Equi-Join Table Optimization Technique for Temporal Data

Mohd Shafry Mohd Rahim, Norazrin binti Kurmin, Mohd Taib Wahid, and Daut Daman
Faculty of Computer Science and Information Systems, University Technology Malaysia, Malaysia

Abstract: *Temporal data management is significant to applications such as environmental management systems. It is to ensure that the process of data storing, retrieving and manipulation can be conducted in an efficient manner. The main focus of this research is on the retrieval of temporal data. Evidently, in the area of temporal data retrieval, the issue that is given most attention by researchers is how to speed up data retrieval time. In our research, we attempt to tackle this issue in an information system which stores hydrological data using a database that utilizes the cube method. As an end result, we managed to establish a technique called the equi-join table optimization technique that was implemented to an existing database system. This technique will also analyze a query statement with several possible query executions to determine the most possible optimum implementation. The outcome indicated that there is indeed an improvement concerning the data retrieval response time.*

Keywords: *Database, information retrieval, optimization, temporal data management, hydrological data.*

Received December 30, 2005; accepted August 162006

1. Introduction

Presently, an effective method in information retrieval is becoming more and more important due to expansion of resources and information technology usage. In regards to this rapid expansion, the need to obtain immediate feedback in any data transaction has become the utmost priority to users. It has encouraged researchers to improve existing methods to significantly reduce the amount of time required to process data retrieval in databases hence the main focus of our research. This paper is organized as follows. Next two sections describe the problem background and research background. Section 4 reviews the optimization techniques. Section 5 establishes the methodology of this paper. Section 6 includes the testing and implementation aspects. Section 7 discusses the results. Finally, section 8 discusses the future works.

2. Problem Background

For temporal data applications, there is a higher need for an effective management of data. As these data are affected by time element, the stored data are not just only current but also include past data. Every change needs to be recorded and stored in the database. Hydrological data is one example of such data, which is also our main concern in this research. In an example situation of managing hydrological data, a total of rain dispersion data will be recorded for every hour of every day and so forth. As a result, the data stored in the database can reach up to many years worth of data. It is simply because the data are recorded continuously and in turn will increase the number of records in the database, thus making it more complex to manage. The most common problem in handling temporal data is the difficulty in handling complex data [3, 7, 9, 12, 13]. In addition to this, the processing speed in temporal data retrieval is also known to be a main issue among researchers [4, 8, 10, 18, 19]

3. Research Background

Database refers to an assembly or a collection of related data which are logically linked in order to fulfill the needs of an organization. Database Management System (DBMS) is a system that enables user to define, build and maintain a database. Additionally, DBMS also provides control towards the database [5]. Managing and analyzing data in a database which is constantly changing has become a critical issue for larger applications where a faster speed of data retrieval process is much needed. One of the techniques that is able to resolve this is the Table Optimization Technique.

However, by using this technique, certain problems still arise concerning data handling in the database. Paper [4] suggested an optimization technique based on the histogram. The equi-join association method is utilized to link two relations with the same attribute and subsequently implements the histogram-based algorithms which are the Averaged Rectangular Attribute Cardinality (R-ACM) and Trapezoidal Attribute Cardinality (T-ACM) in order to improve the run time of the querying process.

A group of researchers from the Institute of Software Technology, Faculty of Computer Science and Information System [16, 17, 18] then introduced the Cube System approach to create database model. Based on the Cube System, each hydrological data combine three main grounds which are non-space data, space data and time. This concept stated that the hydrological data that is being accessed must possess 3 significant properties which are Where?, What?, and When?. A retrieval process must refer to these three properties. It helps in assisting a retrieval process by only stressing on those three important elements compared to many relations that exist in a database. However, the Cube System still does not resolve problems concerning retrieval processes that involve large number of records. Storing a time-based data such as hydrological data has become a bigger issue since the records are being kept on for many years [16].

In this research, an existing system called the Malaysian Hydrological Information System (MHIS) has been used as a case study. MHIS is an information system for storing hydrological data concerning rain, water level, evaporation and water quality. It was developed by the Institute of Software Technology, University Technology Malaysia [17, 18]. The data stored will be used for water resource evaluation study and hydrological information. Apart from that, it will also be used for the purpose of development and management. In order to assist the party involved in these tasks, the data stored will be analyzed and displayed in the form of a report based on the requirements intended.

4. Review of Optimization Techniques

The key objective in problems related to temporal data management is to reduce the amount of time required to process data retrieval in databases. According to [21], the run time duration for a query statement can be reduced using several join methods. These methods suggest joining relationships in the database in order to speed up query processes. Some of these methods proposed include the semi-join [2], equi-join [10, 20], project join [11] and the two way semi-join method [6]. Lubna [10] used a semi-join method as a relationship reducer in the database which prevents the need to create a large size relationship. Outer-join method is commonly used whenever there is a neglected value in a joint attribute in a relationship that has been linked. However, not all DBMS support this type of relationship directly [15]. The equi-join method is also known to be capable of reducing the size of relationship and at the same time eases the retrieval process. In spite of these proposed methods, there are still weaknesses in handling query statements that involve accessing data with numerous records.

Several techniques and algorithms have been introduced for query optimization and among these was the AHY algorithm, which uses the semi-join method [2]. This algorithm was introduced to achieve the following objectives:

1. To minimize response time.
2. To minimize the time required for transferring data.

However, the disadvantage of the AHY algorithm is that it is static, meaning that whenever a query statement is executed, it could not be altered. This is because it is formed according to a designated pattern.

Timos [20] proposed the interleaved IE and HA. IE generates all possible implementation methods for each statement, which were created by reducing the original relationship size using the equi-join method. HA algorithm, on the other hand is employed to decrease the number of selection of implementation method in order to obtain the best possible implementation method. Lubna [10] then proposed an optimization technique known as JAL Dynamic Algorithm. This technique was introduced to overcome the disadvantage of the AHY technique, and enable the querying process to be dynamic. Apart from that, it eliminates redundancy in order to reduce query response time. Haraty and Fany [6] implemented the Partially Encoded Record Filters (PERF) using the AHY algorithm, which consequently produces the AHYPERF algorithm. PERF join is a two way semi-join, which uses byte vector as the backward phase.

Chen [4] proposed an optimization technique using a histogram-based approach. He uses the equi-join method to link two relations that possess the same attributes. The histogram-based algorithm is then implemented, which is Averaged R-ACM and T-ACM to enhance the query implementation process. Mc Mahan [11] soon after introduced an approach based on project-join method. This approach uses structure properties for each query to minimize the size of the temporary results of a query during the query evaluation process. It also shows how structured techniques such as projection join, pushing join and reassemble join could improve the query run time. Table 1 summarizes the comparison between all of the optimization techniques mentioned.

In this research, we propose the usage of the equi-join method to handle the problems discussed previously. The equi-join operation can be implemented using the nested loop join algorithm, sort merge loop join algorithm and hash join algorithm [14, 15]. For each technique developed, the nested loop join algorithm will implement the join operation selected. This is due to the fact that this method is the easiest and most simplified method to apply.

5. Methodology

Figure 1 illustrates the implementation flow of the produced system architecture. The following is the implementation sequence:

1. For each report generated, a query statement is created to retrieve the data intended from the database.
2. The query statement is created using either one of the following methods:
   a. Based on the original database relations' structure without employing any optimization technique.
   b. Based on the table optimization technique created.
3. Next, the query analyzing process will generate several query statement implementation methods and determine which of these methods is the most optimal in terms of cost in generating a report.

Table 1. Comparison of Optimization Techniques.

Technique/ Author	Join Method	Join Implementation
AHY Algorithm [2]	Semi-join	Three implementation phases: 1. Local processing to filter unnecessary data. 2. Semi-join diminishes data transportation from one section to another. 3. Assembles data to destination.
Interleaved Execution Algorithm [20]	Equi-join	There are two types of implementation: 1. IE Algorithm: Generates all possible implementation method using equi-join technique. 2. HA Algorithm: Reduces number of selection for implementation method.
AHYPERF Join [6]	Two-ways semi join	Stages in PERF join: 1. Focus on the attributes that will be joined in R1 and set to PR1. 2. Transfer PR1 into R2 (Forward phase). 3. R2 relation can be reduced by implementing semi-join on PR1. 4. Byte vector (PERF), which contains a byte for each tuple will be send back to R1 in the same sequence. If the data is suitable, tuple is then set to 1. Otherwise it will be set to 0.
JAL Dynamic Algorithm [10]	Semi-join	Eliminates redundancy to reduce query response time.
Histogram-based Algorithms [4]	Equi-join	1. Uses equi-join to join relations that has the same attributes. 2. The implementation uses histogram-based algorithm, which is Averaged R-ACM and T-ACM to improve query response time.
Structured Techniques [11]	Project-join	Uses structured techniques: 1. Projection. 2. Pushing. 3. Reassembling joins.

5.1. Table Optimization Technique

Our produced technique, known as the Equi-join Table Optimization Technique is an enhancement to the join technique established by [15]. The enhancement done on the equi-join method refers to the entity classification step. For instance in Figure 2-a, a single entity is classified into several entities based on group of suitable attribute which are the same characteristic and same behavior. The purpose of this step is to identify and collect the repeating attributes into a new attribute. In addition, it also decreases the number of tuples to be referred to during the data retrieval process.

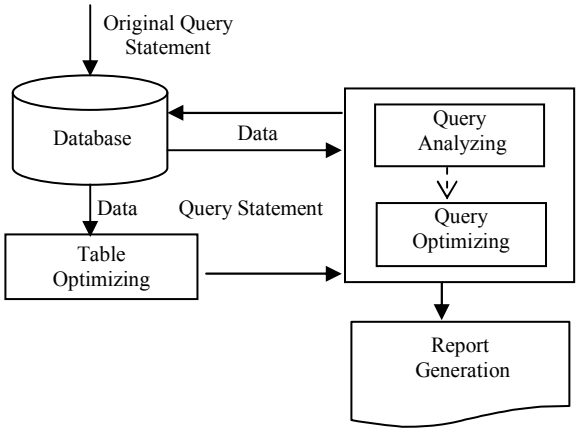

Figure 1. System Architecture.

Under the equi-join concept, two entities are joined to form a new entity. An example shown in Figure 2-b, entities *R* and *S* are joined to form a new entity, *T*. Tuple t in the entity *T* is formed from tuples *r* and *s*, in which *r* is a subset to entity *R* and *s* is a subset to entity *S*. In tuple *t*, the attribute value is a result of joining the attribute value of tuple *r* which is equivalent to the value in tuple *s*. By joining two entities to form one, the number of entities and relations in the database can be decreased. Hence, the data retrieval process could be performed in a much more efficient manner. Figure 2 illustrates the design of the technique developed.

5.2. Rules of Equi-Join Table Optimization

A few rules are applied in order to produce this optimization technique:

1. Eliminate attributes' redundancy: Based on this rule, attributes that are equivalent in two entities will be joined by eliminating one of the attributes from either one entity. Other attributes will be combined to form a new entity. The attribute value for each tuple will be verified. If the attributes' values for both entities are equivalent, the value will be joined and stored in a new entity. With reference to the operation perform on the set [1], this rule can be represented as follows:

$$\{t = r \cup s \mid r \in R, s \in S, t \in T\} \quad (1)$$

2. Joining two entities to form single entity: In this rule, certain entities will be joined to form a new entity. Each attribute that is used to define the condition for joining must be compared using relational operators. For the technique developed, the equality operator (=) will be used. Based on the

fundamental definition of join [15], the rules can be defined as follows:

$$T = R \bowtie_{r(a) = s(b)} S \qquad (2)$$

3. Dividing a single entity into several entities based on group of suitable attribute which are of the same characteristics and same behavior.

Similar to rule 1 and 2, the goal of rule 3 is to ease the data retrieval process. Entities will be classified into other attributes based on group of suitable attribute which are the same characteristics and same behavior. There are two steps involved in implementing this process which are clustering and classification.

Clustering is performed by collecting the same attributes into a temporary entity. In clustering, the attribute value that will be used as the condition for classification will be verified. If the attribute value in the first tuple is equivalent to value in the following tuple, these values will be joined and stored in a temporary entity. This process will be repeated until it reaches the last tuple of that entity.

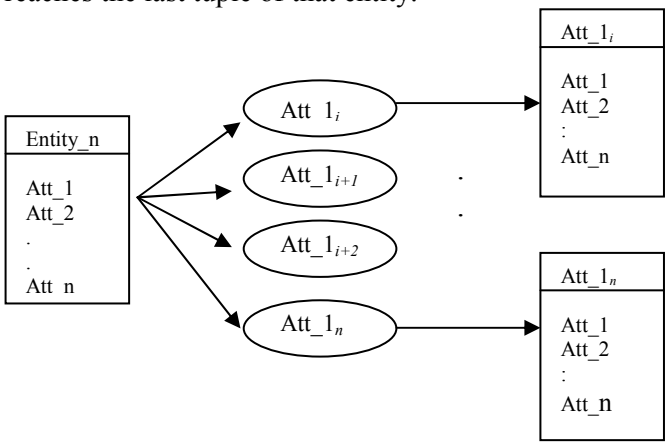

Figure 2-a. Enhancement on the Equi-join.

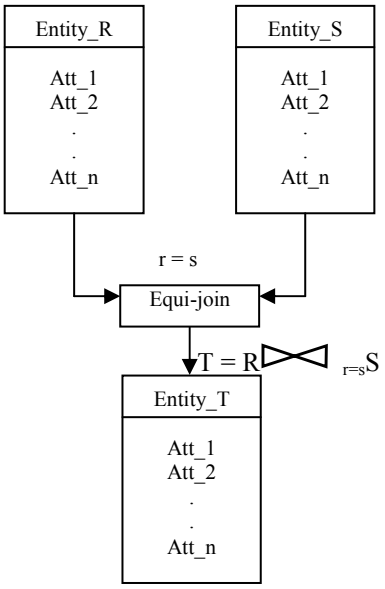

Figure 2-b. Equi-join method.

Generally, the rules can be described using the following equations:

$$r' = \sum_{i=1}^{m} \left[\sum_{j=1}^{n} \left((r_j = r_i, r_{j+1} = r_{i, \ldots\ldots}, r_n = r_i), (r_j = r_{i+1}, r_{j+1} = r_{i+1, \ldots\ldots}, r_n = r_{i+1}), \ldots\ldots, (r_j = r_m, r_{j+1} = r_{m, \ldots\ldots}, r_n = r_m) \mid r \in R \right) \right] \qquad (3)$$

where:
r' = Entity to be classified.
r = Entity to be classified.
n = Number of tuples for classified entities.
m = Number of tuples for classified entities.

*//******** Joining Entity*********
1. For each entity, R_1 and R_2 to be joined, the following will be done
   1.1 For each tuple R_1, do
      a. For each tuple R_2, check
         If attribute $R_1(a) = R_2(b)$
         then join R_1 and R_2
      b. Save the newly formed relation, R'

*//********Classifying Entity********
2. Initialize number_of_entities = 0
3. For each entity to be classified, R_3, do
   3.1 Set the value for the first tuple , t_1, to a variable,
   Tuple_entity$_{number_of_entities} = t_1$
   3.2 For each tuple, t_n, in R_3, check
      a. If R (t_n) attribute value = tuple_entity$_{number_of_entities}$ then eliminate t_n
      b. Save t_n in newly formed relation, R_{t1}
4. number_of_entities = number_of_entities + 1
5. Repeat step 3 and 4
   while number of tuple <> 0

Figure 3. Table optimization algorithm.

6. Testing and Implementation

The main goal of the testing phase is to see the effectiveness of the technique developed compared to the existing table optimization technique used in our case study system, MHIS. Several sets of query statements based on Structured Query Language are created to test the effectiveness of this technique. These statements will be used to retrieve data from the database. A Relational Database Management System (RDBMS), MS Access 2003 were used in this testing phase. Active Server Pages (ASP) was used for generating reports of the query implementation. We also used Oracle 9i to determine and select the most optimum query statement. Table 2 shows the entities and number of tuples for each entity to be used in the testing stage.

The execution of our technique involved three entities which are dt_char, dt_real and dt_name. These entities store data concerning station name, value of rainfall, evaporation, water suspending sediment and

water quality. The numbers of tuples used initially amount to 871788 but after running the Table Optimization Algorithm, it is reduced to 862038 tuples. The algorithm managed to eliminate about 9750 tuples that stores the same information thus reduces redundancy. This is achieved when the algorithm classifies the data into differences of entities based on type of event. Thus the data in the three original entities will be grouped into new entities which are dt_station, dt_rainfall, dt_evaporation, dt_water_level and dt_sus_sediment. Originally, the database structure in our study case system MHIS, stores all information by type of entity, not by type of event. Therefore to increase the speed of query processing time, entities must be classified using the technique proposed.

Table 2. MHIS relation before and after optimization (Total data capacity is 100 MB).

Entity	Without Optimization	With Optimization
	Number of Tuples	Number of Tuples
dt_char	36900	-
dt_name	3500	-
dt_real	83388	-
dt_station	-	3650
dt_rainfall	-	183145
dt_evaporation	-	42229
dt_water_level	-	258307
dt_sus_sediment	-	347707

6.1. Comparison Criteria

The criteria considered in comparing our developed technique to the existing technique are the query run time. Each query will be executed at least 10 times. The run time is the average obtained from every execution.

Run Time = (Execution 1 + Execution 2 + ... + Execution 10) / 10

In order to obtain the run time for each execution, the query statement is executed using Oracle SQL Analyzer.

6.2. Comparison Before and After Optimization

In this section, although we have run the test for all data classified, we will only discuss query that was executed by using rainfall data (temporal data). This section will explain how various possible query executions are selected using the ORACLE query optimization. This implementation only uses the ORACLE query optimization and does not alter any basic rules in selecting the most optimal execution. For each query statement, there are several possible executions. Appendix 1 illustrates eight possible executions for each query statement listed in Table 3 and Table 5.

6.2.1.Total of Rain Dispersion

Table 3 defines the run time average for query statements of total rain dispersion P1, P2, P3, and P4 before optimization process. The initial statement for each statement is included in Appendix 1.

Table 3. P1, P2, P3, and P4 run time.

Second	P1	P2	P3	P4
1	0.85	1.00	1.60	1.76
2	0.79	0.99	1.42	2.07
3	0.77	0.97	1.44	1.77
4	0.78	0.96	1.51	1.68
5	0.79	0.93	1.38	1.71
6	0.75	0.96	1.32	1.71
7	0.80	0.94	1.36	1.72
8	0.75	0.93	1.38	1.77
9	0.77	0.94	1.35	1.72
10	0.77	0.93	1.39	1.77
Average	0.782	0.955	1.415	1.768

Table 4 defines the difference between the run time before and after optimization. The run time obtained before optimization is from the time average in Table 3 whereas the run time obtained after optimization is based on the best possible executions selected as discussed earlier. Each possible execution selected for statement P1, P2, P3 and P4 will be compared with the run time before optimization.

Table 4. Comparison between run time for P1, P2, P3, and P4

	P1	P2	P3	P4
Before Optimization	0.782	0.955	1.415	1.768
After Optimization	0.052	0.117	0.156	0.225
Percentage Increased (%)	93.4	87.7	89.0	87.3

For P1, the run time after optimization is much faster than before optimization by 0.052 seconds. P2, P3, P4 also have faster run time compared to before (refer to Table 3). The comparison results obtained are shown in Figure 5. According to the graph, the speed percentage is increased by approximately 89%. Judging from the result, it is clear that the technique developed is much more effective in handling the problems discussed earlier.

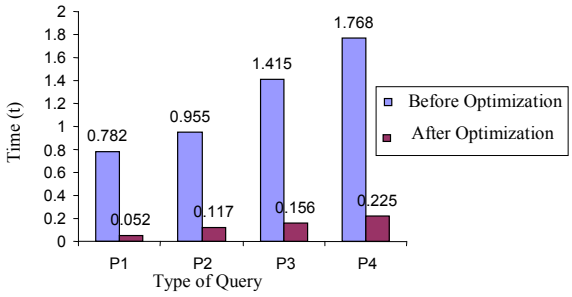

Figure 5. Comparison Graph P1, P2, P3, and P4.

6.2.2. Average of Rain Dispersion

Next comparison is for P5, P6, P7 and P8 which are query statements for accessing information regarding the rain dispersion average. Based on the test

conducted, the execution that produces the most optimal time will be selected. The chosen execution will be compared with the original query statement process. This is done to test the effectiveness of the optimization technique developed. Table 5 defines the run time average for each query statement P5, P6, P7, P8. Table 6 shows the difference between each query statement's run time execution before and after optimization.

Table 5. Run time for statement P5, P6, P7, and P8.

Second	P5	P6	P7	P8
1	0.73	0.94	1.43	1.89
2	0.75	0.94	1.82	1.97
3	0.75	0.94	1.36	1.68
4	0.77	0.95	1.32	1.71
5	0.77	0.93	1.50	1.79
6	0.80	0.97	1.40	1.77
7	0.76	0.99	1.34	1.74
8	0.72	0.99	1.36	1.77
9	0.75	0.97	1.37	1.86
10	0.74	0.99	1.35	1.74
Average	0.754	0.961	1.425	1.792

Table 6. Run time difference between P5, P6, P7, and P8

	P5	P6	P7	P8
Before Optimization	0.754	0.955	1.415	1.768
After Optimization	0.091	0.117	0.156	0.225
Percentage Increased (%)	87.9	87.7	89.0	87.3

This result also indicated that the run time for the execution of query statement after optimization is much faster compared to before optimization. Figure 6 illustrates the percentage increased for each query statement executed. The average for the increased percentage of run time is approximately 87.3% as shown in Table 6.

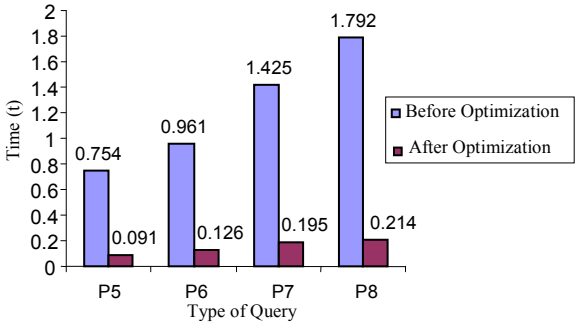

Figure 6. Comparison graph P5, P6, P7, and P8.

7. Discussions

From the results, we could conclude that the technique developed is able to overcome the problems mentioned earlier which are slow speed processing in data retrieval from the database. There are several advantages in using the optimization technique developed. It can prevent data redundancy by table optimization algorithm. Entity is classified into several entities based on the group of features identification.

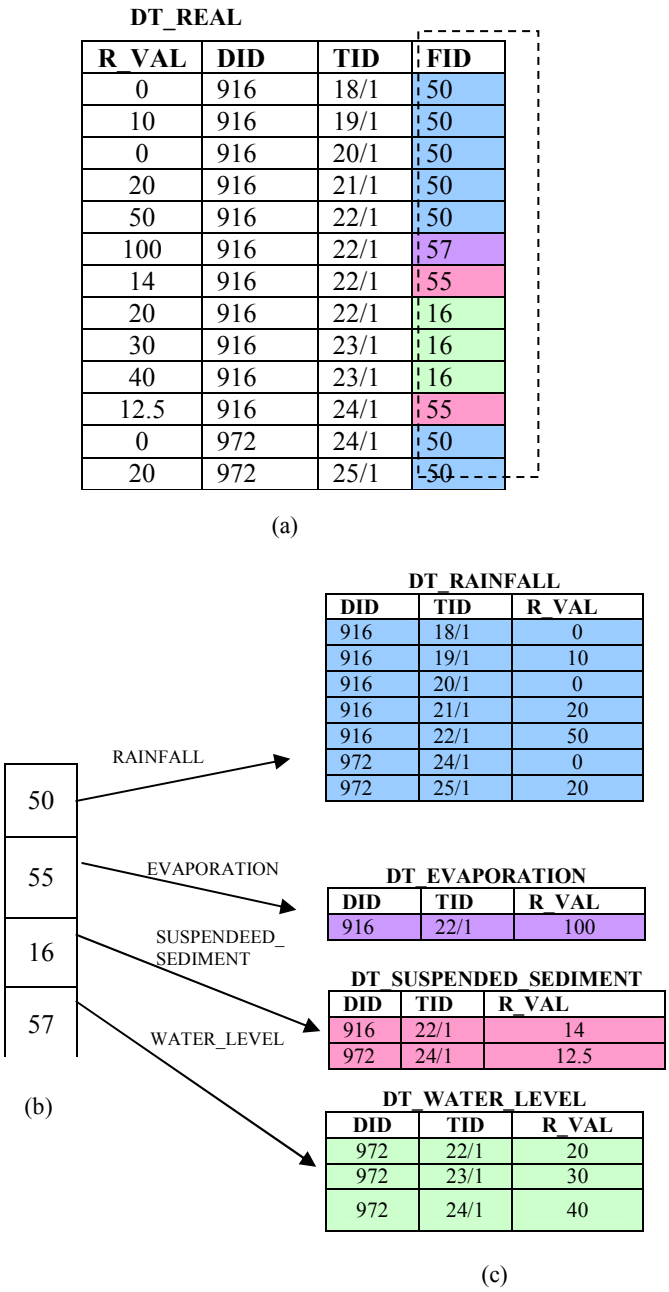

Figure 7. Illustration of table optimization process.

Figures 7-a, 7-b, and 7-c, illustrate how Table Optimization Algorithm works. Figure 7-a shows an example of hydrological data stored in the DT_REAL entity. Referring to the table, there are a few repeating tuples meaning that each of these tuples has the same value. The value of each attribute will be confirmed according to the classification values obtained from Figure 7-b. Next, if the attribute value confirmed is equivalent, it will be stored in new entities which are DT_RAINFALL, DT_EVAPORATION, DT_SUSPENDED_SEDIMENT, DT_WATER_LEVEL as shown in Figure 7-c.

Furthermore, the implementation of this technique is able to assist in data retrieval from the database. Using the same Figure 7 as example, the entity DT_REAL can be classified into other entities based on one attribute which is FID. For each new entity, the

attribute value for each tuple is equivalent to the FID attribute value. This would assist in the retrieval data process where it only needs to refer to a certain entity compared to before, where it has to refer to an entity with many repeated data. Consider the information requested is the data with the FID value of 50. The entity referred to before classification is the DT_REAL as shown in Figure 7-a where there is number of tuples that need to be processed is 13 lines. However, after classifying the entity, it only needs to refer to a single entity which is DT_RAINFALL. The number of tuples that need to be referred to is 7 lines lesser than the number of tuples referred without entity classification. This will significantly reduce the time needed to process a query which was proven in the testing implementation explained in section 6.

However, there are still some weaknesses in this new technique proposed. For each query executed, possible executions could be rather alike to one another. As described earlier, each query statement has four possible similar executions. This is due to the fact that each query statement produced is based on the table optimization technique uses the same set rules.

8. Future Works

To increase the effectiveness of data retrieval, several suggestions for improvement have been studied and implemented. Among these suggestions is producing an algorithm that is able to analyze queries in determining and selecting the query with the most optimum implementation period.

This paper focuses on the implementation period for data retrieval from the database where the usage of space factor is not taken in account. Therefore, it is proposed that this factor should be taken into account for future research to enhance the effectiveness of the technique produced.

References

[1] Abdullah M. Z., *Discrete Mathematics, Introduction of Mathematic*, First Edition, Open University Malaysia, Malaysia 2004

[2] Alan R. H., Wu O. Q., and Yao S. B., "Query Optimization on Local Area Networks," *ACM Transaction on Office Information*, vol. 3, no. 1, pp. 35-62, 1985.

[3] Chen D., "Real-time Online Hydrological Information and Modeling System Using Object-Oriented Approach and Relational Database for Flood Defense," *in Proceedings of the Flood Defence'02*, New York, Science Press, 2002.

[4] Chen J. B., "On Utilizing New Histogram-Based Methods for Query Optimization," *Master Thesis*, Carleton University, Canada, 2003.

[5] Connoly T. and Begg C., *Database Systems: A Practical Approach to Design, Implementations, and Management*, Addison Wesley, UK, 1998.

[6] Haraty R. and Fany R., "Distributed Query Optimization Using PERF Join," *in Proceedings of the 2000 ACM Symposium on Applied Computing*, Como, Italy, pp. 284-288, 2000.

[7] Kafer W. and Schoning H., "Realizing a Temporal Complex-Object Data Model," *in Proceedings of the SIGMOD'1992*, San Diego, New York, USA, ACM Press, pp. 266-275, 1992.

[8] Kouramajin V., "Temporal Database: Access Structures, Search Methods, Migration Strategies and Declustering Techniques," *PhD Dissertation*, The University of Texas at Arlington, 1994.

[9] Lorentzos N. A., "DBMS Support for Nonmetric Measurement Systems," *Knowledge and Data Engineering, IEEE Transactions*, vol. 6, no. 6, pp. 945-953, December 1994.

[10] Lubna S., "Dynamic Technique in Distributed Query Optimization," *Master Thesis*, University of Windsor, 2002.

[11] McMahan B. J., "Structral Heuristics for Query Optimization," *Master Thesis*, Rice University. Houstan, Texas, 2004.

[12] Mesru K., Cicekli N. K., and Yazici A., "Spatio-Temporal Querying in Video Databases," *Information Sciences-Informatic and Computer Science: An International Journal*, vol. 160, no. 1-4, pp. 131-152, March 2004.

[13] Min J. S., Kim D. H., and Ryu K. H., "A Spatiotemporal Data and Indexing," *in Proceedings of IEEE Region 10th International Conference*, vol. 1, pp. 110-113, August 2001.

[14] Navin K., "Query Optimization for Object Relational Database System," *PhD Thesis*, University of Wiconsin, Madison, 1999.

[15] Priti M. and Margaret H. E., "Join Processing in Relational Databases," *ACM Computing Surveys*, vol. 24, no. 1, pp. 63-113, 1992.

[16] Rahim M. S. M., Daman D., and Selamat H., "Design and Implementation of Double Cube Data Model for Geographical Information Systems," *The International Arab Journal of Information Technology*, vol. 1, no. 2, pp. 215-220. July 2004.

[17] Rahim M. S. M., Daman D., and Selamat H., "Spatial and Non Spatial Enhancement Database for Hydrological Information System (HIS)," *Technical Report, Research Monograph*, University Technology Malaysia, Malaysia, 2002.

[18] Syed H. B. M., Daman D., and Selamat H., "Establishment of Hydrological Information System for Water Resources Pelanning, Development and Management, Jabatan Pengairan Dan Saliran Malaysia, Kuala Lumpur," *Technical Report*, Department of

Irrigation Malaysia, Ministry of Agriculture Malaysia, vol. 1- 4, 1999.
[19] Tao Y. "Indexing and Query Processing of Spatio-Temporal Data," *PhD Dissertation*, The Hong Kong University of Science and Technology, Hong Kong China, 2002.
[20] Timos K. S., "Multiple Query Optimization," *ACM Transaction on Database Systems*, vol. 13, no. 1, pp. 23-52, 1988.

Mohd Shafry Mohd Rahim received his Diploma in computer science 1997, BSc computer science 1999 form University Technology Malaysia, and his MSc in research GIS and visualization from University Technology Malaysia, in 2002. Currently, he is a lecturer in the Department of Computer Graphic and Multimedia, Faculty of Computer Science and Information Systems at University Technology Malaysia (UTM). He has over seven years of experience in the field of computer graphics & visualization and GIS data management. He has also been actively involved in many research projects related to GIS & visualization and GIS data management, and has published more than 45 publications. Currently , his research is on development of spatial, temporal, and spatiotemporal data management applications.

Norazrin binti Kurmin received first degree from University Technology of Malaysia (UTM) with Bachelor in computer science in software engineering in 2003 and Master in computer science in information systems in 2006. She holds the position of IT officer in Research Management Center, University Technology of Malaysia. She has strong research background in the fields of query optimization for database, temporal data in a database, and web programming.

Mohd Taib Wahid received first degree in computer science in information systems from University Technology of Malaysia in 1996 and Master in computer science in real time software engineering in 1998 from University Technology of Malaysia and Thomson Campus, France. Currently, he is holding the position of lecturer in the Department of Information Systems, University Technology of Malaysia. He is also being actively involved in many research projects related data model and software development project as a project leader with various public and private sectors. His research interest includes database design and data modelling, data retrieval, query optimization, and enterprise resource planning.

Daut Daman received his BSc from Universiti Sains Malaysia and MSc from University of Cranfield, United Kingdom. Currently, He is an associate professor in the Faculty of Computer Science and Information Systems at University Technology Malaysia (UTM). He has over 26 years of experience in the field of computer graphics & visualization. He taught subjects in the area of computer graphics, scientific visualization, and software technologies. He has also been actively involved in many research projects related to computer graphics & visualization and has published more than 100 publications.

Appendix 1: Queries Processing Used for Testing

Query	Query Statement
P1 – Query statement for the total of rain dispersion in one year	Select dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid, SUM(dt_real.rval) as total From dt_name, dt_real Where (((dt_name.fid) = 916) and ((dt_real.fid) = 916) and ((dt_real.did) = 50) and ((dt_real.tid) >=# 1/1/1990 8:50:0# and (dt_real.tid) <=# 12/31/1990 12:0:0#)) Group by dt_real.tid;
P2 - Query statement for the total of rain dispersion in 5 years	Select dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid, SUM(dt_real.rval) AS total From dt_name, dt_real Where (((dt_name.fid) = 916) and ((dt_real.fid) = 916) and ((dt_real.did) = 50) and ((dt_real.tid) >= #1/1/1990 8:50:0# and (dt_real.tid)<=#12/31/1994 12:0:0#)) Group by dt_real.tid;
P3 - Query statement for the total of rain dispersion in 10 years	Select dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid, SUM(dt_real.rval) as total From dt_name, dt_real Where (((dt_name.fid) = 916) and ((dt_real.fid) = 916) and ((dt_real.did) = 50) and ((dt_real.tid) >= #1/1/1990 8:50:0# and (dt_real.tid) <= #12/31/1999 12:0:0#)) Group by dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid;
P4 - Query statement for the total of rain dispersion in 15 years	Select dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid, SUM(dt_real.rval) as total From dt_name, dt_real Where (((dt_name.fid) = 916) and ((dt_real.fid) = 916) and ((dt_real.did) = 50) and ((dt_real.tid) >= #1/1/1987 8:50:0# and (dt_real.tid) <= #12/31/2001 12:0:0#)) Group by dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid;
P5 - Query statement for the average of rain dispersion in one year	Select dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid, AVG (dt_real.rval) as average From dt_name, dt_real Where (((dt_name.fid) = 916) and ((dt_real.fid) = 916) and ((dt_real.did) = 50) and ((dt_real.tid) >=# 1/1/1990 8:50:0# and (dt_real.tid)<=#12/31/1990 12:0:0#)) Group by dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid;
P6 - Query statement for the average of rain dispersion in 5 years	Select dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid, AVG (dt_real.rval) as average From dt_name, dt_real Where (((dt_name.fid) = 916) and ((dt_real.fid) = 916) and ((dt_real.did) = 50) and ((dt_real.tid) >= #1/1/1990 8:50:0# and (dt_real.tid) <= #12/31/1994 12:0:0#)) Group by dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid;
P7 - Query statement for the average of rain dispersion in 10 years	Select dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid, AVG (dt_real.rval) as average From dt_name, dt_real Where (((dt_name.fid) = 916) and ((dt_real.fid) = 916) and ((dt_real.did) = 50) and ((dt_real.tid) >= #1/1/1990 8:50:0# and (dt_real.tid) <= #12/31/1999 12:0:0#)) Group by dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid;
P8 - Query statement for the average of rain dispersion in 15 years	Select dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid, AVG (dt_real.rval) as average From dt_name, dt_real Where (((dt_name.fid) = 916) and ((dt_real.fid) = 916) and ((dt_real.did) = 50) and ((dt_real.tid) >= #1/1/1987 8:50:0# and (dt_real.tid) <= #12/31/2001 12:0:0#)) Group by dt_name.did, dt_name.fid, dt_name.cval, dt_real.fid, dt_real.did, dt_real.tid;