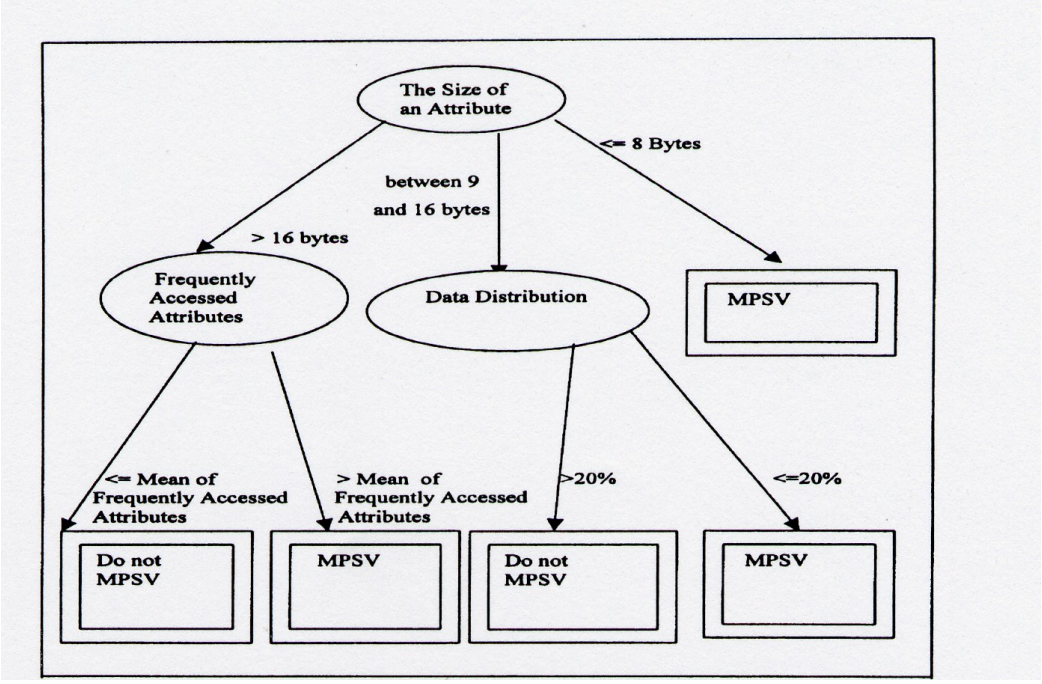


Fig. 6: Hunt's Method for Finding Qualified MPSV

The results in Table 4 show that 14 attributes (a_1, a_2, a_3, a_4, a_6, a_7, a_8, a_9, ForeignStaff, Gender, NoOfSkill, Salary, YExperience and YJoin) out of 22 attributes, qualify for creation as MPSV, *MPSVA.txt* after time t_{35}. As shown in the high priority user's access log file, after time t_{35}, a high priority user might issue the same query at time $t_2$as follows:

t_{36} select a_1, a_6, a_8 , b_2, b_3,c_5 from A, B, C
where A.a_2 = B.b_1 and B.b_2 = C.c_1
and Salary > 2500;

The database management system only needs to process data in the MPSV. As the attributes a_1, a_2, a_6, a_8 and Salary can be found in *MPSV_A*, the database management system will rewrite the query above as follows:

t_{36} select a_1, a_6, a_8 , b_2, b_3,c_5 from *MPSV_A*,
B, C where A.a_2 = B.b_1 and B.b_2 = C.c_1
and Salary > 2500;

A mechanism is required to keep track of how efficient MPSVs are created for high priority users. If an attribute is not used over a certain period, it is suggested to drop it from the qualified list. When an attribute is dropped, it is possible to promote another attribute to replace it.

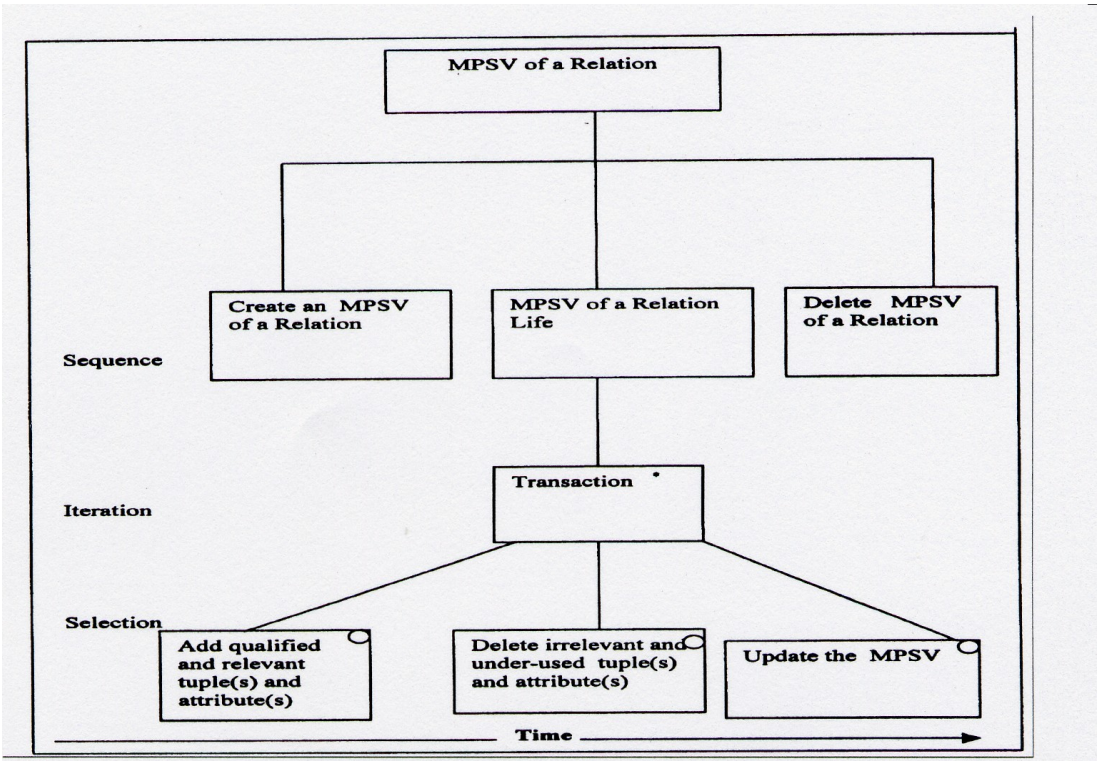


Fig. 7: MPSV Life History

4 Implementation of Data Mining Prototypes

This section discusses the steps to create an MPSV.

Assume that the MPSV has already been generated as *MPSVA.txt* and passed onto the data warehouse administrator. The Entity Life History is used to model the life history of MPSV, as shown in Fig. 7.

4.1 Creation of a MPSV

The data warehouse administrator can use the following 3 steps to generate an MPSV, based on the results of Table 4:

Step 1: Create an MPSV of a Relation

```
Create Table MPSV_A (
        a1 varchar(30),
a2 varchar(45),
a3 varchar(18),
a4 varchar(24),
a6 varchar(4),
a7 varchar(3),
a8 varchar(5),
a9 varchar(8),
ForeignStaff varchar(3),
Gender varchar(6),
NoOfSkill integer(3),
Salary decimal(7,2),
YExperience integer(3),
YJoin integer(3) );
```

Step 2: Insert a value into the MPSV, *MPSV_A*, from Relation *A* as follows:

```
insert into MPSV_A (a1, a2, a3, a4, a6, a7, a8, a9,
  ForeignStaff, Gender, NoOfSkill, Salary,
  YExperience, YJoin) select a1, a2, a3, a4, a6, a7, a8,
  a9 , ForeignStaff, Gender, NoOfSkill, Salary,
   YExperience, YJoin from A
    where ForeignStaff='No' or Gender = "Female" or
     NoOfSkill > 3 or Salary > 2500 or YExperience
     >    2 or YJoin > 3;
```

When the transfer of data from Relation *A* to MPSV *MPVS_A* is complete, we need to know where last tuple has been transferred. The following statements can be used to find out the last tuple of the relation and insert it into Relation *Counter_MPSV*.

Step 3: Create a Counter_MPSV Table

```
Create Table Counter_MPSV (
   RelationName varchar(30),
   TotalNumberOfRecord int(20)) ;
Insert into Counter_MPSV(RelationName) with 'A';

Select count(*) from A;

Total Number of Records is 10000;

Update Counter_MPSV Set TotalNumberOfRecord = 10000
Where RelationName = 'A';
```

4.2 Delete Under-Used Attributes in the MPSV

If the usage of attribute is lower than the threshold value, it is recommended that the data warehouse administrator deletes irrelevant attributes.

Assume that a_3 of MPSV_A is lower than the threshold value, the following steps is done:

Delete an irrelevant attribute of MPSV_A.

ALTER TABLE MPSV_A DROP a_3;

If the usage of data distribution is lower than the threshold value, it is recommended that the data warehouse administrator re-selects the tuples or deletes irrelevant tuples.

Subsequently, the frequently accessed attributes of ForeignStaff = 'No' is changed from ForeignStaff = 'Yes'. The following steps are taken to delete irrelevant tuples and add in relevant tuples:

Delete an irrelevant tuple of MPSV_A.

Step 1) Delete the content of MPSV_A.
Delete from MPSV_A;

Step 2) Insert relevant tuples into MPSV_A
insert into MPSV_A (a_1, a_2, a_3, a_4, a_6, a_7, a_8, a_9, ForeignStaff, Gender, NoOfSkill, Salary, Experience, YJoin) select a_1, a_2, a_3, a_4, a_6, a_7, a_8, a_9 , ForeignStaff, Gender, NoOfSkill, Salary, YExperience, YJoin from A where ForeignStaff= 'Yes' or Gender = "Female" or NoOfSkill > 3 or Salary > 2500 or YExperience > 2 or YJoin > 3;

4.3 Add an Attribute into the MPSV

If the usage of attributes in the unqualified attributes in the Relation *A* is higher than the threshold value, it is recommended that the data warehouse administrator adds relevant attribute(s) into the MPSV.

Assume that a_5 of A is higher than the threshold value, the following steps are taken:

Step 1) Add relevant attribute(s) from M1.
ALTER TABLE *MPSV_A* ADD a_5 varchar(21);

Step 2) Delete the content of MPSV_A.
Delete from MPSV_A;

Step 3) Insert relevant tuples into MPSV_A
insert into MPSV_A (a_1, a_2, a_3, a_4, a_5,a_6, a_7, a_8, a_9, ForeignStaff, Gender, NoOfSkill, Salary, YExperience, YJoin) select a_1, a_2, a_3, a_4, a_5,a_6, a_7, a_8, a_9 , ForeignStaff, Gender, NoOfSkill, Salary, Yexperience, Yjoin from A where ForeignStaff='Yes' or Gender = "Female" or NoOfSkill > 3 or Salary > 2500 or YExperience > 2 or YJoin > 3;

4.4 Update the MPSV of a Relation

Once we have created the MPSV, we need to update it periodically, for example, on a weekly or monthly basis. The following steps will help us to update the MPSV of MPSV_A:

Step 1: Insert a value into the MPSV *MPSV_A* from Relation *A* as follows:

insert into MPSV_A (a_1, a_2, a_3, a_4, a_6, a_7, a_8, a_9, ForeignStaff, Gender, NoOfSkill, Salary, YExperience, YJoin) select a_1, a_2, a_3, a_4, a_6, a_7, a_8, a_9, ForeignStaff, Gender, NoOfSkill, Salary, YExperience, YJoin from A, Counter_MPSV where ForeignStaff='No' or Gender = "Female" or NoOfSkill > 3 or Salary > 2500 or YExperience > 2 or YJoin > 3 and (A.recordno > Counter_MPSV.TotalNumberofRecord and Counter_MPSV.RelationName = 'A');

Step 2: Update Relation Counter_MPSV

Select count(*) from A;

Total Number of Records is 20000;

Update Counter_MPSV Set TotalNumberOfRecord = 20000 Where RelationName = 'A';

4.5 Delete the MPSV of a Relation

If the MPSV is not used at all, it is recommended that the data warehouse administrator deletes the MPSV as follows:

Step 1: Drop the table.

Drop Table MPSV_A;

Step 2: Delete the tuple of Counter_MPSV.

Delete From Counter_MPSV where
RelationName = 'A';

5 Conclusions

The table below shows performance TPC-H [3] sample queries.

The results of this work are expected to encourage wider use of data mining techniques in the MPSV. The current tools do not allow data warehouse administrators to optimise the performance of their data warehouse. The high priority user's access log file has a significant impact on the model that had been built. The selected attributes indexed, only represent the behaviour of a high priority user accessing a particular relation for a certain time. To overcome the limitation of high priority user's access log file in creating materialised projection view, it is recommended that data warehouse administrators or decision makers delete a materialised projection view, if it is not used at all. The present research is the

Query Name	Before the Decision Tree Model	After the Decision Tree Model
Query 1	1:17 min	47 sec
Query 2	17:09 min	1:08 min
Query 3	2:47 min	1:39 min
Query 4	50 sec	9 sec
Query 5	20 sec	5 sec
Query 6	3:25 min	6 sec
Query 7	18 sec	3 sec
Query 8	14 sec	6 sec
Query 9	37 sec	30 sec
Query 10	3:08 min	3 sec
Query 11	7 sec	4 sec
Query 12	11 sec	11 sec
Query 13	46 sec	2 sec
Query 14	4:51 min	2 sec
Query 16	6 sec	3 sec
Query 17	3:54 min	2:26 min
Query 18	2:06 min	1:19 min
Query 19	1:39 min	4 sec

first attempt to handle redundant data structures dynamically through the entity life history of a relation.

References

[1] S. Agrawal, S. Chaudhuri, and V. Narasayya: Automated Selection of Materialized Views and Indexes for SQL Databases. In: Proceedings of the 26th International Conference on Very Large Databases, San Francisco, Morgan Kaufmann (2000)

[2] L. Bellatreche, K. Karlapalem, and M. Mohania: Some Issues in Design of Data Warehouse Systems , in, Dr. Shirley A. Becker (Eds.), Developing Quality Complex Database Systems: Practices, Techniques, and Technologies, Western Hemisphere, Ideas Group Publishing, (2001) 125-172

[3] Transaction Processing Performance Council, *TPC-H*. Available from World Wide Web: http://www.tpc.org/tpch/default.asp Last modified: February 24, 2003.

[4] Ferguson et al.: An application of data mining for product design. IEE Colloquium on Knowledge Discovery and Data Mining. (1998) 5/1 -5/5

[5] Ogilvie et al.: Use of data mining techniques in the performance monitoring and optimisation of a thermal power plant. IEE Colloquium on Knowledge

Discovery and Data Mining. (1998) 7/1 -7/4

[6] Steele et al. Knowledge discovery in medical databases: what factors influence a successful bone marrow transplant for Hodgkin's disease. IEE Colloquium on Knowledge Discovery and Data Mining. (1998) 3/1 -3/8

[7] V. Harinarayan, A. ,Rajaraman and J. Ullman, Implementing Data Cubes Efficiently. New York: ACM Press. (1996)

[8] K. A. Ross, and Z. Li: Fast Joins Using Join Indices. The VLDB Journal. 8(1). (1999) 1- 24

[9] Claussen et al.: Exploiting early sorting and early partitioning for decision support query processing. The VLDB Journal, 9(3), (2000) 190-213.

[10] Chaudhuri, S., and Narasayya, V. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. *In*: *Proceedings of the 23rd International Conference on Very Large Databases.* San Francisco: Morgan Kaufmann. (1997) 146-155.

[11] Vassiliadis, P, and Sellis, T. K. *A Survey of Logical Models for OLAP Databases. In* : http://portal.acm.org/toc.cfm?id=342009&type=proceeding&coll=portal&dl=ACM&CFID=7259477&CFTOKEN=41741337 *Proceedings of SIGMOD.* New York: ACM Press http://dblp.uni-trier.de/db/journals/sigmod/sigmod28.html - VassiliadisS99. (1999) 64-69.

[13] Joshi et al. *Parallel Algorithms for Data Mining.* San Francisco: Morgan Kaufmann. (2000).