Manarat International University Studies, 2 (1): 166-176, December 2011
ISSN 1815-6754
@ Manarat International University, 2011

An Algorithm for Further Decomposition of BCNF/3NF to Reduce the Query Cost

Md. Ashraful Islam*

ABSTRACT

In this study we proposed an algorithm, which reduce the query cost in terms of disk accesses. In BCNF/3NF relation the attributes are related with each other by functional dependencies and a lot of attributes exists in a table, which are dependent with each other. Huge number of attributes in a specific relation with different purpose of use and using several times cause the query cost high. In our proposed algorithm we prescribed to decompose a table, which joined with different level of dependency in term of their use. In specific period of time or for specific purpose different attributes are used but not all simultaneously. For that we made some subsets of attributes with same primary key and form several relation based on frequency of use or purpose of use for a specific relation. Some attributes in a relation, that play more than one role means they used several times or purpose and when we are going to decompose a relation into several relation then the space increased. Those are also considered as a problem. We have tried to reduce the query cost according to less increase of space by considering trade-off between excess spaces needed for replication of primary key in each decomposed table.

Keywords: Algorithm, BCNF/3NF, Disk, Communication.

1. INTRODUCTION

Query Cost is measured in terms of disk accesses, CPU time to execute the query and cost of communication. If query is executed in a single computer, the cost of communication remains absent. Since the disk access is an order slower than memory and CPU operations, the disk access time plays dominant role in query cost. The disk access is the measure of the number of transfers of blocks from disk, which can be expressed using the following formula [Silberschatz et al., 1999]

$$b_r = \lceil (n_r * s_r) / b \rceil$$

Where,

b_r = number of blocks containing tuples of relation, r

n_r = number of tuples in the relation r, which are stored in a file

s_r = size of the type of relation r in bytes

* *Bangladesh Islami University, 89/12, R.K Mission, Road,Dhaka-1203, Bangladesh, Email: ashraful47@cse.iiuc.ac.bd*

b = block size

The number of blocks b_r to be accessed for retrieving/updating a relation in a disk file determines the query cost. The more the number of blocks in the relation, the more disk access (thus more time) is required for executing a query on the relation.

For a given block size (b) and the number of tuples(n_r) in a relation, the number of blocks(b_r)(and therefore, the query cost) is directly proportional to the size of the tuple(s_r).That means that meaningful decomposition of a relation can reduce the query update cost. Scientist had conducted studies for query optimization based on search method for selection, join operation, project operation, set operation of query tree and query graphs, and user-defined predicates, However, no such algorithm appears to have been devised for reducing query cost based on decomposition of a relation. Decomposition of a relation had been widely used for reducing redundancy, which ultimately reduced the update anomalies such as insertion anomalies.

In this study an algorithm was developed for reduction of query cost by further decomposition of a BCNF/3NF relation, based on frequency of use of the attributes or groups of attributes identified by the purpose of their use. Here we identify the frequency by counting the time of use of specific attributes and groups are considered by purpose of using. The properties of the BCNF/3NF relations were kept intact into the decomposed relations. The trade of between the excess space requirement due to replication of the primary key and query cost reduction was considered while developing the algorithm. This algorithm is applied on the relation of a banking, garments or others companies' database and reduction in query cost was investigated.

2. Background of the problem

The problem was to reduce the query cost by decomposition of a BCNF/3NF relation. Two types of dependencies were identified based on which the BCNF/3NF relation could be decomposed. They are:

a) Frequency dependency and
b) Group dependency

The frequency dependencies of a relation were identified based on frequency of use of attributes. The group dependencies were identified based on purpose of their use.

2.1 Frequency of use of the attributes

In practice, there may exist hundreds of attributes in a BCNF/3NF relation, but frequency of use of individual attributes may vary substantially. Some attribute may be used and updated very frequently and some are very rarely. Every time an attribute is updated, all the attributes in the relation are handled. Thus, the update cost becomes unnecessarily high. The scientists paid due attention on reducing redundancy, thereby eliminating update anomalies. They have used decomposition of relations for reducing redundancy. The technique of decomposition of relations may also be used for obtaining smaller relations based on dependency of attributes on their use-frequency. This may reduce the query cost without loss of any of the property of normalization.

Let us consider the Bank-Account relation shown in Table 1, in which Account_No is the primary key. The relation is in BCNF and therefore in 3NF, 2NF and 1NF also. Let there be 1000 number of customers having 1000 accounts in the bank. Considering a block size of 512 bytes, the total number of blocks (b_r) comes to 4672. Let us consider a case of linear search. In a linear search, each block is scanned, and all records are tested to see whether they satisfy the selection condition. Since all blocks have to be read, the estimated cost $E=b_r$. For a selection on a key attribute, we assume that one-half of the blocks will be searched before the record is found at which point the search can terminate. The estimate in this case is $E=b_r/2=2336$ blocks per update.

From the experience, a blank Database Administrator (DBA) knows that each time a customer makes a transaction, his balance is updated and Account_No, Currency, Status are used for qualifying the transaction.

Table 1: Bank-Account relation

Attribute Name	Size (Byte)	Attribute Name	Size (Byte)
Account_No	8	Status	5
Name	30	Balance	16
F_H_Name	30	Interest_Rate	8
Address 1	30	Interest_Accrued	16
Address 2	30	Interest_Paid	16
Address 3	30	Intt_Cal_Upto_Date	10
Telephone_No	30	Nominee	30
Account_Type	10	Introducer	30
Currency	5	Signature	1024
Opening_Date	10	Photograph	1024
		Total (Bytes)	2392

If it is a cash withdrawal, Account_Type and Signature are also verified. Let on an average, one such transaction per account per day is made. Interest is calculated on monthly basis, thus Interest_Rate, Interest_Accrued, Interest_Paid and Intt_Cal_Upto_Date are used once in a month. Other information is used rarely, say quarterly on an average. Thus there are three distinct frequency dependencies in the Bank_Account relation. These are:

(i) {Account_No, Account_Type, Currency, Status, Balance, Signature} $\longrightarrow$ f1

(ii) {Account_No, Interest_Rate $\longrightarrow$, Interest_Accrued, Interest_Paid, Intt_Cal_Upto_Date}
f2
(iii) {Account_No, Name, F_H_Name, Address1, Address2, Address3, Telephone_No, Opening_Date,
Nominee, Introducer, Photograph $\longrightarrow$} f3 $\longrightarrow$

Where f1=daily, f2=monthly and f3=quarterly or in other words, f1=300times in a year, f2=12 times in a year and f3=4 times in a year, if year is taken as the base unit for the frequency. Based on the frequency dependencies, the relation may be decomposed into three relations as shown in Figure 2. The size of tuple of each of the here decomposed relations are 1068, 58 and 1282 bytes respectively. For the linear search and 1000 tuples in each of the relation, the estimated costs are 1043, 56 and 1252 blocks per update respectively (this is the number of times the disk needs to access the database for each update). Therefore, the total cost per year before and after decomposition comes to:

i) **Before decomposition**: 2336 blocks per update*(3000+12+4) updates per year

= 7, 38,176 blocks per year

ii) **After decomposition**: (1043*300) + (56*12) + (1252*4)

= 3, 18,580 blocks per year

Saving in access time is 56.84%.

2.2 Classification of attributes into groups based on purpose of use

Some relation in BCNF/3NF may contain attributes, which are used for different purposes at different time units. However the frequency of use of these attributes is almost same for a particular time period. For example, let us consider the Cheque-Leaf relation shown in Table 3.3. The primary key of the relation is Cheque_Leaf_No. The relation is in BCNF. The size of the relation is 75 bytes. Considering 1000 tuples and a block size of 512 bytes, the update cost of the relation becomes 73 blocks.

Table 2: Bank-Account relation decomposed into three relations:

Account_Transaction Relation		**Account_Interest Relation**		**Account_Other Relation**	
Attribute Name	**Size In Byte**	**Attribute Name**	**Size In Byte**	**Attribute Name**	**Size in byte**
Account_No	8	Account_No	8	Account_No	8
Account_Type	10	Interest_Rate	8	Name	30
Currency	5	Interest_Accrued	16	F_H_Name	30
Status	5	Interest_Paid	16	Address1	30
Balance	16	Intt_Cal_Upto_Date	10	Address2	30
Signature	1024			Address3	30
				Telephone_No	30
				Opening_Date	10
				Nominee	30
				Introducer	30
				Photograph	1024
Total Size	1068	Total Size	58	Total Size	1282

There are four groups of attributes in the relation used at different times for different purposes. When a branch receives cheque leaves from the Head Office, they make entry of the same into the relation in which case the Cheque_Leaf_No and the Receiving_Date update. When few of the cheque_leaves inserted earlier into the database are issued to a particular customer, Account_No and Issue_Date attributes are updated against the cheque_Leaf_No issued at this time to the customer. When the customer presents the cheque leaf to the bank counter for withdrawal of money, the cashier makes entry of the cheque leaf particulars into the computer, and Payment_Amt and Payment_Date attributes are updated. If a customer loses a cheque leaf, he informs it to the bank authority and the bank makes it stop payment. In this process, the Stop_Payment and Stop_Pay_Date attributes are updated. Thus we observed that the relation has four distinct groups of attribute, which are used for four different purposes and updated separately. Decomposition of this relation into four relations based on the purposes or in other words, based on their use at different time units, may reduce the query cost.

Table 3: Cheque-leaf relation

Attribute Name	**Size(Byte)**	**Attribute Name**	**Size(Byte)**
Cheque_Leaf_No	10	Payment_Amt	16
Receiving_Date	10	Payment_Date	10
Account_No	8	Stop_Payment	1
Issue_Date	10	Stop_Pay_Date	10
		Total(Bytes)	75

The four dependency functions of the relation of Table-3 are as under:
i) {Cheque_Leaf_No, Receiving_Date}------→ g1
ii) {Cheque_Leaf_No, Account_No,Issue_Date} →g2
iii) {Cheque_Leaf_No, Payment_Amt,Payment_Date} → g3
iv) {Cheque_Leaf_No, Stop_Payment, Stop_Pay_Date} → g4

Where g1, g2, g3 and g4 are four groups of attributes representing different purposes or time units in which they are used together.Based on the group dependencies, the relation may be decomposed into four relations such as ,
R1 (Cheque_Leaf_No, Receiving_Date),
R2 (Cheque_Leaf_No, Account_No, Issue_Date), R3 (Cheque_Leaf_No, Payment_Amt, Payment_Date) and R4(Cheque_Leaf_No, Stop_Payment, Stop_Pay_Date). The sizes of tuple of each of the four decomposed relations are 20, 28, 36 and 21 bytes respectively. For the linear search and 1000 tuples in each of the relation, the estimated costs are 39, 54, 70 and 41 blocks per update respectively.

Considering that 1000 number of cheque leaves are received from Head Office of the bank in a month, these cheque leaves are issued to 100 customers, these customers use 950 leaves for cash withdrawal and rest 50 cheque leaves are made stop payment due to losses or damages, the total cost per month before and after decomposition comes to:

i) **Before Decomposition**: 73 blocks per update* (1000+1000+950+50) updates per month
= **2, 19,000 blocks per month**

ii) **After Decomposition**: (39*1000) + (54*1000) + (70*950) + (41*50)
=**1, 61,550 blocks per month**

Saving in access time is 26.23%

Therefore, it may be resolved that the BCNF/3NF relations having more than one dependency functions determined either by "use of frequency" or "group on different purposes", can further be decomposed to obtain smaller BCNF/3NF relations which will reduce the query cost over a given period of time. Keeping this in view, a formal algorithm has to be developed to decompose the original relation into smaller ones.

3. The Proposed Algorithms

In developing the basic algorithm, the relation in BCNF/3NF was first examined for presence of (i) more than one distinct frequency of use of attributes or (ii) more than one distinct groups classified by purpose of use of the attributes. Based on the distinct frequencies or groups, dependency functions were created. For each of the dependency fountains, one relation consisting of the primary key and the dependent attributes was created. Keeping primary key along with along with all the new relations ensure the properties of the original relation into the decomposed relations. The algorithm thus created, called basic algorithm, does not consider (i) the attribute common in more than one dependency functions and (ii) trade-off between the excess space needed for replication of primary key and the savings in query cost. The basic algorithm is presented below:

Algorithm-1: The basic algorithm for further decomposition of the BCNF/3NF relation

```
1. Identify the distinct frequencies or groups, fi $1 \le i \le n$ in the relation R based on either frequency
   of use or group on purpose of use of the attributes, where n is the total number of occurrences
   of use-frequencies or groups. If n=1 terminate the algorithm.
2. Except the primary key $A_k$, create dependency functions $Ai \rightarrow fi$ where Ai is a subset of
   attributes that occurs during fi.
3. For every dependency functions, create a relation consisting of the primary key $A_k$ and the
   attributes Ai of the dependency functions.
```

Let us consider a relation R= (A, B, C, D, E, F, G), where {A, B} is the primary key. Let there are two dependency functions f1 and f2, and their dependency with the attributes other than primary key are as under:

$$\{C, D, E\} \rightarrow f1 \qquad \{E, F, G\} \rightarrow f2$$

Therefore, according to step-3 of the algorithm, we decompose R into two relations as under: R1= (A, B, C, D, E), R2= (A, B, E, F, G)

Considering that every attributes has a size if 10 bytes, block size of 512 bytes, number of tuples of 1000 in the relation, f1 represents "daily" in ayear domain, and f2 represents "monthly", the query cost per year was found as under:

(a) **For the original relation R**:

Query cost = ((70*1000/512)/2)*(365+13)

=**25,771 blocks**

(b) **For the decomposed relations R1 & R2**: Query cost = ((50*1000/512)/2)*365 + ((50*1000/512)/2)*12

= **18,408 blocks**

The saving in query cost is 28.57%.

Algorithm- 2: Algorithm considering a new relation for common attributes.

1. Identify the distinct frequencies or groups, fi≤i≤n in the relation R based on either frequency of use or groups on purpose of use of the attributes, where n is the total number of occurrence of use-frequencies or groups. If n=1 terminate the algorithm.
2. Except the primary key A_k create dependency functions Ai→fi where Ai is a subset of attributes that occurs during fi.
3. For every dependency functions, create a relation consisting of the primary key A_k and the attributes Ai of the dependency functions.
4. Create a relation for all the attributes common to more than one dependency functions taking A_k as primary key and delete these attributes from the relations created at step-3.
5. After deletion operation at step-4, if one or more relation contains only the primary key A_k, drop that relation.

Algorithm-3: Algorithm keeping common attributes with lowest order relation.

1. Identify the distinct frequencies or groups, fi≤i≤n in the relation R based on either frequency of use or group on purpose of the attributes, where n is the total number of occurrence of use-frequencies or groups. If n=1 terminate the algorithm.
2. Except the primary key A_k, create dependency functions Ai→fi where Ai is a sub-set of attributes that occurs during fi.
3. For every dependency functions, create a relation consisting of the primary key A_k and the attributes Ai of the dependency functions.
4. Keep the common attributes with the lowest order relation and delete them from the rest.
5. After deletion operation at step-4, if one or more relation contains only the primary key A_k, drop that relation.

If we apply algorithm-2 to the relation R= (A, B, C, D, E, F) of section 2.1, we obtain the following three relation:
The following three relations:

R1= (A, B, C, D), R2= (A, B, F, G), R3= (A, B, E)

As we know from section 2.1, the total query of the original relation was 25,771 blocks and that of relations obtained from algorithm-1 was 18,408 blocks. For the decomposed relations R1, R2 and R3 obtained from algorithm-2, the query cost was ((40*1000/512)/2) *365 + ((40*1000/512)/2) *12 + ((30*1000/512)/2) * (365+12) = 25,771 blocks. This showed that algorithm-5.2 did not reduce the query cost over algorithm-1.
If we apply algorithm-3 to the relation of section algorithm-1, we obtain the following two relations:
R1= (A, B, C, D, E), R2= (A, B, F, G)
For the decomposed relations R1 and R2 obtained from algorithm-3, the query cost was ((50*1000/512)/2)*(365+12) + ((40*1000/512)/2)*12=18,877 blocks. This showed that algorithm-3 also did not improve the query cost over algorithm-1. Step-4 of algorithm-3 suggested keeping the common attributes with lowest order relation. We can examine the consequence of keeping the common attributes (i) with the highest order relation and (ii) with the relation having fewer attributes. Changing step-4 of the algorithm-3 accordingly, does not reduce the query cost over algorithm-1.Therefore, it may be resolved that common attributes to be kept with the respective relations where they naturally appears as per dependency functions.

Considering trade-off between excess spaces needed for replication of primary key: One of the objectives of this study was to retain the properties of BCNF/3NF into the resultant relations. To follow this objective, we must replicate the primary key of the original relation into the new relations. This, on the other hand, increases the storage requirement of the database system. Let us consider a relation R= (A, B, C, D, E, F, and G) having primary key {A, B} with the dependency functions as under:

{C}→f1(daily, i.e., 300 times in a year)
{D}→f2(weekly, i.e., 52 times in a year)
{E}→f3(monthly, i.e., 12 times in a year)
{F}→f4(quarterly, i.e., 4 times in a year)
{G}→f5(yearly, i.e., once in a year)

According to the algorithm-1 (which is optimum so far), we obtain following five relations:

R1= (A, B, C), R2= (A, B, D),
R3= (A, B, E), R4= (A, B, F), R5= (A, B, G)

Let the size of each of the attributes is 10 bytes, the block size is 512 bytes and there are 1000 tuples in the relation. Therefore, the query cost of the original relation R is ((70*1000) 512)/2*(300+52+12+4+1)= 24,541 blocks. The query cost of the decomposed relations is ((30 * 1000 /512) /2) * 300 + ((30 * 1000 /512) /2 * 52 + ((30 * 1000 /512) /2) * 12 + ((30 * 1000 /512) /2) * 4 + ((30 * 1000 /512) /2) * 1 = 10,810 blocks which is 55.95% less than that of original relation.

The size of the original relation was 70 bytes whereas the total size of the decomposed relations is 150 bytes, which is an increase of 114% over the size of the original relation. Algorithm-1 awarded us with the reduction in query cost but it also increases the total space requirement. Therefore a trade-off is required between the reduction in query cost and increases in total storage requirement. Adding steps 4 and 5 in the algorithm-5.1, which is shown below, implement this:

Algorithm-4: Decomposition of the BCNF/3NF relation considering trade-off.

```
1. Identify the distinct dependency functions fi ≤ I ≤ n in the relation R based in either "frequency
   of use" or "group of purpose of use" of the attributes, where n is the total number of occurrence
   of use-frequencies or groups. If n =1 terminate the algorithm.
2. Except the primary key $A_k$, create dependency functions Ai→fi where Ai is a sub-set of
   attributes that occurs during fi.
3. For every dependency functions, create a relation consisting of the primary key $A_k$ and the
   attributes that occurs during fi.
4. Let So and $S_d$ be the storage requirement for the original relation and the decomposed relations.
5. while $S_d/S_o \geq k$, where k is a constant to be determined by the DBA.
       (a) merge the two relations having nearest dependency functions.
       Compute $S_d$ for the new relations.
```

After application of steps 1 to 3 of algorithm-5.4 on the relation R of the above example we obtained following five relations:

R1= (A, B, C) with dependency constraint f1 (daily, i.e., 300 times in a year)
R2= (A, B, D) with dependency constraint f2 (weekly, i.e., 52 times in a year)
R3= (A, B, E) with dependency constraint f3 (monthly, i.e., 12 times in a year)
R4= (A, B, F) with dependency constraint f4 (quarterly, i.e., 4 times in a year)
R5= (A, B, G) with dependency constraint f5 (yearly, i.e., once in a year)

Let k=2. We have S_d/S_o=150/70=2.14, which is greater than k. thus, according to steps-4 and 5 of the algorithm, we have to merge R4 and R5 resulting following relations:

R1=(A, B, C) with dependency constraint f1(daily, i.e., 300 times in a year)
R2=(A, B, D) with dependency constraint f2(daily, i.e., 52 times in a year)
R3=(A, B, E) with dependency constraint f3(daily, i.e., 12 times in a year)
R45=(A, B, F, G) with dependency constraint f4(daily, i.e., 4& 1 times in a year)

Now we have S_d/S_o=130/70=1.86 which is less than k. here we exit the loop and stop the algorithm. In the resultant relations, we obtained a reduction of 55.75% in query cost against an increase of 85.71% in space requirement. If we need further reduction in space requirement, we have to take the value of k less than 1.86 and allow increase in percentage of reduction in query cost.

4. Conclusion

The algorithm mainly based on the properties of BCNF/3NF.By applying the algorithm we can reduce the query cost. And we also apply linear search method where each block are scanned, and all records are tested to see weather they satisfy the condition. This algorithm can be applied in different organizations that maintain their software by database to reduce their query cost. There are some problem arise such that for reducing the query cost the space complexity increase successively. We try to overcome the problem by reducing the cost according a little amount space by considering trade –off. And the algorithm was found most economical in the scenario of practical life.

5. REFERENCES

Codd E. (1972). Further normalization of database relational model. Cited in the book “Database Systems” edited by Rustin R.

Codd E. (1974). Recent investigation in Relational Database Systems. Proc. Of the International Federation for Information Processing (IFIP) Congress.

elmasri R. and Navathe S. B. (2000). Fundamentals of Database Systems. 3rd Ed. Addison Wesley (Singapore) Pte. Ltd.

fagin R. (1977). Multivalued Dependencies and a New Normal Form for Relational Database ACM Transactions on Database Systems (TODS), 2:3, September, 1977.

fagin R. (1979). Normal Form and Relational Database Operators. In proc. Of the ACM SIGMOD International Conference on Management of Data. P 153-160.

fagin R. (1981). A Normal Form for Relational Database. That is based on Domains and Keys. ACM Transactions in Database Systems (TODS), 6:3, September, 1981.

grant, j (1987). Logical Introduction to Databases. Harcourt Brace Juvanovich, Inc. Florida.

nambiar K.K., Gopinath B., Nagraj T. and .Manjunath S(1997). Boyee-Codd Normal Form Decomposition. Computer Math. Apple. Vol. 33. no. 4, ppl-3.

Nicolas, j (1978). Mutual Dependencies and some results on Undecomposable relations. In proc. Of the International Conference on Very Large Database(VLDB).

Silverschatz A., Korth H. F., Sudarshan S. (1999). Database System Concepts. 3rd ed. Mc-Graw Hill Companies.