

IMPROVE DATABASE PERFORMANCE AND MAINTAIN
REFERENTIAL INTEGRITY FOR A LARGE CLIENT
SERVER APPLICATION SYSTEM

By

HUI XU

Bachelor of Science
Petroleum University
Shandong, China
1982

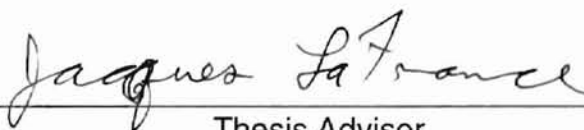
Master of Science
Research Institute of Petroleum Exploration
And Development
Beijing, China
1985

Doctor of Philosophy
Louisiana State University
Baton Rouge, Louisiana
1990

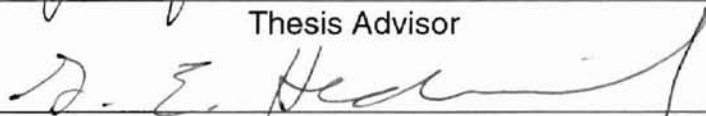
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1999

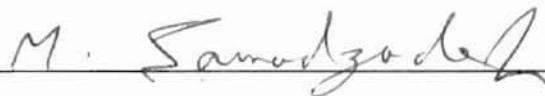
IMPROVE DATABASE PERFORMANCE AND MAINTAIN
REFERENTIAL INTEGRITY FOR A LARGE CLIENT
SERVER APPLICATION SYSTEM

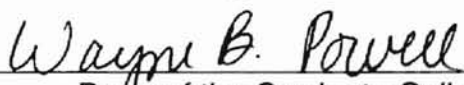
Thesis Approved:



Thesis Advisor







Dean of the Graduate College

ACKNOWLEDGMENTS

I sincerely appreciate my major advisor Dr. Jacques LaFrance for his intelligent supervision, constructive guidance, and encouragement of my thesis work. My sincere appreciation extends to Drs. George E. Hedrick and H. K. Dai, my committee members, for their valuable suggestions and comments.

This thesis was based on a project I completed when I worked at Transok, Inc. of Tulsa, OK. I particularly like to thank my supervisor at Transok, Roni Taylor, manager of Application Services at Information Services Department of Transok. My appreciation is extended to Judy Edmonson and H. Tim Reid of Transok, Inc. and David Covich of DC System for their useful inputs.

TABLE OF CONTENTS

Chapter.....	Page
I. INTRODUCTION	1
Relational Database Performance Problems	1
Referential Integrity vs. Performance.....	3
Purpose of the Study	4
Outlook of Work	4
II. REFERENCE REVIEWS	6
Relational Database Model.....	6
Foreign Keys and Referential Integrity.....	7
Foreign Key Effects for Database Performance	9
Database Normalization	11
Database Denormalization	12
Database Triggers	14
Referential Integrity for Large Database	14
III. COMMERCIAL DATABASES AND DATABASE TRIGGERS.....	16
Sybase Database	16
Oracle Database.....	17
Database Triggers	20
IV. IMPROVE PERFORMANCE WITH REFERENTIAL INTEGRITY	21
Foreign Key Referential Integrity Maintaining Processes	21
Client Server Application System.....	23
Maintain Referential Integrity during Data Deletion.....	24
Maintain Referential Integrity during Data Insertion and Updating.....	25
Maintain Referential Integrity for Denormalized Database.....	27
V. APPLICATION EXAMPLE OF A LARGE CLIENT SERVER SYSTEM	29
Basic Information about the Example System	29
Triggers For Maintaining Referential Integrity during Data Deletion	33
Maintaining Referential Integrity during Data Insertion and Updating	39
Triggers for Denormalized Database	40
VI. SUMMARY AND CONCLUSIONS	42
REFERENCES	43
APPENDIXES.....	46

LIST OF TABLES

Table	Page
1. List of the tables and columns names using contract_nbr as foreign keys	30
2. Child tables information used for designing "contract" table delete trigger	34

CHAPTER I

INTRODUCTION

Relational Database Performance Problems

Relational database performance is affected by many factors when the database is very large [Anderson, 1997, Paulsel, 1996, Ramakrishnan, 1997, and Roy and Sugiyama, 1996]. The factors include computer hardware, computer system setup, database design, database query procedure designs, database referential integrity checks, etc.

Computer hardware is one of the most important factors for the database performance. However, the computer hardware upgrade is limited due to the current hardware technique and also the project budget. The hardware upgrade is one of the options for the database performance improvement, but not the total solution.

Computer system setup is another important factor for the database performance. Since a query's execution time is mostly spent on reading data pages from disk, the system administrator has several guidelines to avoid the bottleneck and improve the database performance. By spreading heavily used tables or frequently joined tables on separate disks, the query processes will reduce the numbers of disk input/output into a disk. The improvement from the system setup

will be limited when queries need to read too many data pages and database tables at one time.

Database design is not only important for a good and complete physical modeling of the business, but also important for the database performance. The database design needs to balance several factors to get good database performance. For example, if the database involves data manipulation like data insertion, updating, and retrieving, the number of table indexes needs to be limited. The table indexes will be very helpful when retrieving data, but will be very costly when inserting or updating data.

The effect of the database query procedures could become very important when the database is large. When the query needs to do table scans for large database tables, the process will be very slow. There are several procedures that can be used to optimize the query. The procedures include setting table index, creating temporary tables, reducing table joins, etc.

The database performance will also be affected by the database referential integrity check (foreign key reference). The database referential integrity check is the procedures to ensure the data saved in the database to be correct. The referential integrity check procedures will be involved when data are modified in one or more tables related by the foreign key definitions. The performance effect due to the referential integrity checks is the main research topic of this thesis.

In real world applications, hundreds of database tables could be involved in the referential integrity checks when data are modified in one of the database tables. These tables are usually related each other as the relationships of parent

and child tables defined by the foreign keys. In the other word, when one of the parent table have hundreds of child tables, any data changes in the parent table require referential checks in the parent table or any data changes in any of the child tables require referential checks at the parent table. If large amount of data need to be inputted into the database system at almost the same times, the database performance affected by the referential checks could be the bottleneck of the system. Another minor performance problem for this kind of database system is that many extra table joins are required when querying data from the child tables of the database for daily production reports. This thesis is to provide an easy and fast solution for these two performance problems.

Referential Integrity vs. Performance

Several energy companies engaged in natural gas transmission business are using the database models with some of the parent tables referred by too many child tables. Due to the slow performance when performing the referential integrity checks (foreign key references), these companies simply ignored the referential integrity checks to trade for the database performance. Ignoring the referential integrity checks is a very dangerous action for their data. When the database referential integrity has some problems, wrong data could be stored in the database and many valuable data could also be vanished in the database.

When balancing between the database performance and any other problems (e.g. referential integrity problem), we are more favorable on the database performance. However, the referential integrity problem is also very important for

our applications. We need to find out an alternative way to keep the referential integrity checks without affecting too much on the database performance.

Purpose of the Study

The purpose of this thesis is to study the database performance problems during the referential checks or data querying for a database system with some parent tables referred by too many child tables and to provide a practical solution for these problems. The results are illustrated by an application example.

Outline of Work

This thesis work includes six chapters. Chapter I introduces the main problems for this thesis, purpose of this study, and the outlook of the thesis work. Chapter II is concentrated on reference reviews including some fundamental database concepts, databases triggers, referential integrity, database denormalization, and database performance problems in large relational database. Chapter III reviews the features of some commercial relational database models and their triggers. Chapter IV analyzes the procedures of maintaining the referential integrity performed by foreign keys and discusses an alternative way to maintain the database referential integrity and to improve the database performance. This chapter also studied the database performance improvement by database denormalization to reduce table joins and the method to maintain referential integrity for the denormalized database. Chapter V provides an application example of a large client server application system to illustrate the implementations of the

database performance improvements. Chapter VI summarizes the results of this thesis study and concludes this thesis.

CHAPTER II

REFERENCE REVIEWS

Relational Database Model

The relational database model and some of the concepts are reviewed in the thesis. The references on the database model and concepts in this chapter are from the following books and articles in authors' alphabetical order: Brathwaite, 1991; Codd, 1971, 1972, 1974, 1979, 1983, 1988; Date 1981, 1990, 1995; Fagin, 1977, 1979; Koch & Loney, 1997; Logic Works, 1995; Paulsell, 1997; Silberschatz, Korth, & Sudarshan, 1997; and Ullman & Widom, 1997.

Relational database system has been widely used in industry for storing, reviewing and updating data. In a relational database model, data are perceived by the user as tables. The entire information content of the relational database is represented in one and only one way. There is always exactly one data value at every row and column position in every table of the relation database.

Basically, the relational database model is concerned with three aspects of data: data structure, data integrity, and data manipulation. The database structure is defined by the tables in which the attributes are represented by columns and the records are represented by rows. The data integrity defines the rules the relational database must obey. For example, one of the rules defines that each row in a table must include a unique value. The data manipulation is the operations to extract,

insert, or update data from specified rows and columns in one or more tables. Structured Query Language (SQL) is used for the data manipulation of the relational database.

The uniqueness property of each table is defined by a primary key. The primary key can include one or more columns in a table. Primary key is required to be unique and irreducible and every table must have a primary key.

Foreign Keys and Referential Integrity

Foreign key is the non-primary key column(s), which is the primary key in other table. Foreign key links the tables like the parent-child relationship, in which the table of the primary key is the parent table and the table of non-primary key is the child table.

Referential integrity refers to the accuracy or correctness of the data in the database. The primary and foreign keys will enforce the referential integrity of the database. All the attributes in the primary key are implicitly declared to be not null and must be unique. The attributes of the foreign key can be null, provided that they have not otherwise been declared to be non-null. When non-null attributes of the foreign key are inserted or updated in the child table, the attributes of the child table will be checked for the existence in the parent table. When the attributes in the parent table are updated or deleted, the child tables defined by the foreign key will be checked. Unlike the database rules that simply reject any operations that would result in illegal state, foreign keys can be assigned with certain operations to enforce the referential integrity of the database. These operations include

'RESTRICT' and 'CASCADE' during the delete and update operations. The delete RESTRICT rule will restrict the delete operation when the deleting data exist in the foreign key referred tables. The delete CASCADE rule will cascade the delete operation to all the referred tables by the foreign key. The update RESTRICT rule will restrict the update operation when the updating data exist in the foreign key referred tables. The update CASCADE rule will cascade the update operation to all the referred tables by the foreign key. The following is an example of creating the foreign keys:

```
create table CUSTOMER (  
    customer_id      CHAR(10),  
    address_purpose    CHAR(5),  
    contact          CHAR(40),  
    status           CHAR(1),  
    beg_eff_date     DATE,  
    PRIMARY KEY (customer),  
    FOREIGN KEY (customer_id, address_purpose) REFERENCE  
        ADDRESS(customer_id, address_purpose) )
```

Integration rules can also be built into the database to enforce the data integrity. When a new integration rule is created, the database system should first check to see whether the current state of the database satisfies the specified constraint. If it does not, the new rule should be rejected; otherwise it will be accepted.

The following is an example of creating rules in the database.

```
create rule pubid_rule  
as @pub_id in ('1234', '5678', '3456')
```

Foreign Key Effects for Database Performance

The transactions defined by foreign keys will be performed automatically when data in parent tables or child tables are modified. The foreign key transactions are depended on the foreign key definitions and data transactions in parent tables or child tables.

When data in parent table are being updated, the foreign key transactions will be responded to the data updating depending on which part of the parent table is updated. If non-primary key parts of the parent table are updated, no data change in child tables is required. If primary key parts of the parent table are updated, the child table data need to be modified based on the foreign key definitions. If cascade update is defined by the foreign keys, all the child tables need to be updated with the new values. If restrict update is defined by the foreign keys, the update on the parent will be rolled back to the original values if the values of the parent table are used in the child table.

When data in parent table are inserted, child tables will not be affected and no actions are required for the child tables.

When deleting data in parent table with cascade deletion, the data of the corresponding child table will be deleted or set to be null if allowed in the child tables. When deleting data in parent table with restrict deletion, the deletion of the parent table data depends on the existence of the data in child tables. If the

corresponding values exist, the parent table data deletion will be stopped and the data transaction will be rolled back. Otherwise, the parent table data deletion will be committed.

When child table data in the foreign key defined columns are being updated, the corresponding values in parent table will be checked. If the corresponding values exist, the child table update will be committed. Otherwise, the data being updated are not valid data and the transaction of the child table update will be rolled back.

When new data are being inserted into the child tables, the corresponding values in parent table will be checked. If the corresponding values exist, the child table insertion will be committed. Otherwise, the child table insertion will be rolled back.

When data are being deleted from the child tables, referential integrity check is not required in the parent table.

The transactions of the referential integrity checks defined by the foreign keys require computing time. When a few child tables refer one parent table, these transactions can be completed very fast without performance problem. However, when too many child tables refer one parent table, the required times for the referential checks could be a problem in the application system. The database performance will be worse when many data changes are required at the same times. Several energy companies reported this kind of problems in their large client server application systems.

Database Normalization

When designing a relational database, we usually need to normalize the database. The goal of normalization is to ensure the uniqueness property. That is, there is only one way to know a "fact." The process of normalizing a model is one of removing all model structures that provide multiple ways to know the same fact. Another way to look at normalization is as a method of controlling and eliminating redundancy in data storage. There are three basic normalization definitions for the relational database. First normalization means that the rows and columns of the database data must form a two-dimensional table with no nested structure inside any cell. That is, every data value in the relational database must be atomic with no lists, repeating elements or internal structure. Informally, we can define that a table is in first normal form if each of its attributes has a single meaning and not more than one value for each instance.

Second normalization means that a "fact" can only be determined by knowing all the attributes of the primary key of the table. That is, a table violates second normal form, if a "fact" can be determined by knowing only part of the key of the table.

Third normalization means that the non-key attributes are mutually independent and irreducibly dependent on the primary key. That is, if a fact can be known by knowing the value of some non-key attribute of the table, then third normal form is violated. In other word, a table is in third normal form if every non-key attribute depends on the key, the whole key and nothing but the key.

There are several other normalization definitions, like Boyce/Codd normal form, Fourth normal form, Fifth normal form. In practice, third normal form is the standard.

However, a fully normalized design may not always yield the best performance. We may intentionally denormalize the database in order to gain performance. It is recommended that we design the database in third normal form, and then, as performance issues arise, denormalize the database to solve the performance problems. Denormalized database also has some disadvantages. It usually speeds retrieval but can slow data modification. The denormalization is usually application-specific and needs to be re-evaluated if the application changes. The size of database tables can also be increased. Data referential integrity problem can be raised when the database is denormalized.

Database Denormalization

At the level of the physical database design, choices are usually made to "denormalize" a structure in favor of performance for a certain set of transactions. This may introduce redundancy in the structure, but is often worth it. The advantages of the denormalization may include: minimizing the need for joins; reducing the number of foreign keys on tables; reducing the number of indexes, saving storage space and reducing data modification time; pre-computing aggregate values, that is, computing them at data modification time rather than at select time.

The denormalization techniques include adding redundant columns, adding derived columns, and collapsing tables.

Adding redundant columns can eliminate frequent joins since joins are very costly in performance. For example, in a payment table, buyer, seller, and payee may be involved. The buyer, seller, and payee names can be found from a business associate table using a unique ba_number. If we want to get detail information about the payment, we need to join the payment table with the business associate table three times to get all the names and may need to join with some other tables for other information. We may denormalize the payment table design by adding the columns of the buyer, seller, and payee names in the payment table to reduce the number of the table joins.

Adding derived columns is to add columns for aggregate values. It can help eliminate joins and reduce the time to produce aggregate values. For example, in payment and pay_detail tables, one payment transaction can involve several payments that are described in the pay_detail table. To get the total payment for a transaction, we need to join these two tables and calculate the sum of the payments. To eliminate the join and the calculation time when retrieve the sum, we can add a column in the payment table to save the total payment data for each transaction.

Collapsing tables is to combine two tables into one in order to eliminate the join. For example, the payment and pay_detail table can be combined into one table to eliminate the join.

The performance improvement from the denormalization will result in the data referential integrity problem. For example, when the business associate name is changed in the business associate table, the old name is still shown in the

payment table. To ensure the data referential integrity, we can use trigger or run batch processing to update the changes in payment table.

Database Triggers

Trigger is a special tool equipped in most of the commercial databases. Triggers are developed by SQL and some special trigger codes and can be used to do many kinds of data transactions. Triggers will be automatically fired when data in the tables attached with triggers are modified. Triggers can only be developed and owned by DBA type users. Every database table can be attached with a trigger for every kind of data transaction. That is, every table can have an insert trigger, an update trigger, and a delete trigger. Data manipulations on a table with an attached trigger will not be committed until the attached trigger is successfully completed. When the trigger execution is failed, the data transactions on the table will be rolled back. Database triggers are usually used for enforcing the busyness rules.

Referential Integrity for Large Database

Most of conference proceedings, magazines, and many reference books in relational database areas were reviewed in the areas of referential integrity for large databases. The conference proceedings [<http://sunsite.informatik.rwth-aachen.de/dblp/db/conf/index.a.html>] include ADBIS (Advances in Databases and Information Systems), ADBT (Advances in Data Base Theory), DOOD (Deductive and Object-Oriented Databases), EDBT (Extended Database Technology), ICDE (International Conference on Database Engineering), ICDT (International

Conference on Database Theory), ICOD (International Conference on Databases), IDEAS (International Database Engineering and Application Symposium), OOER (Object-Oriented and Entity-Relationship Modeling), PODS (Symposium on Principles of Database Systems), RIDS (Rules in Database Systems), SIGMOD (International Conference on Management of Data), SSD (Symposium on Large Spatial Databases), TODS (ACM Transactions on Database Systems), VLDB (International Conferences on Very Large Data Bases), etc. The magazines [<http://sunsite.informatik.rwth-aachen.de/dblp/db/journals/index.html>] include "The Database & Client/Server Solutions Magazine" [<http://www.dbmsmag.com>], "Database Programming & Design" [<http://www.dbpd.com>], "Journal of Database Management" [<http://www.idea-group.com/jdm.htm>], and some current research areas [<http://www-db.stanford.edu/~ullman/ullman-papers.html>]. These papers and books explored many research topics including the database referential integrity problems. However, none of them discussed the database performance problems in larger database when too many child tables refer a parent table. Of course, no solution was suggested for this problem.

CHAPTER III

COMMERCIAL DATABASES AND DATABASE TRIGGERS

Sybase Database

Sybase database [Cliffor, 1997, and Garbus, Solomon, Tretter, & Rankins, 1996] is one of the most popular relational databases used in the industry due to its reliability and high performance as a distributed database or client/server system. Sybase supports standard SQL and provides many useful build-in functions. Sybase is very famous for its simplicity and great performance. Sybase provides many system procedures and tables that are very useful for obtaining system information and supporting database management.

Sybase supports the three types of triggers, i.e. insert, update, and delete triggers. Sybase will create a table called "deleted" to keep the deleting data when deleting or updating from a table and a table called "inserted" to keep the inserting data when inserting data into a table. The "deleted" and "inserted" tables are structurally same as the table for which the trigger is defined. During the data updating transaction, the old values are deleted first and the new values are added later. Therefore, the data update transaction will first affect the "deleted" table, and then the "inserted" table. The following is an example of delete trigger for deleting data from a table called TITLEAUTHOR. In this example, cascade deletion is

defined for a TITLE table. When data are deleted from the TITLEAUTHOR table, the data from the TITLE table will also be deleted:

```
create trigger tD_titleauthor on TITLEAUTHOR
for delete as
begin
    delete TITLE
    from TITLE, deleted
    where deleted.title_id = TITLE.title_id
    delete TITLEAUTHOR
    from TITLEAUTHOR, deleted
    where deleted.title_id = TITLEAUTHOR.title_id
end
```

If there is a delete trigger attached to the TITLE table, the delete trigger will also be fired by the TITLEAUTHOR table data deleting transaction.

Oracle Database

Oracle database [Koch & Loney, 1997, Singh, Leigh, Zafian, et al, 1997, and Urman, 1997] is another most widely used relational database. Oracle runs on almost every kind of computer, from minicomputer, mainframe computer to PCs, Macintoshes. Oracle supports standard SQL and many other build-in functions. One of the very powerful tools Oracle supports is the SQLPLUS. SQLPLUS is a

kind of interactive report writer. SQL can be used to get information from the Oracle database and create polished reports with easy controls. SQL commands can also be executed directly inside the SQLPLUS. Other Oracle features include objected database, database security, database integrity, packages, stored procedures, functions, triggers, etc.

Oracle supports the following twelve types of triggers:

BEFORE INSERT row

BEFORE INSERT statement

AFTER INSERT row

AFTER INSERT statement

BEFORE UPDATE row

BEFORE UPDATE statement

AFTER UPDATE row

AFTER UPDATE statement

BEFORE DELETE row

BEFORE DELETE statement

AFTER DELETE row

AFTER DELETE statement

Here, row-level triggers execute once for each row in a transaction. Statement-level triggers execute once for each transaction. Within the trigger, "Old" and "New" refer to the old and new values involved in the transaction. Triggers for multiple insert, update, and delete commands on a table can be combined into a

single trigger, provided that they are all at the same level (row-level or statement-level). The following is an example of insert and update trigger for a table called LEDGER [Koch & Loney, 1997]. When insert or update a row in LEDGER table, the trigger will insert a row in a table called LEDGER_AUDIT table with “new” or “old” data:

```
create trigger ledger_bef_upd_ins_row
before insert or update of Amount on LEDGER
for each row
begin
    IF INSERTING THEN
        insert into LEDGER_AUDIT
        values (:new.Action_date, :new.Action, :new.Item, :new.Quantity,
                :new.QuantityType, :new.Rate, :new.Amount, :new.Person);
    ELSE
        insert into LEDGER_AUDIT
        values (:old.Action_date, :old.Action, :new.Item, :old.Quantity,
                :old.QuantityType, :old.Rate, :old.Amount, :old.Person);
    END IF
end
```

Data transactions created by the trigger of one table can ignite the trigger of other tables. That is, if there is an insert trigger for LEDGER_AUDIT table, this

insert trigger will also be automatically fired when inserting data into the LEDGER_AUDIT table.

Database Triggers

As we mentioned before, database trigger is a database procedure attached to a database table that can be automatically fired when the delete, update, or insert transactions in the table are completed or going to start. The trigger can be executed before or after an insert, delete, or update transaction in the table. The trigger action can be defined by the special trigger codes and many supported SQL statements.

When triggers are defined for many tables in the database, one trigger execution could start the executions of many other triggers. In this case, the triggers should be designed very carefully to avoid a trigger loop [Silberschatz, Korth, & Sudarshan, 1997]. That is, a trigger loop will be happened when table A delete trigger is cascaded on table B and table B delete trigger is restricted on table A. Deleting data from table A will trigger the data deleting from table B first. Deleting data from table B will be depended on whether the data exist in table A. In the other word, deleting data from table A depends on whether the data exist in table A. In this case, the data will never be deleted from either table A or Table B. The trigger loop could involve more than two tables.

Most of the commercial databases support the triggers. The databases include Sybase, Oracle, Microsoft SQL Server [Soloman, Woodbeck, Ramkins, Garbus, & McEwand, 1996, and Wynkoop, 1997], etc.

CHAPTER IV

IMPROVE PERFORMANCE WITH REFERENTIAL INTEGRITY

Foreign Key Referential Integrity Maintaining Processes

The purpose of this study is to improve the database performance and maintain the database referential integrity when many child tables defined by the foreign keys refer to a parent table. To improve the database performance, we need to find a more efficient way to perform the database referential integrity checks during the data manipulations. The data manipulations include the data updating, insertion, and deletion of both parent and child tables.

During the parent table data updating, the non-key part data updating will not affect the referential integrity. The key part data updating requires the checks for the existence in the child tables. The key part data updating is completed by deleting the old data and then inserting the new data. The parent table updating becomes the combinations of the parent table deletion and insertion. The parent table deletion and insertion will be discussed next. In real application systems, the key part data updating is not very often. Most of the key column(s) of the parent tables just contains a system generated number, which is physically meaningless. We may simply restrict user from updating the parent table key part data.

The parent table data insertion will not affect the database referential integration. No referential integrity is required for this case.

During the parent table data deletion, the deleting data will be checked for the existence in the child tables. The cascade and restriction deletions will be defined in the deleting process.

The referential integrity checks during the child table updating depend on whether the foreign key parts are updated. If the foreign key parts are updated, the new values must be existed in the parent table. If the new values are found in the parent table, the child table updating transaction is committed. Otherwise, the transaction will be rollback.

The child table data insertions also require the referential integrity checks. The child table foreign key parts must be existed in the parent table. The referential integrity process is completed with the help of the parent table index and is usually very fast. However, when one parent table is referred by hundreds of child tables and large amount of data will be inserted in these child tables, the database performance could be the bottleneck of the system.

The child table data deletions will not affect the database referential integrity. No database system check is required.

In summary, if we can perform the referential integrity checks for parent table deletion and child table insertion and updating, we can replace the functionality of the foreign keys. The next three sections of this chapter will analyze the client server system and provide alternative methods to perform these referential integrity checks in the client server system. The purpose of this thesis is to find a more efficient way to perform these referential integrity checks.

Client Server Application System

Client server application system [Anderson, 1997, and Date, 1995] is a very popular system currently used in industry. With the current Microsoft Window technology and many other network companies' network technologies, the application software developments become much faster and the development costs are much lower. The functionality of the client server system is more diversified and satisfies many application requirements. The client server system is replacing the old and expensive main frame system in many companies.

The client server system consists of two main parts, i.e. client system and server system. The client system consists of many users' PCs. The server system consists of several high performance workstation computers. Local Area Network (LAN) is used to connect the client system PCs with the server system workstation computers. Database systems and many business processing programs are loaded in the server workstation computers and the application interface programs are loaded in the users' PCs. Data manipulations inside the database are completed by the server workstation computers through the requests from the users' PCs. Many batch job programs are also stored in the server workstation computers and the batch jobs can be scheduled to run 24 hours a day. Currently, the most popular database systems include Oracle, Sybase, Microsoft SQL Server, Informix, etc.

The client application programs are usually the Microsoft window application programs. The application programs are usually functioned as the interface between the users and the database. The system users can retrieve, insert, update, or delete data in the database resided in the server workstation computers

from the application program interface. Some SQL tool programs installed in the clients' PCs could also manipulate the data in database using direct SQL codes. DBA version SQL tool programs can be used to manage the database in the server workstation computers. Currently, the popular programming tools used to develop the client application window interface programs are PowerBuilder, Visual Basic, Visual C++, etc. Several SQL tool programs from different vendors can be used as either a data manipulation tool or a database management or program development tool. Rapid SQL and DBA version Rapid SQL are the examples of the SQL tool programs.

Maintain Referential Integrity during Data Deletion

As we discussed before, the child table data deletion will not affect the database referential integrity. The parent table data deletion will affect the referential integrity. A deletion process defined by the foreign keys will perform either cascade or restrict deletions or combinations of the both for all the child tables. When without the foreign key definitions, a carefully designed delete trigger can perform the same functionality during the parent table data deletion. The delete trigger can perform the same process for all child tables with either cascade or restrict deletions or combinations of both. An example of the parent table delete trigger is illustrated in Chapter V.

When deleting data from the parent table referred by hundreds of child tables, the cascade deletions will delete lots of data from the child tables. Thus, the deletion process could be very slow. Most of the parent table data deletions will be

stopped by the restriction conditions and the data cannot be deleted from the database. In real applications, if the parent table data are referred by many child table data, these data are very valuable and deletions of these data are very rare. Most of the parent table deletions are happened when the data are just inserted and are found to be useless. The data deletion in this case is not very slow.

The parent tables usually contain some important business data. These data are not changed very often. Extra cautions may be required to protect these data. Very limited users who understand the business well should be granted with the rights to modify these data. This security caution in the system can reduce the chances of the data deletions and the performance impact caused by the data deletions.

Maintain Referential Integrity during Data Insertion and Updating

To maintain the referential integrity during child table data insertion and updating, we need to make sure that the foreign key part values of the child tables must be exist in the parent table. If the foreign key part values of the child tables are guaranteed to be correct, the foreign key referential integrity checks can be skipped. To enter correct values to the foreign key parts of the child tables, we can simply force the system users to select correct values when performing the child table data insertion and updating.

In our client server system, the client application programs provide the interfaces between the system users and the server database. The client application programs are developed using standard Microsoft windows. The system

users need to enter some data through the windows in the application programs to start the child table data insertion and updating process. In these windows, all the parent table key part data can be preloaded in a dropdown list box before the windows are opened. When the system users need to input or change the child table foreign key part data in the windows, they can only select correct values from the dropdown list box. The window interface will reject any manually typed data to avoid any human mistakes. Therefore, correct values will always be entered into the child tables and the referential integrity checking is not necessary anymore. In the other word, the data referential integrity is already checked before the data can be saved into the database.

The difference between this referential checking system and the delete trigger system is the data checking time. The delete trigger will check the data after the data transaction requests are submitted to the database. The above system will check the data before the data transaction requests are submitted to the database. The second case creates a chance for the system users to bypass the application window interface system and store wrong data in the child tables. The system users can use some SQL tool programs to insert or update data into the child tables using SQL codes. The data changes from the direct SQL codes cannot guarantee the data referential integrity in the child tables. This problem can be resolved by adding one more layer of security system. Two database passwords will be set for each system user. One of them will be given to the user. The other password is stored in a password table that is not visible to the user. The password known by the user can only be used to log in the application interface system. After logging

in the application interface program, the user's password will be converted into the other password, which can be used to change data in the database. This will force the user to change the database data from the system application interface programs.

In summary, the foreign key referential integrity checks for the database system can be replaced by a carefully designed system. This system includes the parent table delete triggers, the window interface designs of the client application programs, and a database password swapping system. The bottleneck of the database performance problems during the foreign key referential checks is happened when one parent table is referred by hundreds of child tables and large amount of data will be inserted into the child tables. With this improved system, the referential integrity check are completed before inserting data into the child tables and the database system will not waste any time on the referential integrity checks.

Maintain Referential Integrity for Denormalized Database

Another database minor performance problem is that too many table joins are required when querying data from the database when too many child tables refer to the parent tables. Table joins are very costly when the number of table joins is over certain limit.

Database denormalization technique can be used to reduce the number of the table joins. Denormalized database usually adds some redundant columns of the data from the parent tables to some child tables. When querying data from both the child and parent tables, the number of the table joins will be reduced. However, the redundant columns could create data integrity problem. When the data in the

parent tables are modified, the data in the redundant columns of the child tables need to be changed to reflect the correct values. Database trigger can be used to update the data in the child tables. The application example in next chapter illustrated the denormalized databases and the triggers to maintain the referential integrity for the redundant columns.

CHAPTER V

APPLICATION EXAMPLE OF A LARGE CLIENT SERVER SYSTEM

Basic Information about the Client Server Application System

The application example of the client server system used in this study is a Gas Information System (GIS) developed by an Oklahoma energy company engaged in the natural gas transportation business. GIS is used to record the real time natural gas volumes flowed into the company's pipeline from the natural gas producers and the natural gas volumes flowed to its customers. GIS also stores lots of other business critical information. GIS is a client/server system developed inside the company using PowerBuilder and Sybase. PowerBuilder is used to develop the client side application programs and Sybase is used as the relational database engine to manage the data on the server computers. GIS is a large application system with about 600 database tables and about 500 PC window program interfaces.

Four parent tables in the system are referred by large numbers of the child tables. The four parent tables are "contract", "busassoc" (business associate), "pipeline", and "station" tables. To enforce the database referential integrity, foreign keys need to be defined for the parent child relationship. For example, the contract number (contract_nbr) is the key column of "contract" table and the "contract_nbr" needs to be defined as the foreign keys of all its child tables. The "contract_nbr" is

referred 117 times by 95 child tables. Table 1 lists all the tables and the column names using contract_nbr as foreign keys.

Table 1 List of the tables and columns names using contract_nbr as foreign keys

	Table Name	Column Name
1	accrual_trans	contract_nbr
2	accrual_trans_detail	netting_k_nbr
3	acct_contract_info	contract_nbr
4	acctg_trans	contract_nbr
5	acctg_trans_detail	netting_k_nbr
6	alerts	contract_nbr
7	alerts_action	contract_nbr
8	allocated_volume	downstream_k_nbr
9	allocated_volume	upstream_k_nbr
10	amendment	contract_nbr
11	area_contract_ba	contract_nbr
12	bid	contract_nbr
13	bid	k_price_k_nbr
14	bid	trn_k_nbr
15	bid_received	contract_nbr
16	bid_received	k_price_k_nbr
17	bid_received	trn_k_nbr
18	billing	contract_nbr
19	contact_purpose_k	contract_nbr
20	contact_purpose_k_station	contract_nbr
21	contract	contract_nbr
22	contract_ba	alias_contract_nbr
23	contract_ba	contract_nbr
24	contract_groups	contract_nbr
25	contract_misc	contract_nbr
26	contract_qty_pur_sls	contract_nbr
27	contract_qty_trn_gth	contract_nbr
28	contract_rate_schd	contract_nbr
29	contract_rcpt_dlvry_pts	contract_nbr
30	contract_reservation	contract_nbr
31	contract_station_ba	contract_nbr
32	contract_station_ba_cat	contract_nbr
33	contract_station_ba_status	contract_nbr

34	contract_status	contract_nbr
35	crescent_pricing	contract_nbr
36	deals	contract_nbr
37	deals_order	contract_nbr
38	deals_order	downstream_k_nbr
39	deals_order	upstream_k_nbrcontract
40	deals_price_prov_adjs	netting_contract_nbr
41	deals_pricing	k_price_k_nbr
42	deals_trans	downstream_k_nbr
43	deals_trans	trn_knbr
44	deals_trans	upstream_k_nbr
45	discrepancy	contract_nbr
46	discrepancy	downstream_k_nbr
47	discrepancy	upstream_k_nbr
48	gathering_contract	contract_nbr
49	invoice_component	contract_nbr
50	k_area_dedication	contract_nbr
51	k_sta_ba_carryover	contract_nbr
52	k_sta_ba_reservation	contract_nbr
53	measure	contract_nbr
54	monthly_quantity_info2	contract_nbr
55	monthly_quantity_info2	related_contract_nbr
56	move_notification	downstream_k_nbr
57	move_notification	move_k_nbr
58	move_notification	upstream_k_nbr
59	move_order_rollup	downstream_k_nbr
60	move_order_rollup	move_k_nbr
61	move_order_rollup	upstream_k_nbr
62	mqi2	contract_nbr
63	mqi2	related_contract_nbr
64	pathleg	trn_k_nbr
65	payee_deck	contract_nbr
66	pmnt_component	contract_nbr
67	price_credit_info	contract_nbr
68	price_prov_adjs	contract_nbr
69	price_prov_adjs	netting_contract_nbr
70	price_prov_fixedvar	contract_nbr
71	price_provisions	contract_nbr
72	process_and_delivery	compression_k
73	process_and_delivery	contract_nbr

75	process_and_delivery	gather_k
76	process_and_delivery	process_k
77	process_and_delivery	transport_k
78	process_and_delivery	treated_k
79	prov_sta_ba_supertype	contract_nbr
80	provision_rcpt_dlvry	contract_nbr
81	provision_rcpt_dlvry_super	contract_nbr
82	provision_station_ba	contract_nbr
83	provision_stations	contract_nbr
84	provision_stations_supertyp	contract_nbr
85	purchase_contract	contract_nbr
86	py_assign	contract_nbr
87	quality	contract_nbr
88	regulatory	contract_nbr
89	related_contracts	contract_nbr
90	related_contracts	related_contract_nbr
91	release	contract_nbr
92	sales_accrual_trans	contract_nbr
93	sales_accrual_trans_detail	netting_k_nbr
94	sales_acctg_trans	contract_nbr
95	sales_acctg_trans_detail	netting_k_nbr
96	sales_contract	contract_nbr
97	scheduled_nomination	downstream_k_nbr
98	scheduled_nomination	upstream_k_nbr
99	seller_doi	from_contract_nbr
100	seller_doi	to_contract_nbr
101	seller_err	contract_nbr
102	seller_processing	contract_nbr
103	seller_transfers	contract_nbr
104	seller_rep	contract_nbr
105	susp_components	contract_nbr
106	taggs_k_xref	contract_nbr
107	taggs_k_xref	downstream_k_nbr
108	taggs_k_xref	upstream_k_nbr
109	temp_acctg_ksp_query	contract_nbr
110	temp_flow_date_contract	contract_nbr
111	temp_mqi	contract_nbr
112	temp_mqi	related_contract_nbr
113	transport_contract	contract_nbr
114	user_notes	contract_nbr

116	well_commitment	contract_nbr
117	well_dedications	contract_nbr

The other three parent tables are "busassoc", "pipeline", and "station" tables and they are referred as the foreign keys 146, 93, and 79 times respectively. About 100 peoples use the GIS client application programs and large amount of data need to be inserted into the child tables daily. If all these foreign keys are defined, the system performance will be significantly slowed down and is not acceptable for the business applications. Therefore, we need a solution to improve this system.

The performance improvement method discussed in last chapter is applied here. Delete triggers of these four parent tables were developed to maintain the data referential integrity during the parent table deletion. The system application interfaces and the database security system were developed with the considerations of the data referential integrity. The delete triggers, application interfaces, and the database security system are discussed in next two sections.

Triggers For Maintaining Referential Integrity during Data Deletion

As we discussed before, a delete trigger is needed for each of the four tables (contract, busassoc, pipeline, and station) which are the parent tables of many other child tables. When designing the delete trigger, we need to understand the business rules and identify which tables should be defined for cascade delete and which tables should be defined for restrict delete.

To illustrate the designs of the delete trigger for "contract" table, we need to analyze the child table information and their delete triggers. Table 2 lists the foreign

key column names, deletion definition (i.e. cascade or restrict deletion), and the delete triggers of the child tables. Some of the child tables are marked by **** in both cascade and restrict columns. A table marked by **** means that the table is either a history or a temporary table. The data in the history tables will always be saved for future references and will not be deleted regardless the data existence in the parent table. The temporary tables are used to store some temporary data in a process and the data in these tables will always be deleted after the process. Both the history and temporary tables will not affect the design of the delete trigger.

Table 2 Child tables information used for designing “contract” table delete trigger

	Table Name	Column Name	Cascade	Restrict	Child Trigger Cascade	Child Trigger Restrict
1	accrual_trans	contract_nbr	****	****	accrual_trans_detail	
2	accrual_trans_detail	netting_k_nbr	****	****		
3	acct_contract_info	contract_nbr	YES			
4	acctg_trans	contract_nbr		YES	acctg_trans_detail, acctg_payee_trans	
5	acctg_trans_detail	netting_k_nbr	****	****	acctg_payee_trans_detail	
6	alerts	contract_nbr	YES		alerts_action	
7	alerts_action	contract_nbr	YES			
8	allocated_volume	downstream_k_nbr		YES		
9	allocated_volume	upstream_k_nbr		YES		
10	amendment	contract_nbr	YES			
11	area_contract_ba	contract_nbr	YES			
12	bid	contract_nbr	YES			
13	bid	k_price_k_nbr	YES			
14	bid	tm_k_nbr	YES			
15	bid_received	contract_nbr	YES			
16	bid_received	k_price_k_nbr	YES			

17	bid_received	trn_k_nbr	YES			
18	billing	contract_nbr	YES			
19	contact_purpose_k	contract_nbr	YES			
20	contact_purpose_k_stations	contract_nbr	YES			
21	contract	contract_nbr				
22	contract_ba	alias_contract_nbr	****	****		area_contract_ba
23	contract_ba	contract_nbr	YES			area_contract_ba
24	contract_groups	contract_nbr	YES			
25	contract_misc	contract_nbr	YES			
26	contract_qty_pur_sls	contract_nbr	YES		provision_stations_supertype, prov_sta_ba_supertype	
27	contract_qty_trn_gth	contract_nbr	YES		provision_rcpt_dlvry_super	
28	contract_rate_schd	contract_nbr	YES			
29	contract_rcpt_dlvry_pts	contract_nbr	YES		provision_rcpt_dlvry	
30	contract_reservation	contract_nbr	****	****	provision_stations_supertype, provision_rcpt_dlvry_super	
31	contract_station_ba	contract_nbr	YES		k_sta_ba_reservation, k_sta_ba_carryover, release, contract_station_ba_cat, contract_station_ba_status	
32	contract_station_ba_cat	contract_nbr	YES			
33	contract_station_ba_status	contract_nbr	YES			
34	contract_status	contract_nbr	YES			
35	crescent_pricing	contract_nbr	YES			
36	deals	contract_nbr		YES	deals_trans, deals_order	
37	deals_order	contract_nbr	****	****	deals_pricing, deals_bs_order_vol_detail, acctg_trans, sales_acctg_trans	ppa_transfer_sellers, acctg_trans, sales_acctg_trans
38	deals_order	downstream_k_nbr		YES		
39	deals_order	upstream_k_nbr		YES		

40	deals_price_prov_adj	netting_contract_nbr		****		
41	deals_pricing	k_price_k_nbr	****	****	deals_price_prov_adj	
42	deals_trans	downstream_k_nbr		YES	deals_move_order_vol_detail	
43	deals_trans	trn_knbr		YES	deals_move_order_vol_detail	
44	deals_trans	upstream_k_nbr		YES	deals_move_order_vol_detail	
45	discrepancy	contract_nbr	YES			
46	discrepancy	downstream_k_nbr	YES			
47	discrepancy	upstream_k_nbr	YES			
48	gathering_contract	contract_nbr	YES			
49	invoice_component	contract_nbr	****	****		
50	k_area_dedication	contract_nbr	YES		area_contract_ba	
51	k_sta_ba_carryover	contract_nbr	YES			
52	k_sta_ba_reservation	contract_nbr	YES			
53	measure	contract_nbr	YES		provision_stations_supertype, provision_rcpt_dlvry_super	
54	monthly_quantity_info2	contract_nbr		YES		
55	monthly_quantity_info2	related_contract_nbr		YES		
56	move_notification	downstream_k_nbr	****	****		notices
57	move_notification	move_k_nbr	****	****		notices
58	move_notification	upstream_k_nbr	****	****		notices
59	move_order_rollup	downstream_k_nbr	****	****		
60	move_order_rollup	move_k_nbr	****	****		
61	move_order_rollup	upstream_k_nbr	****	****		
62	mqi2	contract_nbr	****	****		
63	mqi2	related_contract_nbr	****	****		
64	pathleg	trn_k_nbr		YES		
65	payee_deck	contract_nbr		YES		
66	pmnt_component	contract_nbr	****	****		

67	price_credit_info	contract_nbr	YES			
68	price_prov_adj	contract_nbr	YES			
69	price_prov_adj	netting_contract_nbr	YES			
70	price_prov_fixedvar	contract_nbr	YES			
71	price_provisions	contract_nbr	YES		crescent_pricing, price_prov_adj, price_credit_info, price_prov_fixedvar, provision_stations_supertype, prov_sta_ba_supertype	
72	process_and_delivery	compression_k	YES			
73	process_and_delivery	contract_nbr	YES			
74	process_and_delivery	dehydration_k	YES			
75	process_and_delivery	gather_k	YES			
76	process_and_delivery	process_k	YES			
77	process_and_delivery	transport_k	YES			
78	process_and_delivery	treated_k	YES			
79	prov_sta_ba_supertype	contract_nbr	YES		provision_station_ba	contract_qty_pur_sls
80	provision_rcpt_dlvry	contract_nbr	YES			
81	provision_rcpt_dlvry_super	contract_nbr	YES		provision_rcpt_dlvry	contract_qty_tm_gth, regulatory, measure, contract_reservation
82	provision_station_ba	contract_nbr	YES			
83	provision_stations	contract_nbr	YES			
84	provision_stations_supertype	contract_nbr	YES		provision_stations	regulatory, measure, contract_qty_pur_sls, quality, contract_reservation
85	purchase_contract	contract_nbr	YES			
86	py_assign	contract_nbr	****	****		
87	quality	contract_nbr	YES		provision_stations_supertype	
88	regulatory	contract_nbr	****	****	provision_stations_supertype, provision_rcpt_dlvry_super	
89	related_contracts	contract_nbr	YES			
90	related_contracts	related_contract_nbr		YES		

91	release	contract_nbr	YES			
92	sales_accrual_trans	contract_nbr	****	****	sales_accrual_trans_detail	
93	sales_accrual_trans_detail	netting_k_nbr	****	****		
94	sales_acctg_trans	contract_nbr		YES	sales_acctg_trans_detail	
95	sales_acctg_trans_detail	netting_k_nbr	****	****		
96	sales_contract	contract_nbr	YES			
97	scheduled_nomination	downstream_k_nbr		YES		
98	scheduled_nomination	upstream_k_nbr		YES		
99	seller_doi	from_contract_nbr	****	****		
100	seller_doi	to_contract_nbr	****	****		
101	seller_err	contract_nbr	****	****		
102	seller_processing	contract_nbr		YES		
103	seller_transfers	contract_nbr	****	****		
104	sellers_rep	contract_nbr	YES		provision_stations_supertype, prov_sta_ba_supertype	
105	susp_components	contract_nbr		YES		
106	taggs_k_xref	contract_nbr		YES		
107	taggs_k_xref	downstream_k_nbr		YES		
108	taggs_k_xref	upstream_k_nbr		YES		
109	temp_acctg_ksp_query	contract_nbr	****	****		
110	temp_flow_date_contract	contract_nbr	****	****		
111	temp_mqi	contract_nbr	****	****		
112	temp_mqi	related_contract_nbr	****	****		
113	transport_contract	contract_nbr	YES			
114	user_notes	contract_nbr	YES			
115	well_commit_status	contract_nbr	YES			well_commit_status_code, deals_order
116	well_commitment	contract_nbr	YES		well_commit_status, well_dedications	provision_stations, contract_station_ba
117	well_dedications	contract_nbr	YES			well_commitment

When designing a delete trigger with cascade deletions, we need to prevent the delete loop happened in the trigger. Based on the information of Table 2, delete loop will not be happened for the delete trigger of the "contract" table. The delete trigger developed using Sybase database trigger codes is attached in Appendix. The delete triggers for other three parent tables (i.e., "busassoc", "pipeline", "station") are also developed in GIS.

These four parent tables are controlled by a group of users working in the company administration section. This will add another layer of security for the data in these four parent tables.

Maintaining Referential Integrity during Data Insertion and Updating

As we discussed before, only the child table data deletion and updating require the referential integrity checks. These referential integrity checks are replaced by GIS application program interfaces, which will enforce correct data to be entered into the system. PowerBuilder's child data window technique was used when developing the application interfaces. PowerBuilder's child data windows will load the data from the parent tables into the select-only dropdown list boxes. Then, the child data window will be loaded into the application program window interfaces. System users can only pick up the correct data from the dropdown list boxes and use these data to start the database insert or update transactions for the child tables. When the insertion and updating transactions performed at the database, data integrity checks are not required anymore.

The GIS system has another layer of security system to stop the database transactions started by the direct SQL codes. When using direct SQL codes to manipulate the data in the production database, the users could enter bad data into the system and break the system data referential integrity. The GIS system only gives every system user the password used to enter the GIS client application program. After logging in GIS, the user's password will be swapped into the real database password by GIS system. The user's database password is stored in a password table in an encrypted form and only company's DBA has right to access this table.

With the help of the GIS application interfaces, data integrity of the child tables is enforced without performing any database referential integrity checks.

Triggers For Denormalized Database

GIS database is denormalized to allow many duplicated data in some of the child tables. The denormalized database can improve the database performance when querying data for the company's daily reports. One of the examples is to store "station name" in many child tables of the "station" table. When querying gas flow rate data and the name of a station, the station table is not required to join the query because the redundant column of the station name is in the child tables. However, the redundant data could create data referential problem. When the station name is changed in the "station" table, the duplicated "station name" columns in the child tables still have the old name. This problem can be resolved by an update trigger attached with the "station" table. This update trigger will be

fired when the "station name" of the "station" table is modified and all the new "station name" will be updated in all the redundant columns of the child tables. This update trigger could run for a long period of time because large amounts of data in the child tables need to be updated. It's better to run this process during off-peak period to reduce the database performance impact on the system. Also, very limited system users should be assigned with the update rights for the parent table to prevent other chances of mistakes.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Database performance in a large client server application system is affected by many factors. One important character of the client server database system is that some of the parent tables could be referred by hundreds of child tables through the foreign keys. When large amounts of data need to be inserted in the child tables, the database performance could be the system's bottleneck due to the foreign key referential integrity checks. Another minor performance problem in this system is that many table joins are required when querying data from the child tables for production reports. Several energy companies reported these performance problems in their natural gas transmission management systems.

This thesis discussed database performance improvement methods for the large client server application system. The proposed methods provide more efficient way of the referential integrity checks than those performed by foreign keys. No data integrity check time is required during the child data insertions when applying the proposed methods and data integrity can be ensured. The thesis also discussed the applications of denormalization technique in this application system to reduce the number of the database table joins when querying data. Table joins are very expensive in database performance.

The database improvement methods discussed in the thesis were applied in a Gas Information System (GIS) developed by Transok, Inc. The GIS is a large client server application system for natural gas transmission business management at Transok. The production results of GIS show positive responses for these improvements in the system.

REFERENCES

1. Anderson, G. W., Client/Server Database Design With Sybase: A High-Performance and Fine-Tuning Guide, Osborne McGraw-Hill, Berkeley, 1997
2. Brathwaite, K. S., Relation Databases - Concepts, Design, and Administration, McGraw-Hill Companies, Inc., New York, 1991.
3. Clifford, C., Mastering Sybase SQL Server 11, McGraw-Hill, a division of McGraw-Hill Companies, Berkeley, 1997
4. Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego, California, November, 1971.
5. Codd, E. F., "Further Normalization of the Data Base relational Model," Data Base Systems, Courant Computer Science Symposia Series 6, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
6. Codd, E. F., "Recent Investigations into Relational Data Base Systems," Proc. IFIP Congress, Stockholm, Sweden, 1974.
7. Codd, E. F., "Extending the Database Relation Model to Capture More Meaning," ACM Transactions on Database Systems 4, No.4, December, 1979.
8. Codd, E. F., "A relational Model of Data for Large Shared Data Banks," Communications of the ACM 13, No. 6, June 1970. Republished in Communications of the ACM 26, No. 1, January 1983.

9. Codd, E. F., "Domains, Keys, and Referential Integrity in Relational Databases," InfoDB3, No. 1, Spring, 1988.
10. Date, C. J., "Referential Integrity," Proc. 7th International Conference on Very Large Databases, Cannes, France, September, 1981.
11. Date, C. J., "Referential Integrity and Foreign Keys. Part I: Basic Concepts; PartII: Further Considerations," C. J. Dates, Relational Database Writings 1985-1989, Addison-Wesley Publishing Company, Reading, Mass, 1990.
12. Date, C. J., An Introduction to Database Systems, Sixth Edition, Addison-Wesley Publishing Company, Reading, MA, 1995
13. Fagin, R., "Multivalued Dependencies and a New Normal Form for Relational Databases," ACM TODS 2, No. 3, September, 1977.
14. Fagin, R., "Normal Forms and Relational Database Operators," Proc. 1979 ACM SIGMOD International Conference on Management of Data, Boston, Mass, May/June, 1979.
15. Garbus, J., Solomon, D., Tretter, B., and Rankins, R., Sybase SQL Server 11 DBA Survival Guide, Second Edition, Sams Publishing, Indianapolis, Indiana, 1996.
16. <http://sunsite.informatik.rwth-aachen.de/dblp/db/conf/index.a.html>
17. <http://sunsite.informatik.rwth-aachen.de/dblp/db/journals/index.html>
18. <http://www-db.stanford.edu/~ullman/ullman-papers.html>
19. <http://www.idea-group.com/jdm.htm>
20. <http://www.dbmsmag.com>
21. <http://www.dbpd.com>

22. Koch, G. and Loney, K., ORACLE8, the Complete Reference, Osborne McGraw-Hill, Berkeley, 1997
23. Logic Works, ErwinERX 2.5 for PowerBuilder User's manual, 1995.
24. Paulsell, K., Sybase SQL Server - Performance and Tuning Guide, International Thomson Computer Press, Boston, MA, 1996
25. Ramakrishnan, R., Database Management Systems, McGraw-Hill Companies, Inc., New York, 1997.
26. Roy , S. and Sugiyama, M., Sybase Performance Tuning, Prentice-Hall, Inc, A Simon & Schuster Company, Upper Saddle River, New Jersey, 1996.
27. Silberschatz, A., Korth, H. F. and Sudarshan, S., Database System Concepts, Third Edition, McGraw-Hill Companies, Inc., New York, 1997
28. Singh, L., Leigh, K., Zafian, J., et al, Oracle 7.3 Developer's Guide, Sams Publishing, Indianapolis, Indiana, 1997.
29. Soloman, D, Woodbeck, D., Ramkins, R., Garbus, J., McEwan, B., Microsoft SQL Server 6 Unleashed, Sams Publishing, Indianapolis, Indiana, 1996.
30. Ullman, J. and Widom, J., A First Course in Database Systems, Prentice-Hall, Inc, A Simon & Schuster Company, Upper Saddle River, New Jersey, 1997.
31. Urman, S., Oracle8 PL/SQL Programming, Osborne McGraw-Hill, Berkeley, 1997.
32. Wynkoop, S., Using Microsoft SQL Server 6.5, Second Edition, Que Corporation, Indianapolis, Indiana, 1997.

APPENDIX

Delete Trigger on Contract table for Sybase database

```
create trigger tD_contract on contract for DELETE as

begin
    declare @errno int,
            @errcnt int,
            @errmsg varchar(255)

    select @errno = 30001, @errcnt = 0
    select @errmsg = 'Cannot DELETE "contract" because the "contract_nbr"
data are used in the following place(s): '

    if exists (select * from deleted, acctg_trans
              where acctg_trans.contract_nbr = deleted.contract_nbr)
    begin
        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg + "'acctg_trans.contract_nbr', '
    end

    if exists (select * from deleted, allocated_volume
              where allocated_volume.upstream_k_nbr = deleted.contract_nbr)
    begin
        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg + "'allocated_volume.upstream_k_nbr", '
    end

    if exists (select * from deleted, allocated_volume
              where allocated_volume.downstream_k_nbr =
deleted.contract_nbr)
    begin
        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg +
"'allocated_volume.downstream_k_nbr', '
    end
```

```

if exists (select * from deleted, deals
          where deals.contract_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg + "deals.contract_nbr", '
end

if exists (select * from deleted, deals_order
          where deals_order.upstream_k_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg + "deals_order.upstream_k_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, deals_order
          where deals_order.downstream_k_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg + "deals_order.downstream_k_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, deals_trans
          where deals_trans.trn_knbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg + "deals_trans.trn_knbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, deals_trans
          where deals_trans.upstream_k_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg + "deals_trans.upstream_k_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, deals_trans
          where deals_trans.downstream_k_nbr = deleted.contract_nbr)
begin

```

```

        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg + "deals_trans.downstream_k_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, monthly_quantity_info2
          where monthly_quantity_info2.contract_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg +
"monthly_quantity_info2.contract_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, monthly_quantity_info2
          where monthly_quantity_info2.related_contract_nbr =
deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg +
"monthly_quantity_info2.related_contract_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, pathleg
          where pathleg.trn_k_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg + "pathleg.trn_k_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, payee_deck
          where payee_deck.contract_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
    select @errmsg = @errmsg + "payee_deck.contract_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, related_contracts

```

```

        where related_contracts.related_contract_nbr =
deleted.contract_nbr)
    begin
        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg +
"related_contracts.related_contract_nbr", '
    end

    if @errcnt = 5
        goto error
    if exists (select * from deleted, sales_acctg_trans
        where sales_acctg_trans.contract_nbr = deleted.contract_nbr)
    begin
        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg + "sales_acctg_trans.contract_nbr", '
    end

    if @errcnt = 5
        goto error
    if exists (select * from deleted, scheduled_nomination
        where scheduled_nomination.upstream_k_nbr =
deleted.contract_nbr)
    begin
        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg +
"scheduled_nomination.upstream_k_nbr", '
    end

    if @errcnt = 5
        goto error
    if exists (select * from deleted, scheduled_nomination
        where scheduled_nomination.downstream_k_nbr =
deleted.contract_nbr)
    begin
        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg +
"scheduled_nomination.downstream_k_nbr", '
    end

    if @errcnt = 5
        goto error
    if exists (select * from deleted, seller_processing
        where seller_processing.contract_nbr = deleted.contract_nbr)
    begin
        select @errcnt = @errcnt + 1
        select @errmsg = @errmsg + "seller_processing.contract_nbr", '

```

```

end

if @errcnt = 5
    goto error
if exists (select * from deleted, susp_components
           where susp_components.contract_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
           select @errmsg = @errmsg + "susp_components.contract_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, taggs_k_xref
           where taggs_k_xref.contract_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
           select @errmsg = @errmsg + "taggs_k_xref.contract_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, taggs_k_xref
           where taggs_k_xref.upstream_k_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
           select @errmsg = @errmsg + "taggs_k_xref.upstream_k_nbr", '
end

if @errcnt = 5
    goto error
if exists (select * from deleted, taggs_k_xref
           where taggs_k_xref.downstream_k_nbr = deleted.contract_nbr)
begin
    select @errcnt = @errcnt + 1
           select @errmsg = @errmsg + "taggs_k_xref.downstream_k_nbr", '
end

if @errcnt > 0
    goto error
delete acct_contract_info
       from acct_contract_info, deleted
       where acct_contract_info.contract_nbr = deleted.contract_nbr

delete alerts
       from alerts, deleted

```

```

        where alerts.contract_nbr = deleted.contract_nbr

delete alerts_action
  from alerts_action, deleted
  where alerts_action.contract_nbr = deleted.contract_nbr

delete amendment
  from amendment, deleted
  where amendment.contract_nbr = deleted.contract_nbr

delete area_contract_ba
  from area_contract_ba, deleted
  where area_contract_ba.contract_nbr = deleted.contract_nbr

delete bid
  from bid, deleted
  where bid.trn_k_nbr = deleted.contract_nbr

delete bid
  from bid, deleted
  where bid.contract_nbr = deleted.contract_nbr

delete bid
  from bid, deleted
  where bid.k_price_k_nbr = deleted.contract_nbr

delete bid_received
  from bid_received, deleted
  where bid_received.trn_k_nbr = deleted.contract_nbr

delete bid_received
  from bid_received, deleted
  where bid_received.contract_nbr = deleted.contract_nbr

delete bid_received
  from bid_received, deleted
  where bid_received.k_price_k_nbr = deleted.contract_nbr

delete billing
  from billing, deleted
  where billing.contract_nbr = deleted.contract_nbr

delete contact_purpose_k
  from contact_purpose_k, deleted
  where contact_purpose_k.contract_nbr = deleted.contract_nbr

```



```

delete contact_purpose_k_stations
  from contact_purpose_k_stations, deleted
  where contact_purpose_k_stations.contract_nbr =
deleted.contract_nbr

delete contract_ba
  from contract_ba, deleted
  where contract_ba.contract_nbr = deleted.contract_nbr

delete contract_groups
  from contract_groups, deleted
  where contract_groups.contract_nbr = deleted.contract_nbr

delete contract_misc
  from contract_misc, deleted
  where contract_misc.contract_nbr = deleted.contract_nbr

delete contract_qty_pur_sls
  from contract_qty_pur_sls, deleted
  where contract_qty_pur_sls.contract_nbr = deleted.contract_nbr

delete contract_qty_trn_gth
  from contract_qty_trn_gth, deleted
  where contract_qty_trn_gth.contract_nbr = deleted.contract_nbr

delete contract_rate_schd
  from contract_rate_schd, deleted
  where contract_rate_schd.contract_nbr = deleted.contract_nbr

delete contract_rcpt_dlvry_pts
  from contract_rcpt_dlvry_pts, deleted
  where contract_rcpt_dlvry_pts.contract_nbr = deleted.contract_nbr

delete contract_station_ba_cat
  from contract_station_ba_cat, deleted
  where contract_station_ba_cat.contract_nbr = deleted.contract_nbr

delete contract_station_ba_status
  from contract_station_ba_status, deleted
  where contract_station_ba_status.contract_nbr =
deleted.contract_nbr

delete contract_status
  from contract_status, deleted
  where contract_status.contract_nbr = deleted.contract_nbr

delete crescent_pricing
  from crescent_pricing, deleted

```

```

        where crescent_pricing.contract_nbr = deleted.contract_nbr

delete discrepancy
  from discrepancy, deleted
  where discrepancy.contract_nbr = deleted.contract_nbr

delete discrepancy
  from discrepancy, deleted
  where discrepancy.upstream_k_nbr = deleted.contract_nbr

delete discrepancy
  from discrepancy, deleted
  where discrepancy.downstream_k_nbr = deleted.contract_nbr

delete gathering_contract
  from gathering_contract, deleted
  where gathering_contract.contract_nbr = deleted.contract_nbr

delete k_area_dedication
  from k_area_dedication, deleted
  where k_area_dedication.contract_nbr = deleted.contract_nbr

delete k_sta_ba_carryover
  from k_sta_ba_carryover, deleted
  where k_sta_ba_carryover.contract_nbr = deleted.contract_nbr

delete k_sta_ba_reservation
  from k_sta_ba_reservation, deleted
  where k_sta_ba_reservation.contract_nbr = deleted.contract_nbr

delete measure
  from measure, deleted
  where measure.contract_nbr = deleted.contract_nbr

delete price_credit_info
  from price_credit_info, deleted
  where price_credit_info.contract_nbr = deleted.contract_nbr
delete price_prov_adj
  from price_prov_adj, deleted
  where price_prov_adj.contract_nbr = deleted.contract_nbr

delete price_prov_adj
  from price_prov_adj, deleted
  where price_prov_adj.netting_contract_nbr = deleted.contract_nbr

delete price_prov_fixedvar
  from price_prov_fixedvar, deleted
  where price_prov_fixedvar.contract_nbr = deleted.contract_nbr

```

```

delete price_provisions
  from price_provisions, deleted
  where price_provisions.contract_nbr = deleted.contract_nbr

delete process_and_delivery
  from process_and_delivery, deleted
  where process_and_delivery.gather_k = deleted.contract_nbr

delete process_and_delivery
  from process_and_delivery, deleted
  where process_and_delivery.process_k = deleted.contract_nbr

delete process_and_delivery
  from process_and_delivery, deleted
  where process_and_delivery.treated_k = deleted.contract_nbr

delete process_and_delivery
  from process_and_delivery, deleted
  where process_and_delivery.transport_k = deleted.contract_nbr

delete process_and_delivery
  from process_and_delivery, deleted
  where process_and_delivery.contract_nbr = deleted.contract_nbr

delete process_and_delivery
  from process_and_delivery, deleted
  where process_and_delivery.compression_k =
deleted.contract_nbr

delete process_and_delivery
  from process_and_delivery, deleted
  where process_and_delivery.dehydration_k = deleted.contract_nbr

delete sellers_rep
  from sellers_rep, deleted
  where sellers_rep.contract_nbr = deleted.contract_nbr

delete prov_sta_ba_supertype
  from prov_sta_ba_supertype, deleted
  where prov_sta_ba_supertype.contract_nbr = deleted.contract_nbr

delete provision_rcpt_dlvry
  from provision_rcpt_dlvry, deleted
  where provision_rcpt_dlvry.contract_nbr = deleted.contract_nbr

delete provision_rcpt_dlvry_super
  from provision_rcpt_dlvry_super, deleted

```

```

        where provision_rcpt_dlvry_super.contract_nbr =
deleted.contract_nbr

delete provision_station_ba
    from provision_station_ba, deleted
    where provision_station_ba.contract_nbr = deleted.contract_nbr

delete contract_station_ba
    from contract_station_ba, deleted
    where contract_station_ba.contract_nbr = deleted.contract_nbr

delete provision_stations
    from provision_stations, deleted
    where provision_stations.contract_nbr = deleted.contract_nbr

delete provision_stations_supertype
    from provision_stations_supertype, deleted
    where provision_stations_supertype.contract_nbr =
deleted.contract_nbr

delete purchase_contract
    from purchase_contract, deleted
    where purchase_contract.contract_nbr = deleted.contract_nbr

delete quality
    from quality, deleted
    where quality.contract_nbr = deleted.contract_nbr

delete related_contracts
    from related_contracts, deleted
    where related_contracts.contract_nbr = deleted.contract_nbr

delete release
    from release, deleted
    where release.contract_nbr = deleted.contract_nbr

delete sales_contract
    from sales_contract, deleted
    where sales_contract.contract_nbr = deleted.contract_nbr

delete transport_contract
    from transport_contract, deleted
    where transport_contract.contract_nbr = deleted.contract_nbr

delete user_notes
    from user_notes, deleted

```

```

        where user_notes.contract_nbr = deleted.contract_nbr

delete well_commitment
    from well_commitment, deleted
    where well_commitment.contract_nbr = deleted.contract_nbr

delete well_commit_status
    from well_commit_status, deleted
    where well_commit_status.contract_nbr = deleted.contract_nbr

delete well_dedications
    from well_dedications, deleted
    where well_dedications.contract_nbr = deleted.contract_nbr

return

error:
select @errmsg = substring(@errmsg, 1, char_length(@errmsg) - 2)
if @errcnt = 5
    select @errmsg = @errmsg + '...'

raiserror @errno @errmsg
rollback transaction
end

```

VITA

Hui Xu

Candidate for the Degree of

Master of Science

Thesis: IMPROVE DATABASE PERFORMANCE AND MAINTAIN
REFERENTIAL INTEGRITY FOR A LARGE CLIENT SERVER
APPLICATION SYSTEM

Major Field: Computer Science

Biographical:

Education: Graduated from Petroleum University, Shandong, China in 1982; received Bachelor of Science Degree in Petroleum Engineering. Graduated from Research Institute of Petroleum Exploration & Development, Beijing, China in 1985; received Master of Science Degree in Petroleum Engineering. Graduated from Louisiana State University, Baton Rouge, Louisiana in 1990; received Doctor of Philosophy Degree in Petroleum Engineering. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in July 1999.

Experience: Petroleum Engineer, worked at Louisiana State University, Baton Rouge, Louisiana; Amoco Production Company, Tulsa, Oklahoma; and Dahua Energy Corporation, Beijing, China from 1990 to 1993. Program Analyst, worked at MiraTech Consulting Group, Inc., Tulsa, Oklahoma; Transok, Inc., Tulsa, Oklahoma; and El Paso Energy, Houston, Texas from 1993 to 1998. Senior Engineer, employed by Schlumberger Austin Product Center, Austin, Texas from 1998 to present.