Investigation on the Side Effects of Denormalizing Corporate Databases

Sangwon Lee* · Namgyu Kim** · Songchun Moon***

Abstract

Corporate databases are usually denormalized, due to the data modelers' impetuous belief that denormalization could improve system performance. By providing a logical insight into denormalization, this paper attempts to prevent every database modeler from falling into the denormalization pit. We indicate loopholes in the denormalization advocates' assertions, and then present four criteria to analyze the usefulness and validity of denormalization; 1) the level of concurrency among transactions, 2) the database independence of the application program, 3) the independence between the logical design and the physical one, and 4) the overhead cost to maintain database integrity under various query patterns. This paper also includes experimental results to evaluate performance of denormalized and fully normalized structures under various workloads.

Keywords : Denormalization, Logical Database Design, Normal Forms, Performance Tuning, Relational Model

Received : 2009. 06. 03 Final Acceptance : 2009. 06. 15.

* Ph.D Candidate, KAIST Business School
** Corresponding Author, Assistant Professor, School of Business IT, Kookmin University, e-mail : ngkim@kookmin.ac.kr
*** Professor, KAIST Business School

1. Introduction

While gaining experience in database consulting with a number of enterprises, we have been astonished that the extent of data redundancy comes to as much as 65 per cent in terms of attributes, rather than entities and relationships in most enterprises [Date, 1998 : Moon, 2003 : Sanders, 2001]. The overhead cost of handling unnecessary redundant data could be large if the enterprise manages considerable data. Replication of certain attributes, allowed as foreign keys, could be regarded as exceptional necessary redundancy, because their presence is inevitable for lossless joins. However, this type of redundancy occupies only a relatively small portion of dataset. A large percentage of data redundancy is unnecessary, and lowers data quality [Wang, 1995].

Are there ways to design databases efficiently and economically reducing unnecessary redundancy? Of course, yes. Normalization [Codd, 1972 : Codd, 1974] certainly not only contributes to lower unnecessary data redundancy, but also improves data integrity. From the conceptual view point, normalization is simply a process to eliminate unnecessary redundancy by abstracting only semantically relevant data to be incorporated in the same table. The virtue of normalization could be appreciated in verifying the semantics of knowledge inappropriately caught in the conceptual design phases, rather than a compulsory procedure carried out in logical modeling, that follows on from conceptual modeling. Theoretically, cautious mapping of a conceptual model to a logical model should yield fully normalized tables [Pascal, 2000]. It is only when poor design has bundled multiple entity types into single tables that those tables must undergo an explicit process of normalization. A fully normalized structure ensures every table contains minimal data redundancy. In that structure, update operations on data are performed on only one table, except when foreign keys are used. Accordingly, it could guarantee data integrity by preventing some tables from holding mutually different states of the database.

Even though normalization contributes to improvement of data consistency, some practitioners, however, insist that it may deteriorate the performance from the standpoint of response time [Avison, 2003]. Their assertion is as follows. Higher levels of normalization increase the number of tables in the database, requiring more joins for data manipulation. Since a join is one of the most important factors that could delay response, more normalized tables may lead to the performance deterioration. Thus, it is asserted that denormalization could improve performance [Bolloju, 1997 : Date, 2003 : Janas, 1995 : Tupper, 1998], although denormalization runs counter to normalization theory. Denormalization is a series of process that restores tables to where they were before normalization to improve performance. This assertion involves much risk that could deteriorate overall performance. Denormalization to elevate piecemeal performance is a rough workaround, not a systematic methodology [Schkolnick, 1980] for data design.

• Example 1(Risk of Denormalization) :

We now demonstrate the risk of denormalization using an example that aids our conceptual understanding. Imagine food preparation in a large restaurant. In general, flavorings, such as sugar, pepper, and salt, are stored in flavoring boxes. Likewise, vegetables are stored with other vegetables, and meat is stored together flesh. When cooked, ingredients are selected from each repository and used for a cuisine as user views in database systems. Assume that a chef prepares an intermediate-stage food, forecasting many orders for beef pilaf. If he takes orders for beef pilaf, as they arise, he could cook them faster than otherwise is the case. Assume at one stage his manager informs him that beef supplied today is insanitary. The cook must replace the existing beef with new ones from the freezer. He must determine all his part-processed dishes that were pre-prepared for data integrity. In the meantime, if he has a rush of orders for new dishes beyond his forecast, he would comply with orders more slowly than others who did not pre-prepare the food, that is, who classified ingredients, exhibiting application independency. A wise cook would choose the following method instead of mixing original ingredients for the intermediate-stage dishes in advance. He could find out information more quickly using labeling for frequently used ingredients as data indexing.

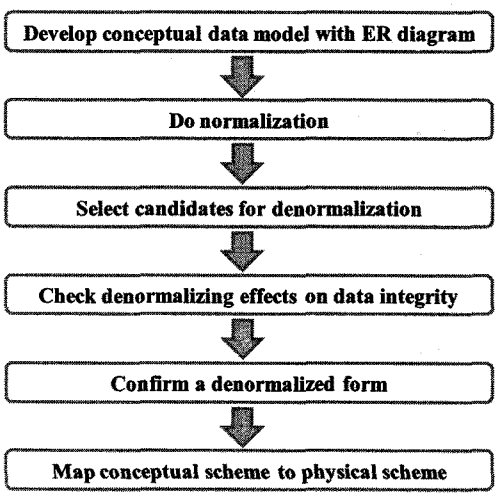
End of Example 1 ■

Building queries dependent on logical schema may damage the independence between the application program and database system, because the schema must be modified if the usage patterns of the application program are changed. We are able to infer the risk of denormalization through the simple example above. The mechanism that combines several tables for queries that are expected to occur frequently must, inevitably, incur overhead to maintain integrity when updating data. If performance matters, you should retain the original normalized tables and use efficient widely used indexing and clustering techniques in the physical design phase. To conclude, the proposition that denormalization could elevate performance is caused by mistaking logical modeling for physical modeling [Bock, 2002]. That is to say, the system performance could be enhanced, not at the logical design phase, but at physical design. At the logical level, we must normalize to eliminate unnecessary redundancy and maintain integrity. Conversely, at the physical level, we must handle performance requests to focus on query analysis and access pattern analysis using indexing or clustering. Reduce your prejudice in denormalization [Wang, 1995]; it could be recognized, not as a key solution, but as a desperate measure. Denormalization rarely improves system information performance.

2. Research Overview

Although denormalization is utilized in various types of database design, it hitherto lacks solid principles and guidelines. Shokolnick and

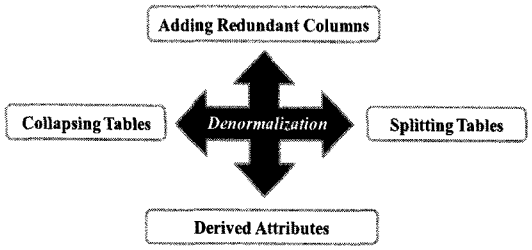
Sorenson introduced denormalization [Pascal, 2002], but insufficiently justify its use. They stressed the need for semantic constraints in the denormalization process that are meaningful as a new concept from an academic viewpoint. Although trials for the denormalization process attempt to face the one issue that lacks solid principles and guidelines in denormalization, denormalization still lacks concrete. A trial for a model of the denormalization process was introduced with a stipulated policy that the database designer should, prior to the denormalization procedure, develop a logical entity relationship model that describes the cardinality for each relationship or volume estimation for each entity and defines process decomposition for the application via a dataflow diagram. The database design cycle with denormalization starts from the conceptual data model and ends in mapping it to physical schema <Figure 1>. That model considers many criteria for denormalization; it handles performance requirements needed for the business and other considerations. The performance requirements include application performance and on-line response time. Other considerations include minimum number of data access paths, minimum amount of storage, transaction frequency, update or query transaction, future application development, maintenance consideration, volatility of application requirements, and so forth.



<Figure 1> Database design phases including denormalization

Though that approach does not restrict the data modeler's intuition, it represents an effort to systematically categorize denormalization. The first trial to categorize denormalization focused on how the data will be used. The attempt to assert denormalization should be considered carefully by a data modeler and it is dependent upon one's subjectivity. Another categorization of denormalization was structured by steps such as adding redundant columns, collapsing tables, splitting tables, and derived data [Pascal, 2002]. There are four denormalization strategies <Figure 2>. Denormalization was introduced at the logical or base relation level. To validate the logical model, denormalization with ERD is carried out. At the time when the structure with logical objects is transformed to physical ones, denormalization decreases the number of logical objects. It could shorten application call paths navigating the database objects. When the structure of physical model is changed, denormalization creates, moves, and consolidates entities or attributes using of redundancy or synthetic keys. Denormalization was introduced at the physical storage, not at the logical or base relation level. Denormalization was regarded as an intermediate step be-

tween the logical level and the physical one. The pre-physical design process performs the logical database refinement process with a practical view.



〈Figure 2〉 Four strategies of denormalization

Unreasonable denormalization could ruin a logical design in that it does not reflect future additional development and maintenance. One of strong points of denormalization is that a more denormalized data model could provide better performance in extending the data model. Especially when a data modeler designs a multidimensional structure, one could solve the complexity of hierarchies by means of facts and dimensions. A data modeler could also navigate the data structure with a more intuitive view. Since there is the trade-off between normalization and denormalization, a data modeler must give careful consideration to denormalization and use it only in the specific situations in which two entities have a one-to-one relationship and a many-to-many relationship with non-key attributes. In the meantime, an alternative to denormalization, clustering, was introduced. Clustering could be useful for information system analysis and design in object-oriented techniques, but it accepts only specific type of physical data structure.

3. Fallacious Doctrine-Denormalization for Performance Enhancement

Taking a comprise attitude for denormalization, the data modeler should have both appropriate knowledge about application requirements and expertise in the business in carrying out the refinement process. He or she should pay attention to optimal performance being aware of flexibility requirements; the database management system, operating system, and data update frequency. In this section, we provide some logical reasoning to indicate loopholes in the assertion of denormalization advocates. We will show a general example for denormalization (Example 2). The table structure in Example 2 will be used for consistent and logical expansion.

- Example 2(Denormalization Case) : Let us assume that following business rules are valid for a simple application.

Rule 1 : A student can have one or more cars and a car can belong to only one student.

Rule 2 : A student can have one alma mater.

The fully normalized schema for this application results in three tables <Figure 3>. In the system organized as in <Figure 3>, assume there is a frequently used query, "Select cars that belong to the student whose name is XXX." It is necessary to join two tables, *Student* and *Car* to execute this query. The *Car_Student* table in

<Figure 4> is created by composing two tables, *Student* and *Car*, using denormalization. We would expect that the response time in <Figure 4> is shorter than that in <Figure 3>, are the retrieval gets the result from only one table without a join. End of Example 2 ■

3.1 Transaction Concurrency Degradation

It is often the case that semantically separated transactions cannot be performed concurrently in a denormalized structure, even though they can be done in a fully normalized structure. For concurrency control, most systems use a locking mechanism in which transactions acquire or release locks on records. However, with huge tables created by denormalization, transaction acquires more data locks on a larger scale than is necessary. This could result in declining concurrency. Example 3 shows that denormalization would decrease system performance, due to unnecessary blocking of concurrent execution of semantically irrelative transactions.

- Example 3 (Performance Degradation Resulted from Unnecessary Locks on

Student

S_ID	Name	Phone	Alma_Master
1000	Michael	111-1111	U-Alpha
1001	John	222-2222	U-Beta
1002	Peter	333-3333	U-Beta

School

School_ID	Grade	Location
U-Alpha	A	North
U-Beta	B	South

Car

Car_ID	Maker_ID	Mileage	S_ID
0001	GM	30,000	1000
0002	Toyota	40,000	1000
0003	Samsung	50,000	1000
0004	GM	40,000	1001
0005	Samsung	50,000	1002

<Figure 3> Fully normalized table structure

Car_Student

Car_ID	Maker_ID	Mileage	S_ID	Name	Phone	Alma_Master
0001	GM	30,000	1000	Michael	111-1111	U-Alpha
0002	Toyota	40,000	1000	Michael	111-1111	U-Alpha
0003	Samsung	50,000	1000	Michael	111-1111	U-Alpha
0004	GM	40,000	1001	John	222-2222	U-Beta
0005	Samsung	50,000	1002	Peter	333-3333	U-Beta

School

School_ID	Grade	Location
U-Alpha	A	North
U-Beta	B	South

<Figure 4> Denormalization comprising two tables

Data) : Recall <Figure 3> One transaction *T-Student* updates the student's phone number; the other, *T-Car*, updates the car mileage. They are performed concurrently, because they access the phone number and mileage stored in separated tables. It is intuitively suitable that these two semantically separated transactions are performed concurrently. However, in <Figure 4>, execution of *T-Car* is not allowed if it tries to access the car with "Car_ID = 0001" if another transaction *T-Student* already holds an update lock on record whose "S_ID = Michael."
End of Example 3 ■

3.2 Data Independence Infringement

The proposition that denormalization is carried out based on access pattern analysis is equal to abandoning well-balanced performance, because the proposition infringes independence between the application program and database. Advocates of denormalization insist that correct and detailed analysis for access pattern should be the prerequisite to improve system performance, whilst data integrity is guaranteed. The schema at the logical level may be changed according to the application programs or usage types. Unfortunately, this proposition infringes independence between the application program and database, one of the most important principles of database systems. When a new application program is developed due to a changing business environment, or when user access patterns are changed, the application-dependent table structure should be changed according to the application program.

- Example 4(Degree of Data Independence Infringement) : Recall Example 2. The structure in <Figure 4> is obtained by denormalizing the structure in <Figure 3> with the expectation that the query, "Select cars that belong to the student whose name is XXX," would be frequently issued. Contrary to expectations, however, assume that the query, "Update the phone of student with S_ID is 1000," is performed frequently due to the changed business environment. This change may degrade overall performance, because of overhead for integrity maintenance. Now, let us assume another type of change. In general, frequently-accessed tables are changed while retrieval operations still occur more frequently than update ones. Assume the structure in <Figure 3>, in which there is a frequently issued query, "What is the grade of school in which the student named XXX graduated?" To reply to this query, from the viewpoint of denormalization, a join between two tables, Student and School, is needed. This results in the structure shown in <Figure 5>.
  Conversely, assume that this query is performed in the denormalized structure <Figure 4>. In this case, the information about student name is stored in the table, *Car_Student*. Therefore, to reply to this query, the join between two tables, *Car_Student* and *School* is needed; this results

in <Figure 6>. Since the joined table in <Figure 6> is larger than the one in <Figure 5>, it is axiomatic that its response time is longer. Thus, even though denormalization is performed cautiously through access pattern analysis on the present program, the alteration of the application program or access pattern could affect overall performance.

End of Example 4 ■

3.3 Boundary Wreck between Logical Model and Physical Model

Advocates for denormalization who confuse the logical design phase with the physical one in data modeling would be the last to perceive and use performance improvement correctly. This could degrade system performance even more. There has been so much research on improvement of performance in the physical design phase; this is known as tuning. There are two representative methods. Indexing manages pointers for frequently accessed data. Clustering stores semantically related tables in an adjacent place. If we reluctantly accept that performance improvement could be achieved by artificially manipulating a fully normalized structure, it does not imply the existing normalized tables are designed incorrectly, but that nor-

Student_School

S_ID	Name	Phone	Alma_Master	Grade	Location
1000	Michael	111-1111	U-Alpha	A	North
1001	John	222-2222	U-Beta	B	South
1002	Peter	333-3333	U-Beta	B	South

Car

Car_ID	Maker_ID	Mileage	S_ID
0001	GM	30,000	1000
0002	Toyota	40,000	1000
0003	Samsung	50,000	1000
0004	GM	40,000	1001
0005	Samsung	50,000	1002

〈Figure 5〉 Joined tables from fully normalized structure

Car_Student_School

Car_ID	Maker_ID	Mileage	S_ID	Name	Phone	Alma_Master	Grade	Location
0001	GM	30,000	1000	Michael	111-1111	U-Alpha	A	North
0002	Toyota	40,000	1000	Michael	111-1111	U-Alpha	A	North
0003	Samsung	50,000	1000	Michael	111-1111	U-Alpha	A	North
0004	GM	40,000	1001	John	222-2222	U-Beta	B	South
0005	Samsung	50,000	1002	Peter	333-3333	U-Beta	B	South

〈Figure 6〉 Joined table from denormalized structure

malized structure could be improved by tuning in the physical design phase.

Even though performance tuning is executed at the physical design phase, it is not necessary to recount that the logical modeling must have the fully normalized table structure without any contamination. The three precedent phases of the physical design are requirement analysis, conceptual modeling, and logical modeling. Nevertheless, deliverables of the three design phases, i.e., requirement specifications, conceptual enterprise model, and tables, do not exist solely to implement the final physical database. They play their own roles as tools to communicate and exchange information in the relevant phase. Let us recall the previous example. In <Figure 3>, Car table has *S_ID* as a role of foreign key. However, no one will add *S_ID* to "*Car*" entity as an attribute on its entity relationship model, since *S_ID* is obviously not an intrinsic attribute owned by the entity "*Car*." Let us assume that someone adds an attribute *S_ID* to entity "*Car*" to the entity relationship model by guessing that it will be added in the *Car* table as a foreign key from the viewpoint of necessary redundancy. Few database modelers clearly understand this situation. A similar relationship exists between the logical and physical model. The goal of the logical model should be to create normalized tables based on several dependencies such as functional dependency, multi-valued dependency, and join dependency. All the effort to improve performance using indexing and clustering are applicable only to physical modeling. We must forbid incompetent database modelers from confusing logical and physical modeling, by embracing denormalization that should be only the part of the tuning process.

3.4 Increased Overheads Cost to Maintain Data Integrity

Even though decreasing the number of physical tables could decrease the delay caused by a join, to decrease the number of tables increases unnecessary data redundancy and gives rise to a maintenance cost for integrity. If any data in tables are modified, other tables having the same or redundant data must be modified at the same time. This process is compulsory to maintain the integrity of data; it is guaranteed by triggers among related items. The higher the degree of unnecessary data redundancy, the higher the maintenance cost is to build triggers. The next example helps understand this phenomenon (Example 5).

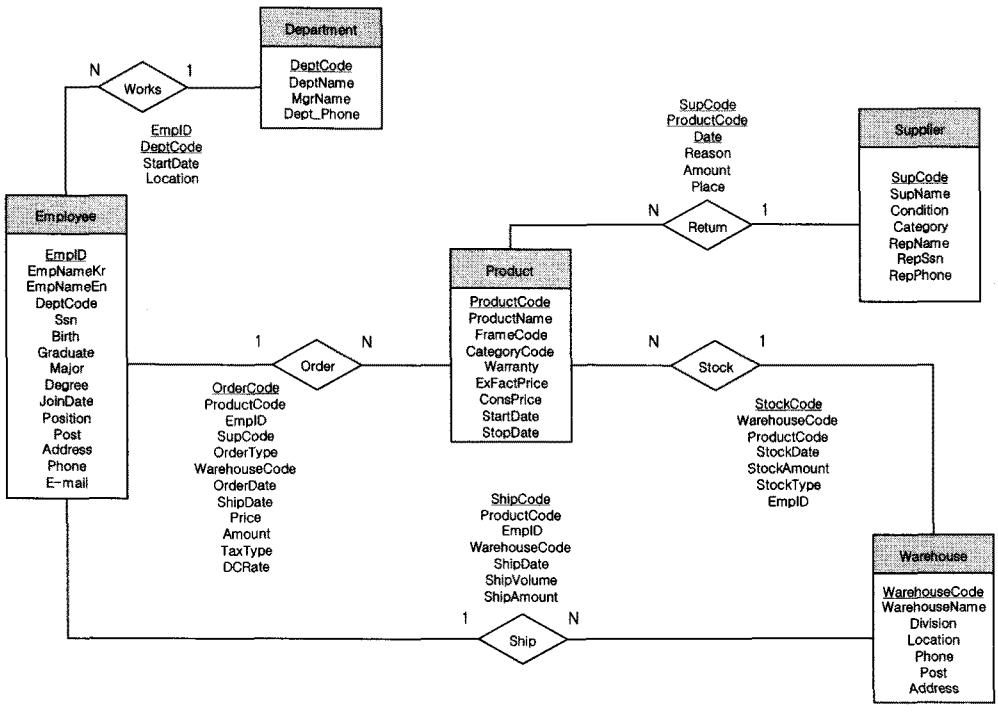
- Example 5 (Performance Degradation Resulted from Data Redundancy) : Recall Example 1. A denormalized structure in <Figure 4> is optimized for the query such as "Select cars that belong to the student whose name is XXX." However, we should consider integrity maintenance. Assume that the query "Update the phone of student with *S_ID* = 1000" is issued. In the case of <Figure 3>, we need to update just one tuple in 3 rows and 4 columns table, Student. Conversely, we must update three tuples for *S_ID* = 1000 in 5 rows and 7 columns table *Car_Student*,

in the case of <Figure 4>. That is, if the update query is performed more frequently than the retrieval one, the denormalized structure degrades overall performance, because redundant data increases the delay time for integrity maintenance.
End of Example 5 ■

In contrast to compromising response time for the sake of performance, data integrity must be observed at all cost. Hence, even advocates of denormalization should weight data integrity over system performance. Rather, they advise that denormalization should be carried out only in the system where retrieval is expected to be executed more frequently than an update. However, research on practical business matters reveals that in average more than 50 percent of queries are update. Based on this observation, we assert that practitioners seldom gain from denormalization.

4. Experiment for Performance Evaluation

Even though, for retrieval queries, the response time in the denormalized database would be shorter than that in normalized database, it is common that, for update queries, the response time in the denormalized database is longer that that in the normalized database. Denormalization has proposed various techniques and methods to improve performance. Many system designers have used it in designing enterprise databases to decrease query response time. However, frequently designer's



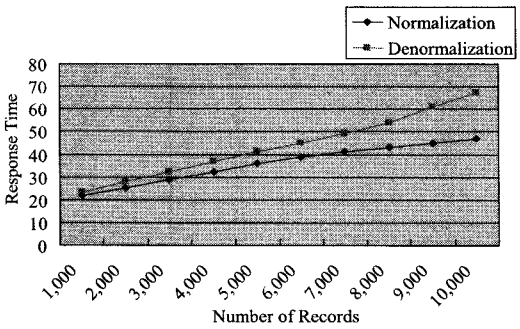
<Figure 7> Normalized ERD for sales information system

advocate denormalization over normalization on the basis of their experience; in contrast to their expectation to improve application program performance, focusing on decreasing the response time for retrieval queries, ignores update queries to the detriment of the overall system.

From the viewpoint of update queries, as well as retrieval ones, the normalized model is superior in the general enterprise environment. By an experiment, we support the fact that denormalization does not contribute to performance improvement. In this section, we summarize the experimental results in [Roh, 2004] to support our assertion. The experiment used a normalized model <Figure 7> for a defined sales information system and then designed its denormalized one to compare system performance. The denormalization was performed by row-column conversion and column duplication. We used the server system Samsung SENS 640, MS Windows XP Server as the operating system, and MS SQL Server 2000 Enterprise Edition as the database management system. The denormalized database could have performed better than normalized one, when there are only retrieval queries. However, the performance of denormalization is lower than that of normalization in general when there are both update and retrieval queries. Especially, many users can lead to frequent locking which results in dramatic deterioration of system performance.

4.1 Average Response Time

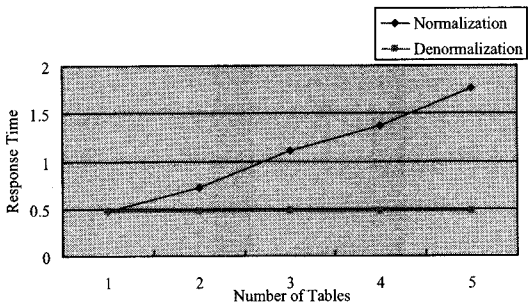
System performance is affected by the database volume as the subject of queries and generally increases as its database volume increases. System performance meanwhile differs according to the efficiency of the designed data model. We observed the average response time with changing the number of records in two database models; one designed by normalization, the other by denormalization <Figure 8>. In this experiment, we varied the number of records from 1,000 to 10,000. When we consider the difference between the performance of real-world servers and that of the server used in this experiment, 10,000 records would be appropriate scale for about 100,000 records in real-world. The average response time increases linearly in the two models as the number of records increases. Though the normalized database includes joins on queries, its average response time is short. This is due to the relatively small size of tables; hence, the fields for the joins reside in main memory in the normalized database. The steep increase of the average response time in the denormalized database is related to the real-time synchronization for data integrity. In this experiment, some table triggers are used for data integrity, so that redundant columns may be real-timely synchronized. Thus, as the total number of records increases, the records changed by triggers increase. This could degrade performance of all the systems. Even though triggers affect the experimental result, they should be sanctioned by usage in most experiments. Apart from the effects of triggers, from the viewpoint of the average response time, as the scale of the database increases, the denormalized database surely raises concerns for system performance.



<Figure 8> Average response time

4.2 Response Time for Tables Joined

One of the most important factors in denormalization is to reduce of joins for system performance. We observed the average response time against changing the number of tables joined in the normalized database and the denormalized one <Figure 9>. To reduce the occasions of conditional joins, the denormalized tables include all the columns that would be created by conditional joins. In the denormalized database, the average response time is not affected by the number of conditional joins since additional table joins do not occur. In the normalized database, the system response time however incrementally increases as the number of tables joined conditionally increases. The response time in retrieving data without the initial conditional joins requires only 0.47 seconds, and yet the response time in retrieving data with five tables joined conditionally increases to 1.76 seconds. Since the increase in response time for four tables joined is at least 1.29 seconds, the average response time for one table joined is as short as 0.32 second. The instantaneous and so then negligible time, 0.32 second, for one table joined, however, is slightly affected by the increase of number of records. This is due to the increased time for physical disk access for conditional joins, which is insignificant.

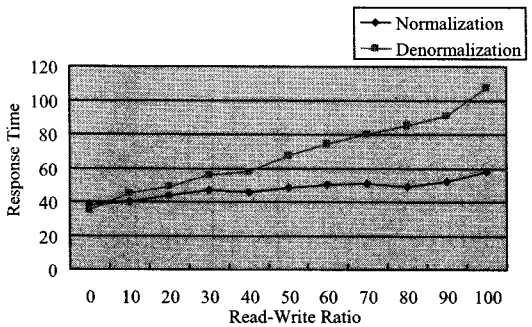


<Figure 9> Response time for tables joined

4.3 Response Time for Read-Write Ratio

We observed the average response time according to query type in the normalized and denormalized database models <Figure 10>. We changed the ratio of write queries to read queries from 0 percent to 100 percent in the two database models. The average response time in the denormalized database is less than in the normalized database, when the ratio of write queries to read queries is in the neighborhood of 0 percent. In contrast, as the ratio of write queries to read queries increases, the average response time in the normalized database is longer than that in the denormalized database. This result shows that, although the denormalized database is more efficient in read queries than the normalized one, it is less efficient in write queries than the normalized one, since additional conditional joins are needed in input queries. The additional conditional joins

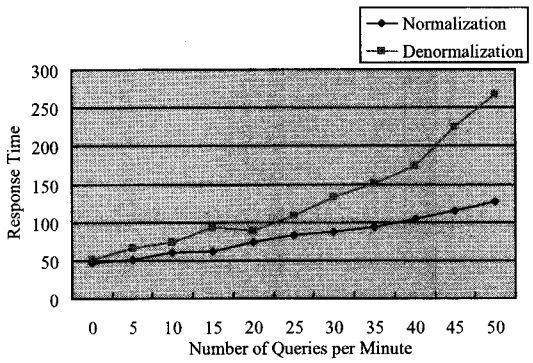
are compulsory in input queries to fill the values of redundant data that tend to decrease the number of conditional joins in read queries. The performance of the system is influenced by the real-time synchronization operations for redundant columns, table triggers, which are used for data integrity in denormalization. Denormalization could be useful in the special situation of information systems where read queries are in the majority. Nevertheless, normalization could be useful in most information systems where the number of write queries is greater than that of read queries, or where the read queries and the write ones are used together for online transactions.



<Figure 10> Response time for read-write ratio

4.4 Response Time for Number of Queries per Minute

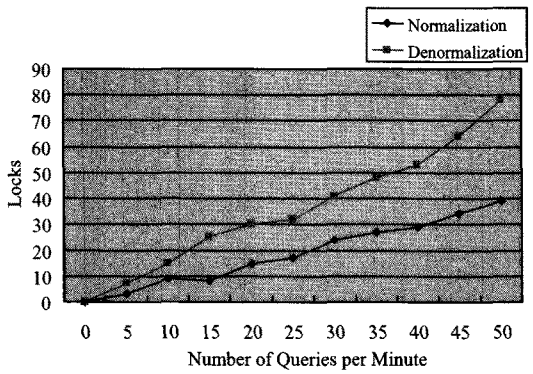
We observed the average response time based on the number of queries per minute <Figure 11>. The ratio of read and write queries is one to one. As the number of queries increases, the average response time in the denormalized database increases steeper than that in the normalized database, because the average response time of write queries in the denormalized database is longer than that in the normalized database. It could be due to the increase in the number of queries leads to the increase of concurrent transactions and then increases locking.



<Figure 11> Response time for number of queries per minute

4.5 Locks for Number of Queries per Minute

The occurrence of locks can be used as an auxiliary measure of system performance, because it delays transaction operations and then lowers system performance. We observed the number of locks according to the number of queries per minute <Figure 12>. The number of locks in the denormalized database increases more than that in the normalized one, due to the relatively slow process of write queries in the denormalized database and the increase in the possibility that additional locks occur. The system performance, i.e., the average response time, is intimately associated with the occurrence of locks. The occurrence of locks is in inverse proportion to the system performance. As the number of queries or system users increases, the number of locks also lowers system performance.



〈Figure 12〉 Locks for number of queries per minute

5. Conclusions

Denormalization in this manner cripples enterprise data by ignoring data integrity. We hope that database modelers and end users do not fall into the fallacious argument that denormalization always improves system performance. In this article, we provide some logical reasoning to indicate loopholes in the assertions of denormalization advocates. First, denormalization lowers concurrency among transactions, because the record size usually increases in denormalized structures. Second, denormalization based on query pattern analysis infringes independence between the application program and database. Third, mentioning performance issues in the logical design phase makes unsophisticated database modelers confuse logical and physical modeling. Finally, increased overheads to maintain data integrity e deteriorates overall system performance when update queries are frequently issued.

However, the experiment in this paper has some limitations as follows. In our experiment, we used a small data set containing only 10,000 records. When we consider the difference between the performance of real-world servers and that of the server used in this experiment, 10,000 records would be appropriate scale for about 100,000 records in real-world. However, we need to perform intensive experiments not on a small data set but on a real-world database to investigate various phenomena which occurs in commercial large-scale database systems.

References

[1] Avison, D. E. and Fitzgerald, G., "Where now for development methodologies?", *Communications of ACM*, Vol. 46, No. 1, 2003, pp. 78-82.

[2] Bock, D. B. and Schrage, J. F., "Denormalization guidelines for base and transaction tables", *ACM Special Interest Group on Computer Science Education*, Vol. 34, No. 4, 2002, pp. 129-133.

[3] Bolloju, N. and Toraskar, K., "Data clustering for effective mapping of object modelings to relational models", *Journal of Database Management*, Vol. 8, No. 4, 1997, pp. 16-23.

[4] Codd, E., Relational completeness of data base sublanguages, *Data Base Systems, Englewood Cliffs*, NJ, USA, 1972.

[5] Codd, E., Recent investigations in relational data base systems, In *Proceedings of International Federation for Information Processing Congress*, Stockholm, Sweden, 1974, pp. 15-20.

[6] Date, C. J., Normalization is no panacea,

Database Programming and Design Online, Web site : http://www.dbpd.com, 1998.

[7] Date, C. J. and Codd, E. F., "A tribute and personal memoir", *SIGMOD Record*, Vol. 32, No. 4, 2003, pp. 4-13.

[8] Janas, J. M., A systematic approach to database denormalization, In *Proceedings of the 17th International Conference on Information Technology Interfaces*, Pula, Croatia, 1995, pp. 323-328.

[9] Moon, S., "Unclassified data is merely garbage : data modeling is more crucial than programming", *Hitech Information*, Vol. 14, No. 9, 2003, pp. 50-51.

[10] Pascal, F., *Practical Issues in Database Management*, Massachusetts : Addison-Wesley, 2000.

[11] Pascal, F., The dangerous illusion : denormalization, performance and integrity, Part 1, *DM Review Magazine*, Web site : http://www.dmreview.com, 2002.

[12] Roh, B. U., *Inadequacy of denormalization in data modeling*, Master's Thesis, Korea Advanced Institute of Science and Technology, 2004.

[13] Sanders, G. L. and Shin, S., "Denormalization effects on performance of RDBMS", In *Proceedings of the 34th Hawaii International Conference on System Sciences*, 2001, pp. 1-9.

[14] Schkolnick, M. and Sorenson, P., "Denormalization : a performance oriented database design technique", In *Proceedings of International Association for the Analog Computation*, 1980, pp. 363-377.

[15] Tooper, C., "The physics of logical modeling", *Database Programming and Design*, Vol. 11, No. 9, 1998.

[16] Wang, R. Y., Storey, V. C., and Firth, C. P., "A framework for analysis of data quality research", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 7, 1995, pp. 623-640.

Author Profile

Sangwon Lee

Sangwon Lee received the B.S. degree in Mathematics from Hanyang University, Korea in 1995 and M.B.A. degree in Management Information Systems from Korea Advanced Institute of Science and Technology (KAIST) in 2002. He is a Ph.D. candidate in Management Engineering of KAIST. His current research interests include enterprise data modeling and data mining for enterprise information systems.

Namgyu Kim

Namgyu Kim received the B.S. degree in Computer Engineering from Seoul National University in 1998 and Ph.D. degree in Management Engineering from Korea Advanced Institute of Science and Technology (KAIST) in 2007. He has been working for Kookmin University since then. His current research interests include enterprise data modeling and data mining.

Songchun Moon

Songchun Moon received his Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 1985. He has been working for Korea Advanced Institute of Science and Technology (KAIST) since then. He has developed a multi-user relational database management system, IM, which is the first prototype in Korea in 1990 and a distributed database management system, DIME, first ever in Korea in 1992. His research interests include enterprise data modeling, security, privacy, piracy, and data warehousing.

◈ 이 논문은 2009년 06월 03일 접수하여 1차 수정을 거쳐 2009년 06월 15일 게재확정되었습니다.