Denormalization Effects on Performance of RDBMS

G. Lawrence Sanders
Management Science & Systems
State Univeristy of New York at Buffalo
mgtsand@mgt.buffalo.edu

Seungkyoon Shin
Management Science & Systems
State Univeristy of New York at Buffalo
ss27@acsu.buffalo.edu

Abstract

In this paper, we present a practical view of denormalization, and provide fundamental guidelines for incorporating denormalization. We have suggested, using denormalization as an intermediate step between logical and physical modeling to be used as an analytic procedure for the design of the applications requirements criteria. Relational algebra and query trees are used to examine the effect on the performance of relational systems. The guidelines and methodology presented are sufficiently general, and they can be applicable to most databases. It is concluded that denormalization can enhance query performance when it is deployed with a complete understanding of application requirements.

1. Normalization vs. Denormalization

Normalization is the process of grouping attributes into refined structures. The normal forms and the process of normalization have been studied by many researchers, since Codd [5] initiated the subject. First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF) were the only forms originally proposed by Codd, and they are the normal forms supported by commercial case tools. The higher form of normalization such as Boyce/Codd Normal Form, the Fourth Normal Form (4NF), and the Fifth Normal Form (5NF) are academically important but are not widely implemented.

The objective of normalization is to organize data into stable structures, and thereby minimize update anomalies and maximize data accessibility. Although normalization is generally regarded as the rule for the statue of relational database design, there are still times when database designers may turn to denormalizing a database in order to enhance performance and ease of use. Even though normalization results in many benefits, there is at least one major drawback – poor system performance [22],[13], [15], [24], [18], [7].

Normalization can also be used as a supplemental tool to provide an additional check on the stability and integrity of an Entity Relationship Diagram (ERD) and produce naturally normalized relational schemas. However, converting the conceptual entity-relationship model into database tables does not guarantee the best performance, nor the ideal geometrical distribution of the data [20]. Thus, even though conceptual data models encourage us to generalize and consolidate entities to better understand the relationships between them, such generalization can lead to more complicated database access path [2]. Furthermore, normalization can create retrieval inefficiencies where a comparatively small amount of information is being sought and retrieved from the database [24]. As a consequence, database designers occasionally trade off the aesthetics of data normalization with the reality of system performance.

Denormalization can be described as a process for reducing the degree of normalization with the aim of improving query processing performance. One of the main purposes of denormalization is to reduce the number of physical tables that must be accessed to retrieve the desired data by reducing the number of joins needed to derive a query answer [17].

Denormalization has been utilized in many strategic database implementations to boost database performance and reduce query response times. One of the most useful areas for applying denormalization techniques is in data warehousing implementations for data mining transactions. The typical data warehouse is a subject-oriented corporate database that involves multiple data models implemented on multiple platforms and architectures. The goal of the data warehouse is to put enterprise data at the disposal of organizational decision makers. The retrieval, conversion and migration of data from the source to the target computing environments is one of many tasks charged to the data warehouse administrator. The transformation of that data in a manner that ensures that the target database holds only accurate, timely, integrated, valid and credible data represents the most complex task on the technical agenda. This is also an area where the tools and techniques are not abundant. It is not enough to simply capture the data needed for the

data warehouse. It is also necessary to optimize the potential of the data warehouse.

Denormalization is particularly useful in dealing with the proliferation of star schemas that are found in many data warehouse implementations. In this case, denormalization provides better performance and a more intuitive data structure for data warehouse users to navigate [1]. The goal of most analytical processes against the data warehouse is to access aggregates such as sums, averages, and trends. While typical production systems usually contain only the basic data, data warehousing users expect to find aggregated and time-series data that is in a form ready for immediate display.

Important and common components in a data warehouse that are good candidates for denormalization include: multidimensional analysis in a complex hierarchy, aggregation, and complicated calculations. Time is a good example of a multidimensional hierarchy (e.g. year, quarter, month, and date). Basic design decisions such as how many dimensions to define and what facts to aggregate can affect database size and query performance.

Although denormalization techniques have been utilized for various types of database design, denormalization is still one issue that lacks solid principles and guidelines. There has been little research related to illustrating how denormalization enhances database performance and reduces query response time. This paper aims at providing a comprehensive guideline regarding when and how to effectively exercise denormalization. The main contribution is to provide unambiguous principles for conducting denormalization. In regards to the organization of the paper, we start by giving an overview of prior research on denormalization in Section 2. A generally applicable denormalization process model and commonly accepted denormalization techniques are to be presented, in Sections 3 and 4. In Section 5 respectively, we use relational algebra and query trees to develop a mechanism to access the effect of denormalization on the performance of Relational Database Management Systems (RDBMS). Finally, Section 6 presents a summary of our findings.

2. Previous work on denormalization

It is commonly believed by database professionals and researchers that normalization degrades response time. A full normalization results in a number of logically separate entities that, in turn, result in even more physically separate stored files. The net effect is that join processing against normalized tables requires an additional amount of system resources.

Normalization may also cause significant inefficiencies when there are few updates and many query retrievals involving a large number of join operations. On the other hand, denormalization may boost query speed, but also degrade data integrity.

The trade-offs inherent in normalization/denormalization should be a natural area for scholarly activity. However, this has not been the case. The pioneering work by Schkolnick and Sorenson [22] introduced the notion of denormalization. In that work, They argues that, to improve performance quality of a database design, the data model viewed by the user has to be capable of notifying the user about semantic constraints. The goals of this paper are to illustrate how semantic constraints can be exhibited and to illustrate an algorithm to carry out the denormalization process.

Hanus [12] developed a list of normalization and denormalization types, and suggested that denormalization should be carefully used according to how the data will be used. A limitation of this work is that the approach to be used in denormalization was not described in sufficient detail.

Tupper [23] proposed two separate dimensions for denormalization. In the first approach, the ERD is used to collapse the number of logical objects in the model. This will in effect shorten application call paths that traverse the database objects when the structure is transformed from logical to physical objects. This approach should be exercised when validating the logical model. In the second approach, denormalization is accomplished by moving or consolidating entities, and creating entities or attributes to facilitate special requests, and by introducing redundancy or synthetic keys to encompass the movement or change of structure within the physical model. One deficiency in this approach is that future development and maintenance are not considered.

Date [8] argued that denormalization should be done at the physical storage level, not at the logical or base relation level. He argued further that denormalization usually has the side effect of corrupting a clear logical design with undesirable consequences.

Hahnke [11] illustrated the positive effects of denormalization on analytical business applications. In designing analytical systems, a more denormalized data model provides better performance and a more intuitive data structure for users to navigate. The two primary components of denormalized data models, facts and dimensions, are also provided to solve the complexity of hierarchies that are an important component of multidimensional analysis when they support drill-down functions in navigation functions. (Facts are the data elements of interest contained in the result set returned by

a query and dimensions define the constraints used to select the facts.)

Cerpa [4] proposes a pre-physical database design process as an intermediate step between logical and physical modeling and provides a practical view of logical database refinement before the physical design. He maintained that the refinement process requires a high level of expertise on the part of the database designer, as well as appropriate knowledge of application requirements.

Rodgers [20] discusses the general trade-offs of denormalization and some of the more common situations in which a database designer should consider denormalization. For example, denormalization is useful when there are two entities with a One-to-One relationship and a Many-to-Many relationship with non-key attributes.

Bolloju and Toraskar [2] present an approach to avoid or minimize the need for denormalization. They introduce the concept of data clustering as an alternative to denormalization. This approach is important in view of the current popularity of object-oriented techniques for information system analysis and design. One problem with the use of data clustering is that it is limited to a particular type of physical data organization.

Another possible drawback of denormalization relates to flexibility. Coleman [6] argues that denormalization decisions usually involve the trade-offs between flexibility and performance, and denormalization requires an understanding of flexibility requirements, awareness of the update frequency of the data, and knowledge of how the database management system, the operating system and the hardware work together to deliver optimal performance.

It is apparent from the previous discussions that there has been little research regarding a comprehensive taxonomy and procedure model for the denormalization process. The next question of this paper sets the stage for the development of an analytical approach for denormalization.

3. A denormalization process model

As noted above, the primary goals of denormalization are to improve query performance and to present the end-user with a less complex and more user-oriented view of data. This is in part accomplished by reducing the number of physical tables and reducing the number of actual joins necessary to derive the answer to a query.

As a rule, denormalization should be considered only when performance is an issue and then only after there has been a thorough analysis of the various impacted

systems. For example, if additional system development is under way, denormalization may not be appropriate and could lead to additional data anomalies and reduce flexibility, integrity, and accessibility. Consequently, denormalization should be deployed only when performance issues indicate that it is needed, and then only after there has been a thorough analysis of the problem domain and the application requirements.

It has been asserted by Inmon [14] that data should be firstly normalized as the design is being conceptualized and then denormalized in response to the performance requirements. Prior to the denormalization procedure, the database designer should develop a logical entity relationship model that indicates; 1) cardinality for each relationship, 2) volume estimation for each entity, and defines 3) process decompositions for the application. Point 3 could be accomplished via data flow diagrams. Figure 1 illustrates the up-front activities that that should be attended to before proceeding to denormalization.

- Development of a conceptual data model (ER Diagram).
- Refinement and Normalization.
- Identifying candidates for denormalization.
- Determining the effect of denormalizing entities on data integrity.
- Identifying what form the denormalized entity may take.
- Map conceptual scheme to physical scheme.

Figure 1. DB Design Cycle with Denormalization.

A typical implementation of the database design process includes the following phases: conceptual database design, logical database design, and physical database design [21]. Conceptual data models encourage the generalization and consolidation of entities. In practice, however, the generalization process leads to more complex database access paths. Physical database design means merely converting the conceptual entity-relationship model into database tables. However, this approach does not account for the trade-offs necessary for performance or for geographic distribution of the database. Therefore, denormalization can be separated from both steps because it involves aspects that are neither purely logical nor purely physical. We propose that the denormalization process should be implemented between the data model mapping and the physical database design so that the procedure can be based on logical and physical database design.

- General application performance requirements indicated by business needs.
- On-line response time requirements for application queries, updates ad processes.
- Minimum number of data access paths.
- Minimum amount of storage.

Figure 2. Criteria for Denormalization

The criteria for denormalization should address both logical and physical issues. After an exhaustive review of journal articles and experts' recommendations in professional journals, we have identified four criteria (Figure 2) that have been used as the rationale for turning to denormalization. These criteria are focused on reducing database access costs, and are affected by database activity, computer system characteristics and physical factors [19]. They also take into account that the dynamics of applications requires that a database design should be reviewed according to maintenance and development plans for the application system.

A primary goal of denormalization relates to how it improves the effectiveness and efficiency of the logical model implementation and how it fulfills application requirements. This requires an analysis of the advantages and disadvantages of possible model implementation. As in any denormalization process, it may not be possible to accomplish a full denormalization that meets all the criteria specified. In such cases, the database designer should exploit knowledge about the application requirements in order to evaluate the degree of importance of each criterion in conflict.

Finally, in Figure 3, in order to contribute to database designers' seamless knowledge in addition to criteria previously specified, we identified a list of variables from literature [10] [3] that should be taken into account when considering an absolute denormalization.

Denormalization has many drawbacks, such as data duplication, more complex data-integrity rules, update anomalies, and increased difficulty in expressing the type of access [16], [20]. The denormalization process can easily lead to data duplication and that in turn leads to update anomaly issues requiring increased database storage requirements.

An update anomaly problem can generally be resolved with database management techniques such as triggers, application logic, and batch reconciliation [9]. A trigger can update duplicated or derived data anytime the base data changes. Triggers provide the best solution from an integrity point of view, although they can be costly in terms of performance. In addition, application logic can be included into transactions in each application in order to update denormalized data to ensure that changes are

atomic. Finally, a batch reconciliation process can be run at appropriate intervals to bring the denormalized data back into agreement. Using application logic to manage denormalized data is risky, because the same logic must be used and maintained in all applications that modify the data.

Denormalization usually speeds up retrieval, but it can slow the data modification processes. It is noteworthy that both on-line and batch system performance is adversely affected by a high degree of normalization [15]. The golden rule is: When in doubt, don't denormalize. The next section of this paper will provide a strategy for increasing performance, but at the same time, minimizing the deleterious effects related to denormalization.

- Application performance criteria.
- Future application development and maintenance considerations.
- Volatility of application requirements.
- Relations between transactions and relations of entities involved.
- Transaction type (update/query, OLTP/OLAP).
- Transaction frequency.
- Access paths needed by each transaction.
- Number of rows accessed by each transaction.
- Number of pages/blocks accessed by each transaction.
- Cardinality of each relation.

Figure 3. Other Considerations of Denormalization

4. Denormalizing for performance

In this Section, we discuss denormalization patterns that have been commonly adopted by experienced database designers. After an exhaustive literature review, we identified and classified four prevalent strategies for denormalization in Figure 4. They are collapsing tables, splitting a table, adding redundant columns and adding derived columns.

- Collapsing Tables.
  - Two entities with a One-to-One relationship.
  - Two entities with a Many-to-Many relationship.
- Splitting Tables (Horizontal/Vertical Splitting).
- Adding Redundant Columns (Reference Data).
- Derived Attributes (Summary, Total, and Balance).

Figure 4. Denormalization Strategies.

4.1. Collapsing Tables

One of the most common and secure denormalization techniques is the collapsing of One-to-One relationships. This situation occurs when for each row of entity A, there is only one related row in entity B. While the key attributes for the entities may or may not be the same, their equal participation in a relationship indicates that they can be treated as a single unit. For example, if users frequently need to see COL1, COL2, and COL3 at the same time and the data from the two tables is in a One-to-One relationship, the solution is to collapse the two tables into one (See Figure 5).

There are several nice advantages of this technique in the form of reduced number of foreign keys on tables, reduced number of indexes (since most indexes are created based on primary/foreign keys), reduced storage space, and reduced amount of time for data modification. Moreover, combining the attributes does not change the business view but does decrease access time by having fewer physical objects and reducing overhead. In general, collapsing tables in One-to-One relationship has fewer drawbacks than others.
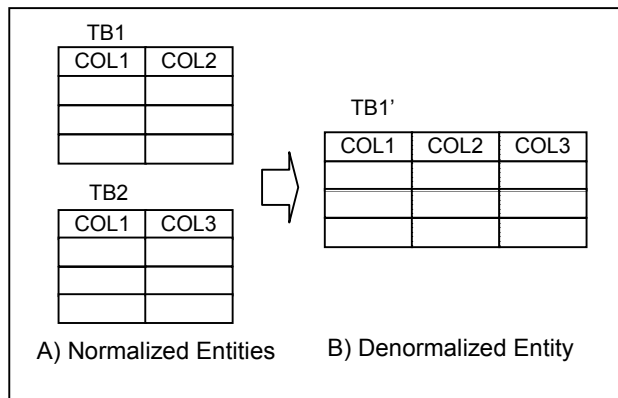

Figure 5. Collapsing Tables.

A Many-to-Many relationship can also be a candidate for the table collapsing. The typical Many-to-Many relationship is represented in the physical database structure by three tables: one table for each of two primary entities and another table for cross-referencing them. A cross-reference or intersection between the two entities in many instances also represents a business entity. These three tables can be merged into two if one of the entities has little data apart form its primary key (i.e. there are not many functional dependencies with the primary key). Such an entity could be merged into the cross-reference table by duplicating the attribute data. There is of course a drawback to this approach. Because data redundancy may interfere with updates, update anomalies may occur when the merged entity has instances that do not have any entries in the cross-

reference table. Collapsing the tables in both One-to-One and One-to-Many eliminates the join, but the net result is that there is a significant loss at the abstract level because there is no conceptual separation of the data. In general, collapsing tables in Many-to-Many relationship has a significant number of problems compared to other denormalization approaches.

4.2. Splitting Tables

When separate parts of a table are used by different applications, the table may be split into a denormalized table for each application processing group. In this case, the table can be split either vertically or horizontally. However, it should be noted that there are cases that the whole table should be able to be used for a single transaction.

A vertical split involves splitting a table by columns so that a group of columns is placed into the new table and the remaining columns are placed in another new table (Figure 6). Vertical splitting can be used when some columns are rarely accessed rather than other columns or when the table has wide rows. The net result of splitting a table is that it may reduce the number of pages/blocks that need to be read because of the shorter row length in the main table. With more rows per page, I/O is decreased when large numbers of rows are accessed in physical sequence. A vertically split table should contain one row per primary key in the split tables as this facilitates data retrieval across tables. In actuality, a view of the joined tables may make this split transparent to the users. This technique has been particularly effective when there are lengthy text fields in a long row. If the text fields are rarely accessed, they may be placed in a separate table.

A horizontal split involves a row split, resulting rows classified into groups by key ranges (Figure 6). This is similar to partitioning a table, except that each table has a different name. A horizontal split can be used when a table is large, and reducing its size reduces the number of index pages read in a query. B-tree indexes are generally very flat, and large numbers of rows can be added to a table with small index keys before the B-tree requires more levels. However, an excessive number of index levels may be an issue with tables that have very large keys. Thus, as long as the index keys are short, and the indexes are used for queries on the table rather than on whole table scan, a horizontal split prevents doubling or tripling the number of rows in the table. The result is that the number of disk reads required for a query by only one index level is reduced.

A horizontal split is usually applied when the table split corresponds to a natural separation of the rows such

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

fastcase®
Smarter legal research.