Paper 17
Course A

A Dual Database System with Real-Time Replication For the Higher Education Industry

Anthony LaMarr Watkins, Cornell University

May 11, 2001

Nice proposal! I find the argument very convincing. You build a solid business case and support it with a few dramatic performance figures.

What prevents this from being a really exceptional paper is that you don't push to the next level. I'm unclear about costs of ownership + administration mechanisms. It wasn't obvious how you might deal with real-time loops with very short response limits, like paging the bursar and then immediately trying to register. And I was left wondering how a dual design impacts database schema + design + query planning.

Still, if I was in this business, I would certainly take such a proposal seriously. You make it clear that you know your topic + have a high-value proposition for the customer!

1 Introduction

More than a decade after the advent of client/server architectures, the higher education market has yet to see commensurate advances in functionality. To the contrary, many of their attempts at client/server systems have failed, in whole or in part. Some well-regarded consultants claim that the new client/server applications are as costly as the legacy-based systems.

As they reflect critically on the question of "why", one possibility is that the fundamental design remains monolithic. The platform has changed and the terms are different, but the underlying structure is an all-encompassing, potentially scope-creep information system optimized for transaction processing, with incremental (cost ineffective) enhancements to support executive information and decision-support.

What is needed is a dual-database strategy, with one system exclusively designed for data entry/updating and another system optimized solely for information retrieval. In order for this type of system to gain acceptance, it must provide real-time guarantees. The key to this will be efficient replication from the data-entry system to the information retrieval system. This replication will be the focus of my project. More specifically, I would like to set up the skeletal framework for the back-end Student Data System (SDS) to the front-end Student Information System (SIS) and write an administrative application that allows for the replication to occur. This application will also allow the replication to be automated on timed intervals. The final product of this project will be a backend system that allows for data-entry, a front-end system that is optimized for information retrieval, and a module that allows and automates replication from the data system to the information system.

2 Background

The higher education industry has recently exhibited a trend towards updating their legacy systems. Many universities turned to PeopleSoft to implement these systems. Joseph Nolan, the vice president for human resources development and labor relations at Cleveland State University, said "At the time Cleveland State selected PeopleSoft, the university had no other choice because it wanted to work with modern business applications that could share human-resources, financial, and student information. Only PeopleSoft was selling such a combination."[6] These modern applications shifted technology from the traditional mainframe system to the newer client/server architecture. Despite the shift to a more modern system and 2.6 million dollars in hardware, these applications were experiencing serious difficulties. Provosts and vice presidents of seven of eight universities in the Big Ten Conference that use PeopleSoft programs sent a joint letter to PeopleSoft stating, "the performance of the systems, in terms of responsiveness, is simply unacceptable."[5] The letter focused on the amount of

Cornell reached a similar conclusion with a big donation from Dubfield helping them decide!

time it took to complete batch tasks, such as tuition calculations. These tasks took as much as six days for some institutions.

2.1 Fitting the Solution to the Problem

"Too often we push the problem into the background because we are in a hurry to proceed to a solution...[T]his tendency to focus on the solution has been harmful both to individual development projects and to the evolution of methods. Many projects have failed because their requirements were inadequately explored and described."[8] This quote from Michael Jackson outlines one of the most significant problems the higher education market faces. Many companies are building complex monolithic systems that are not tailored to the needs of today's colleges and universities. Paul N. Courant, associate provost at Michigan said, "I do not think there was good mutual understanding between the universities and PeopleSoft about what was going to be required for the software to work smoothly. To a certain extent, it's frustrating to educate them."[5] He further notes that universities are different from large commercial enterprises. "They are far less centralized and may have far-flung campuses whose research and graduate programs are distinct from undergraduate schools." It is clear that a fully integrated/monolithic system may not be the best solution for the problems that these highly decentralized universities encounter.

2.2 Retrieval vs. Updating

A classic topic in systems is proper design of the database schema. There are two key factors that impact the design and performance of a database. These are the normal form and field indices.[1] An optimized transactional system is normalized with few indices, while an optimized querying system is denormalized with many indices.

Updating information is best served with a normalized design. Among other things, a normalized design contains no repeating groups of data. Repeating groups occur in a table when the same information is included in each new row. This redundancy of data waste space and complicates the task of updating. Instead of updating data in one place, as is done in normalized tables, multiples copies must be maintained in a denormalized setting.[7] This has two main drawbacks. First and foremost, if all copies are not brought up to date, there will be inconsistencies in the database. Secondly, updating data in multiple places, instead of a single location, will increase the amount of time required to change information in the database, thereby degrading overall performance. It is clear that updating is best accomplished via a normalized design.

Information retrieval is done most efficiently with a denormalized design. A denormalized database spreads the same information across multiple tables.

[1] Familiarity with these two concepts is assumed.

This reduces the number of table joins necessary during a query. This is very significant as table joins are one of the most expensive operations that are performed during information retrieval.

3 Related Work

This is not the first time this problem has recognized and there have been proposed solutions. Ron Soukup, author of Inside Microsoft SQL Server 6.5, says "If the most important and time-critical function your system must perform is fast querying, it often makes sense to consciously back off from a normalized design. Think of normalization as typically being good for updating but potentially bad for querying. In fact, anytime I get beyond a four-way join, I look for alternatives."[1] While Soukup brings light to the problem, he does not provide a general solution and suggests temporary fixes in extremely bad situations. The work presented in this paper is focused at a system-wide solution to the general problem, not just special cases.

This work proposes a dual-database strategy with real-time replication. However, this idea has been visited. Tom Hammergren, author of Data Warehousing - Building the Corporate Knowledgebase, says

> Current technology doesn't allow for update and read activities to effectively occur simultaneously on one database. Thus, a dual-database strategy provides better performance, with one database for operational update activities and one for decision-support read activities. By creating two separate databases, each database can be individually tuned for optimal performance. The number of indexes on an update-intensive transaction database can be minimized, while the number of indexes on a read-only intensive decision-support database can be maximized. Each of these designs provides its respective constituency with maximum performance for a given environment.[2]

While Ham1mergren's work both realizes the problem and suggests a similar dual-database strategy, it does not deal with issues of real-time guarantees in the knowledgebase. It is this remaining step that will be the focus of this work.

4 Discussion

Before the remainder of this paper is presented, it is necessary to discuss a set of key issues. These topics are vital to the understanding the overall goals of this work.

4.1 Real-Time Systems

The largest, most common concern of people when discussing a dual-database strategy is whether the system is real-time. Unfortunately, this term has developed into a buzzword (a heavily sited, yet often misunderstood concept). Many people have the misconception that a real-time system implies that as soon as a change is made in any part of the system, that change is immediately recognized in all other parts of the system. However, this is not the case. E. Douglas Jensen of www.real-time.org writes in his article, Real-Time for the Real World, "Real-time computing is properly defined to have the objective of attaining results with acceptable optimality and predictability of timeliness. Systems operate in real-time to the degree that they achieve that objective." [3]

For instance, most people consider the debit card system to be real-time[2]. It is certainly real-time in the sense that the transaction will check the account balance, check flags for erratic spending that may indicate an unauthorized person is using the card, and confirm/refute the purchase all in a matter of seconds. However, the actual charge may not be deducted from the balance of the account for a period of 2-3 days. Yet, this aspect of the system is still considered real-time. This is due to it satisfying the two constraints listed above. First, it attains its results with acceptable optimality. In most cases, waiting 2-3 days for the charge to be deducted will not negatively affect a person's spending habits. In contrast if the period for recognition of the charge was 30-60 days, it could easily have the affect of people constantly overcharging on their accounts as they might not remember charges they had made a month or two earlier. This situation would not exhibit acceptable optimality. Secondly, it displays predictability of timeliness. The overwhelming majority of debit card transactions are recognized in three days or less. This range is an acceptable scope of time for the application in question. If the range was 30-60 days this would most likely not be considered predictable in time. However, it should be noted that if the range were exactly 30-31 days this would satisfy the predictability of timeliness restriction, but would most likely fail the acceptable optimality constraint.

The key point is that real-time systems are not defined by immediate system-wide recognition of events, rather they are defined by the degree of time it takes an event to propagate through the system and whether that is acceptable for the situation at hand.

4.2 Inquiries vs. Queries

In order to ensure real-time guarantees throughout the system, it is the case that some information be recognized immediately after it is placed in the system. Therefore, it is necessary to employ some degree of information retrieval from

[2]Debit cards are used exactly like credit cards, but the charge is withdrawn directly from the cardholder's bank account.

the Student Data System. This information retrieval is defined by this work as an inquiry. An inquiry differs from a query in that the information is resident in one table and is being searched primarily on an indexed field. Its purpose is to provide the most up-to-date information when necessary without straining system resources in the process. It usually corresponds to retrieving information on data that will be updated soon thereafter. On the other hand, a query can reference many tables/fields, deals with cumulative data, and often references historic information. A query can be quite resource intensive, especially if it is executed on a database that is not designed for that purpose. The remainder of this paper will focus on queries.

5 Overview

The system presented by this paper is based on a dual database strategy with real-time replication. The Student Data System (SDS) is the backend database that will handle transaction processing. This system will follow a normalized design with few indices. The Student Information System (SIS) is the frontend database that will handle information retrieval and decision support. The SIS will be based upon a denormalized design with heavy indexing on fields. The replication between the systems occurs from the SDS (source) to the SIS (destination).

As mentioned in the previous sections, the choice of normalization and indices has a tremendous impact on how the system responds to different tasks. A sample application was built in order to test the speed of querying a normalized versus denormalized table. This application is depicted below:

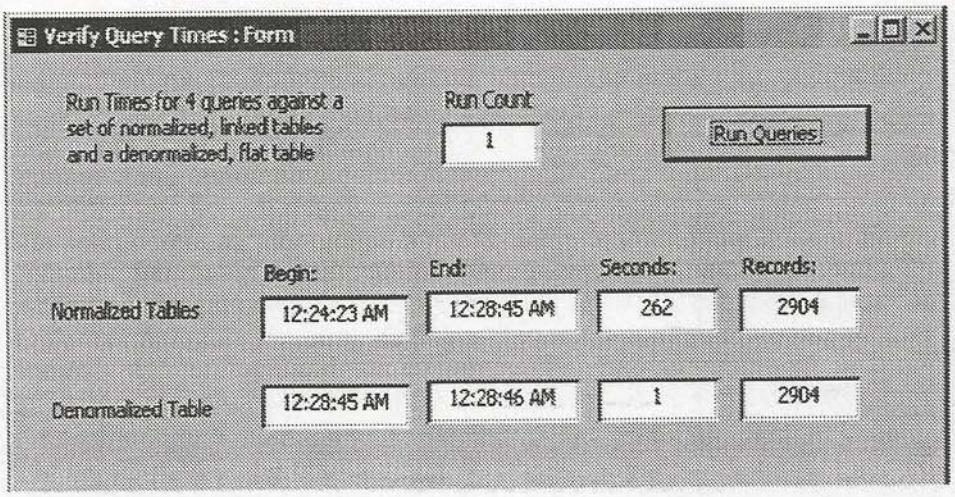

Figure 1: Query Tool

Below is a table containing results from an experimental query run against a normalized table with indices on the primary keys and a denormalized table that is heavily indexed. The times are based on four successive queries. The first query is student's 1) with management science majors, 2) who have applied for graduation, 3) that were admitted as first-time students, 4) from Maryland, and 5) who have earned more than 100 hours. The second query finds all students from the school of Arts and Sciences. The third query finds all currently enrolled transfer students who transferred from a college located in Maryland. Finally, the fourth query finds all currently enrolled students who made Dean's List their previous semester in attendance. These queries were run ten times to obtain results on an average run.

	Average Time
Normalized, indexed on primary key	3 minutes 56 seconds
Denormalized, heavily indexed	.7 seconds

Table 1 - Average Time of 4 Successive Queries

It is clear that a denormalized table provides much better performance in regards to information retrieval. However, as it was noted earlier, updating a denormalized table is not an acceptable strategy. This situation leads to the real-time replication mechanism presented in this paper.

6 Detailed Design

The real-time replication mechanism presented in this paper is implemented between two SQL Server Databases connected by a local area network. The actual replication mechanism is implemented with Microsoft's Data Transformation Services (DTS). DTS allows an application to define a mapping between a source table and a destination table. This mapping can be used in conjunction with a query to combine elements of tables or do complex calculations of data resident in the tables.

The source table in this system is the backend Student Data System (SDS), while the destination table is the front end Student Information System (SIS). In the test application, several of the normalized tables in the SDS are mapped into one denormalized table in the SIS. A figure capturing this mapping is noted below:

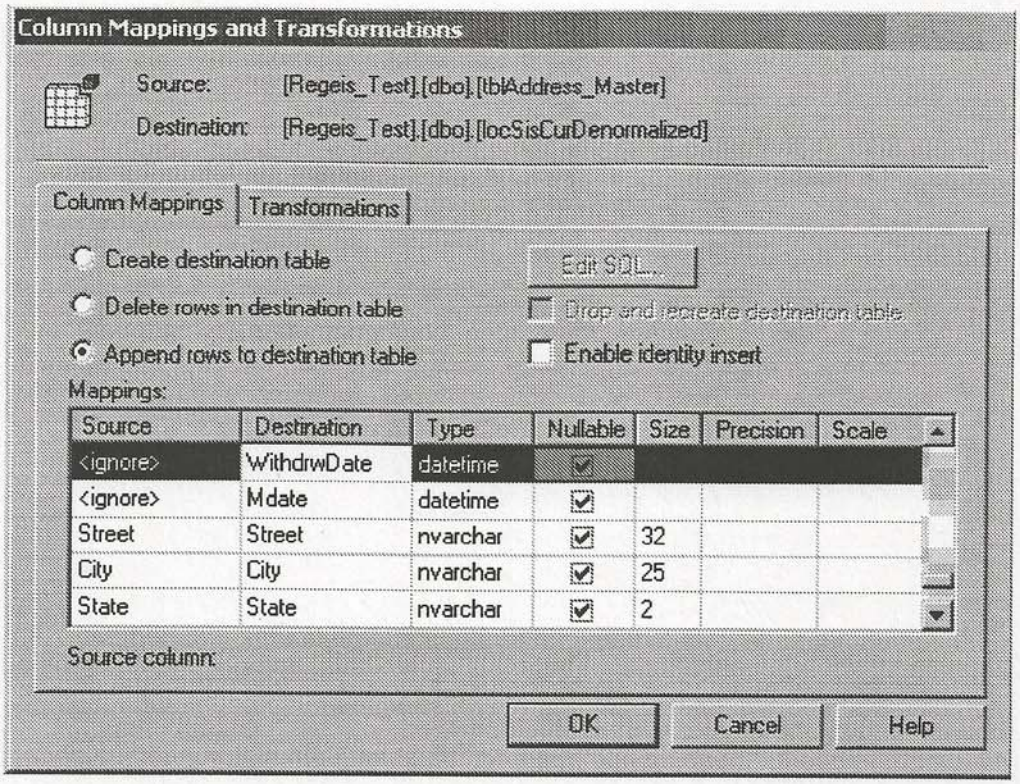

Figure 2: Mapping Tool

In this instance of the transformation, there is a mapping from the address table in the RegeisTest database on the SDS server to the denormalized table in the RegeisTest database on the SIS server. As seen in the example two of the columns, WithdrawDate and Mdate, are ignored. This is due to these columns not being resident in the address table of the source database, therefore they are ignored in the mapping. The columns street, city, and state are all resident in the address table and are therefore replicated to the destination table. This type of mapping was done for all normalized tables in the SDS to the denormalized table in the SIS.

7 Results

The transformations were run ten times to get the average speed of the replication process. The test was run each time with the same number of records transferred. It should be noted that it is highly improbable for even the majority of records to change in between a given replication window. However, as

this research does not have that information, the transfer time per record will be calculated. This information is contained below.

	Average Time	Record Count	Records/Second
Replication Process	4 min, 11 secs	3711	14.8

Table 2 - Replication Process Data

The process took an average time of 4 minutes and 11 seconds, slightly longer than it took to do the one query example presented earlier (avg. time = 3 min, 56 secs). This is a good performance time, especially considering that in practice it is highly unlikely for even a majority of the data to have changed between a replication window[3]. The record count is actually not the number of records transferred. It is the number of records resident in the denormalized table. Since the normalized table has much fewer columns than the denormalized table, many more records are actually transferred. The records/sec statistic given above should be interpreted as the number of rows in the denormalized table that can be populated per second. One of the key issues with this replication scheme is availability. While data is being transferred, both the SDS and the SIS can be accessed. While the SDS is not significantly affected during the process, the SIS has very noticeable delays when attempting to query on tables that are currently being updated. The querying will succeed, but it will take a longer amount of time. This is also slightly database dependent. By default SQLServer uses page-level locking, which are 2KB in size. If all or part of the information being accessed is within this 2KB segment, the query must wait until the lock is released.

8 Future Work

There are some key issues not explored by this research that should be accounted for before this system is placed into production. Foremost of these are fault tolerance mechanisms. This research did not attempt to set up mechanisms to detect failure or report failure when it occurs. In a production system, it is vital that errors in the replication process by trapped and reported. If this is not done, part or all of the replication may fail and as a result the SIS will contain stale data. In this respect, the system would not be providing the real-time guarantees mentioned earlier. Another unexplored facet is the statistics on how many changes are actually encountered during daily operation in a common college setting. Collecting more data on typical changes in data to estimate how long it would take on average for different size schools to run these transformations would be valuable information. With that, it would be easier to determine how many updates must be performed to the SIS during a

[3]Of course this depends on how often the replication process is run, but this research assumes that it will occur at least once a day.

daily basis. Having this information, it could be estimated the degree to which the SIS would have decreased availability and if that was acceptable. However, this estimation would require a large amount of statistical data on a variety of school types (public, private, HBCUs, liberal arts colleges, etc.) and sizes, which is why this topic was unexplored by this research.

9 Conclusion

This research presented a dual database strategy with a real-time replication mechanism. It showed that one monolithic system cannot efficiently support the behavior exhibited by today's colleges and universities. While detailed data was not collected, it is important to note that the proposed system in this paper is a solution tailored to the characteristic problems facing today's colleges.

The real-time replication mechanism was proposed, because colleges are highly data stable. There is only a small portion of time throughout the year when changes in information are widespread. In a semester, there is basically one week of registering and one week after course grades come in were many changes are being made. Information such as SATs, permanent addresses, course grades from previous semesters (and so on) do not change with great frequency. In addition to this, all universities have a freeze date approximately 14 days after classes start, signifying that no changes can be made for reporting purposes[4]. It does not make sense to have one monolithic system tailored to data-entry to serve such an environment. Users should not have to wait four minutes for a simply query to return or six days to do tuition calculations. A dual database strategy with real-time replication may not work for all environments, but for colleges and universities - it is a solution fit to the problem.

[4]Reports to outside agencies like the government for the PELL grant. Universities must use the same information today that they do next year.

10 References

1. Ron Soukup. Inside Microsoft SQL Server 6.5. Microsoft Press, 1999.

2. Tom Hammergren. Data Warehousing - Building the Corporate Knowledgebase. Sybase Press, 1999.

3. E. Douglas Jensen. Real-Time for the Real World. www.real-time.org/noframes/rt-concepts.htm, 2001.

4. Client/Server Software Architectures-An Overview http://www.sei.cmu.edu /str/descriptions/clientserver-body.html, 2000.

5. Wendy R. Leibowitz. Officials of 7 large universities complain to PeopleSoft about its programs. Chronicles of Higher Education, January 7,2000.

6. Florence Olsen. As PeopleSoft Problems Persist, Cleveland State U. Looks for a New Project Manager. Chronicles of Higher Education, January 21,2000.

7. Silberschatz, Korth, Sudarshan. Database System Concepts. McGraw-Hill, 1999.

8. Michael Jackson. Software Requirements & Specifications. Addison-Wesley, 1995.