Updating Semi-Structured Data

Pensri Amornsinlaphachai

A thesis submitted in partial fulfillment of the requirements
of the University of Northumbria at Newcastle
for the degree of Doctor of Philosophy
in Computer Science

Research undertaken in the School of
Computing, Engineering and Information Sciences

June 2007

Content

Abstract

The Web has had a tremendous success with its support for the rapid and inexpensive exchange of information. A considerable body of data exchange is in the form of semi-structured data such as the eXtensible Markup Language (XML). XML, an effective standard to represent and exchange semi-structured data on the Web, is used ubiquitously in almost all areas of information technology. Most researchers in the XML area have concentrated on storing, querying and publishing XML while not many have paid attention to updating XML; thus the XML update area is not fully developed.

We propose a solution for updating XML as a representation of semi-structured data. XML is updated through an object-relational database (ORDB) to exploit the maturity of the relational engine and the newer object features of the OR technology. The engine is used to enforce constraints during the updating of the XML whereas the object features are used to handle the XML hierarchical structure. Updating XML via ORDB makes it easier to join XML documents in an update and in turn joins of XML documents make it possible to keep non-redundant data in multiple XML documents.

This thesis contributes a solution for the update of XML documents via an ORDB to advance our understanding of the XML update area. Rules for mapping XML structure and constraints to an ORDB schema are presented and a mechanism to handle XML cardinality constraint is provided. An XML update language, an extension to XQuery, has been designed and this language is translated into the standard SQL executed on an ORDB. To handle the recursive nature of XML, a recursive function updating XML data is translated into SQL commands equipped with a programming capability. A method is developed to reflect the changes from the ORDB to XML documents.

A prototype of the solution has been implemented to help validate our approach. Experimental study to evaluate the performance of XML update processing based on the prototype has been conducted. The experimental results show that updating multiple XML documents storing non-redundant data yields a better performance than updating a single XML document storing redundant data; an ORDB can take advantage of this by caching data to a greater extent than a native XML database.

The solution of updating XML documents via an ORDB can solve some problems in existing update methods as follows. Firstly, the preservation of XML constraints is handled by the ORDB engine. Secondly, non-redundant data is stored in linked XML documents; thus the problem of data inconsistency and low performance caused by data redundancy are solved. Thirdly, joins of XML documents are converted to joins of tables in SQL. Fourthly, fields or tables involved in regular path expressions can be tackled in a short time by using mapping data. Finally, a recursive function is translated into SQL commands equipped with a programming capability.

Abstract

The Web has had a tremendous success with its support for the rapid and inexpensive exchange of information. A considerable body of data exchange is in the form of semi-structured data such as the eXtensible Markup Language (XML). XML, an effective standard to represent and exchange semi-structured data on the Web, is used ubiquitously in almost all areas of information technology. Most researchers in the XML area have concentrated on storing, querying and publishing XML while not many have paid attention to updating XML; thus the XML update area is not fully developed.

We propose a solution for updating XML as a representation of semi-structured data. XML is updated through an object-relational database (ORDB) to exploit the maturity of the relational engine and the newer object features of the OR technology. The engine is used to enforce constraints during the updating of the XML whereas the object features are used to handle the XML hierarchical structure. Updating XML via ORDB makes it easier to join XML documents in an update and in turn joins of XML documents make it possible to keep non-redundant data in multiple XML documents.

This thesis contributes a solution for the update of XML documents via an ORDB to advance our understanding of the XML update area. Rules for mapping XML structure and constraints to an ORDB schema are presented and a mechanism to handle XML cardinality constraint is provided. An XML update language, an extension to XQuery, has been designed and this language is translated into the standard SQL executed on an ORDB. To handle the recursive nature of XML, a recursive function updating XML data is translated into SQL commands equipped with a programming capability. A method is developed to reflect the changes from the ORDB to XML documents.

A prototype of the solution has been implemented to help validate our approach. Experimental study to evaluate the performance of XML update processing based on the prototype has been conducted. The experimental results show that updating multiple XML documents storing non-redundant data yields a better performance than updating a single XML document storing redundant data; an ORDB can take advantage of this by caching data to a greater extent than a native XML database.

The solution of updating XML documents via an ORDB can solve some problems in existing update methods as follows. Firstly, the preservation of XML constraints is handled by the ORDB engine. Secondly, non-redundant data is stored in linked XML documents; thus the problem of data inconsistency and low performance caused by data redundancy are solved. Thirdly, joins of XML documents are converted to joins of tables in SQL. Fourthly, fields or tables involved in regular path expressions can be tackled in a short time by using mapping data. Finally, a recursive function is translated into SQL commands equipped with a programming capability.

Abstract

The Web has had a tremendous success with its support for the rapid and inexpensive exchange of information. A considerable body of data exchange is in the form of semi-structured data such as the eXtensible Markup Language (XML). XML, an effective standard to represent and exchange semi-structured data on the Web, is used ubiquitously in almost all areas of information technology. Most researchers in the XML area have concentrated on storing, querying and publishing XML while not many have paid attention to updating XML; thus the XML update area is not fully developed.

We propose a solution for updating XML as a representation of semi-structured data. XML is updated through an object-relational database (ORDB) to exploit the maturity of the relational engine and the newer object features of the OR technology. The engine is used to enforce constraints during the updating of the XML whereas the object features are used to handle the XML hierarchical structure. Updating XML via ORDB makes it easier to join XML documents in an update and in turn joins of XML documents make it possible to keep non-redundant data in multiple XML documents.

This thesis contributes a solution for the update of XML documents via an ORDB to advance our understanding of the XML update area. Rules for mapping XML structure and constraints to an ORDB schema are presented and a mechanism to handle XML cardinality constraint is provided. An XML update language, an extension to XQuery, has been designed and this language is translated into the standard SQL executed on an ORDB. To handle the recursive nature of XML, a recursive function updating XML data is translated into SQL commands equipped with a programming capability. A method is developed to reflect the changes from the ORDB to XML documents.

A prototype of the solution has been implemented to help validate our approach. Experimental study to evaluate the performance of XML update processing based on the prototype has been conducted. The experimental results show that updating multiple XML documents storing non-redundant data yields a better performance than updating a single XML document storing redundant data; an ORDB can take advantage of this by caching data to a greater extent than a native XML database.

The solution of updating XML documents via an ORDB can solve some problems in existing update methods as follows. Firstly, the preservation of XML constraints is handled by the ORDB engine. Secondly, non-redundant data is stored in linked XML documents; thus the problem of data inconsistency and low performance caused by data redundancy are solved. Thirdly, joins of XML documents are converted to joins of tables in SQL. Fourthly, fields or tables involved in regular path expressions can be tackled in a short time by using mapping data. Finally, a recursive function is translated into SQL commands equipped with a programming capability.

Abstract

The Web has had a tremendous success with its support for the rapid and inexpensive exchange of information. A considerable body of data exchange is in the form of semi-structured data such as the eXtensible Markup Language (XML). XML, an effective standard to represent and exchange semi-structured data on the Web, is used ubiquitously in almost all areas of information technology. Most researchers in the XML area have concentrated on storing, querying and publishing XML while not many have paid attention to updating XML; thus the XML update area is not fully developed.

We propose a solution for updating XML as a representation of semi-structured data. XML is updated through an object-relational database (ORDB) to exploit the maturity of the relational engine and the newer object features of the OR technology. The engine is used to enforce constraints during the updating of the XML whereas the object features are used to handle the XML hierarchical structure. Updating XML via ORDB makes it easier to join XML documents in an update and in turn joins of XML documents make it possible to keep non-redundant data in multiple XML documents.

This thesis contributes a solution for the update of XML documents via an ORDB to advance our understanding of the XML update area. Rules for mapping XML structure and constraints to an ORDB schema are presented and a mechanism to handle XML cardinality constraint is provided. An XML update language, an extension to XQuery, has been designed and this language is translated into the standard SQL executed on an ORDB. To handle the recursive nature of XML, a recursive function updating XML data is translated into SQL commands equipped with a programming capability. A method is developed to reflect the changes from the ORDB to XML documents.

A prototype of the solution has been implemented to help validate our approach. Experimental study to evaluate the performance of XML update processing based on the prototype has been conducted. The experimental results show that updating multiple XML documents storing non-redundant data yields a better performance than updating a single XML document storing redundant data; an ORDB can take advantage of this by caching data to a greater extent than a native XML database.

The solution of updating XML documents via an ORDB can solve some problems in existing update methods as follows. Firstly, the preservation of XML constraints is handled by the ORDB engine. Secondly, non-redundant data is stored in linked XML documents; thus the problem of data inconsistency and low performance caused by data redundancy are solved. Thirdly, joins of XML documents are converted to joins of tables in SQL. Fourthly, fields or tables involved in regular path expressions can be tackled in a short time by using mapping data. Finally, a recursive function is translated into SQL commands equipped with a programming capability.

Updating Semi-Structured Data

Pensri Amornsinlaphachai

PhD

2007

Updating Semi-Structured Data

Pensri Amornsinlaphachai

A thesis submitted in partial fulfillment of the requirements
of the University of Northumbria at Newcastle
for the degree of Doctor of Philosophy
in Computer Science

Research undertaken in the School of
Computing, Engineering and Information Sciences

June 2007

Abstract

The Web has had a tremendous success with its support for the rapid and inexpensive exchange of information. A considerable body of data exchange is in the form of semi-structured data such as the eXtensible Markup Language (XML). XML, an effective standard to represent and exchange semi-structured data on the Web, is used ubiquitously in almost all areas of information technology. Most researchers in the XML area have concentrated on storing, querying and publishing XML while not many have paid attention to updating XML; thus the XML update area is not fully developed.

We propose a solution for updating XML as a representation of semi-structured data. XML is updated through an object-relational database (ORDB) to exploit the maturity of the relational engine and the newer object features of the OR technology. The engine is used to enforce constraints during the updating of the XML whereas the object features are used to handle the XML hierarchical structure. Updating XML via ORDB makes it easier to join XML documents in an update and in turn joins of XML documents make it possible to keep non-redundant data in multiple XML documents.

This thesis contributes a solution for the update of XML documents via an ORDB to advance our understanding of the XML update area. Rules for mapping XML structure and constraints to an ORDB schema are presented and a mechanism to handle XML cardinality constraint is provided. An XML update language, an extension to XQuery, has been designed and this language is translated into the standard SQL executed on an ORDB. To handle the recursive nature of XML, a recursive function updating XML data is translated into SQL commands equipped with a programming capability. A method is developed to reflect the changes from the ORDB to XML documents.

A prototype of the solution has been implemented to help validate our approach. Experimental study to evaluate the performance of XML update processing based on the prototype has been conducted. The experimental results show that updating multiple XML documents storing non-redundant data yields a better performance than updating a single XML document storing redundant data; an ORDB can take advantage of this by caching data to a greater extent than a native XML database.

The solution of updating XML documents via an ORDB can solve some problems in existing update methods as follows. Firstly, the preservation of XML constraints is handled by the ORDB engine. Secondly, non-redundant data is stored in linked XML documents; thus the problem of data inconsistency and low performance caused by data redundancy are solved. Thirdly, joins of XML documents are converted to joins of tables in SQL. Fourthly, fields or tables involved in regular path expressions can be tackled in a short time by using mapping data. Finally, a recursive function is translated into SQL commands equipped with a programming capability.

Content

List of Tables

List of Figures

Acknowledgements

This thesis would not have been possible without the assistance and support of many great people. I would like to take this opportunity to record my deepest gratitude to the great people who have contributed towards the completion of this thesis.

I would like to express my heartfelt gratitude to my two supervisors, Dr. Nick Rossiter and Dr. Muhammad Akhtar Ali. I owe a great debt to Dr. Rossiter who has given me intensive supervision through my study, I thank him for his time with critical reading, for his encouragement, valuable advice and suggestions. I would like also to thank Dr. Muhammad Akhtar Ali who has given me technical advice. I appreciate the time he put aside for meetings that always give me some knowledge, thanks for his support and attention.

I am also grateful to both Prof. Paul Watson and Dr. Paul Vickers for serving as members of my examination committee.

I would like to thank all the staff in the School of Computing, Engineering and Information Sciences for helping and supporting all study resources.

I also would like to thank Duenpen and Paijit Kochakornjarupong, and my two flat mates, Jutima Meethaneethorn and Marisa Intawong who looked after me during my sickness.

I also thank all other my friends who have supported me and helped me in any form in particular Kwanjai Deejring who has protected the benefit for my family all the time that I have been in UK.

I would like to acknowledge the financial support of the Royal Thai government who gave me an opportunity to study in UK.

These acknowledgements would not be complete without thanks to my mother, sisters and brothers for all the support and encouragement they have given me. I cannot thank my mother enough for her love and continuous support.

This thesis is dedicated to the memory of my dearest father, whose constant inspiration showed me how to be strong during the hardest of times.

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning.

Name: PENSRI AMORNSINLAPHACHAI

Signature: Pensri Amornsinlaphachai

Date: 28/6/2007

Chapter 1: Introduction

Conventional database management systems whose data models are based upon regular structures, e.g. a relational model, are used predominantly today. The Web is crowded with different forms of data ranging from unstructured data to highly structured data and this is a huge achievement. This is because the Web provides the capability to exchange information rapidly on a world-wide scale. With the proliferation of heterogeneous information over the Web, data integration becomes necessary and the need for semi-structured data emerges naturally. This brings the theme of semi-structured data to the front line of research.

Semi-structured data is data with an irregular structure whose schema may not be known fully in advance. The irregular structure means that some elements may be missing. For example, in documents holding restaurant data, the address element may be missing in some cases. Hyper Text Markup Language (HTML), Standard Generalized Markup Language (SGML) and eXtensible Markup Language (XML) are instances of languages delivering semi-structured data.

The amount of semi-structured data on the Internet is growing rapidly. As we know, HTML is utilised to present data on the Web. However, HTML was just designed to display data and to concentrate on how the data looks whereas XML was designed to describe data and to concentrate on what the data is. Furthermore, XML is flexible in that it allows the authors to define their own tags and structure for documents and can handle data whose occurrence is optional. Therefore XML has become an effective standard for representing and exchanging semi-structured data on the Web; thereby our research focuses on XML as the standard representation for semi-structured data.

The Web is employed not only as a server for hypertext but also as a fundamental tool for running enterprise applications. Developing business applications with Web tools is a laborious task. The information is kept in HTML pages; thus searching or updating data is an irksome chore. Moreover, ad hoc applications may be needed to handle the consistency. With these difficulties, the database technology has become a significant player in Web applications.

An archetypal scenario for Web applications exploits the database technology as follows. An application is developed on top of a relational or XML database. Users query or update the data in the database via this application and the output data is converted to HTML format to display on the Web or to be sent in XML format over the Internet. Consider another scenario: business information such as order data or invoice data is sent across the Web in XML format, the Web application receives this data and keeps it in XML format or other formats for reference purposes. Users can query or modify this data when it is required.

From the above scenarios, the updating of XML data is needed. Several methods are used to update XML documents such as XML editors, DOM (Document Object Model) applications and XML databases. Using XML editors [37] is slightly better than updating XML manually; however, a whole XML document must be retrieved, the change incorporated into the document and the document then saved to reflect the change. An update in this manner takes minutes instead of seconds. An XML document can be updated by DOM applications [58]; however, to check data integrity, a whole XML document must be validated after updating rather than checking constraints during the update process. Updating XML documents stored in XML databases is more convenient since they provide some XML update languages. Thus XML databases look promising. Unfortunately, the update mechanism in existing XML databases is not fully developed. The existing XML databases such as Xindice [8], eXist [91] and X-Hive [139] do not support or guarantee the preservation of constraints [12]. Problems in updating XML are not only the preservation of constraints but also the low performance caused by storing redundant data and the use of regular path expressions. The existing XML update languages such as Lorel [4], XML-GL [26], XML-RL [83] and XUpdate [140] are simple update languages. All of these languages cannot join documents in updates and some of them cannot update data whose structure is recursive, and some have no policy for preserving the order of XML elements during an update.

The goal of this research is to devise a more effective solution for updating XML data as the standard representation of semi-structured data in the database management view.

The rest of this chapter is organised as follows. Firstly, the motivation of this research is described in section 1.1. Secondly, the problem statements of the research are

explained in section 1.2 and the overview of our approach is delineated in section 1.3. Thirdly, the research question and hypotheses are described in section 1.4 and the contributions are elucidated in section 1.4. Finally, the organisation of the thesis is outlined in section 1.5.

1.1 Motivation

In applications storing data in XML format, querying and updating XML data are necessary. In the research area of semi-structured data, managing XML has been studied extensively. However, little research into the updating of XML documents has been proposed. Even W3C's XQuery, an XML query language which is a standard for querying XML data, has not provided update features. This is because W3C wanted to publish a standard for an XML query language as soon as possible [29]; hence the area of updating XML data is not fully developed as it has been perceived of lower priority.

There are two dominant approaches for managing XML repositories in use today. The first approach for pure semi-structured data is to use native XML databases (NXD) to handle the data such as Lore [89], Timber [65] and Sedna [55]. Commonly, the data model of an NXD is represented by a hierarchical graph or tree. This approach provides a more natural data model for XML documents since the XML hierarchical structure can fit well with a tree or graph model. However, searching data in this model must start from the document root. The second approach is to apply traditional databases: relational database (RDB), object-relational database (ORDB) and object-oriented database (OODB), by mapping XML data to the databases [40, 73, 118]. With this approach, XML data must be converted to keep it in the database and an XML query language is translated into the query language executed by the underlying database system. The result from the query is converted to XML format, thus the cost for data conversion is expensive. However, this approach exploits maturity, stability and scalability of traditional databases. Unfortunately, none of these approaches have paid much attention to updating XML data.

Much research in the XML area has focused on storing, publishing, and querying XML documents. XML consequently provides most of the features normally expected for a database model. However, there is an omission in that most existing work does not pay much attention to modifying XML or does not mention it at all. For example, the Lore

[51] and STORED [40] systems use Lorel [4] as a query language but Lorel only supports simple creation and deletion operations, and modification to the value of existing nodes on a Lore data graph. Lorel does not even provide an insert operator. Timber uses TAX tree algebra [66] instead of a language to present update operations; however an update feature is not implemented. Sedna utilises XUpdate [140] to provide more complex creation, insertion, deletion and value updating but update operations can only be performed on a particular document. A number of works [2, 73, 108, 118, 119] previously performed have not mentioned updating XML.

Currently, only a few XML update languages have been implemented such as XUpdate. However these update languages perform updates on a specific document and using regular path expressions can yield low performance. Most work on XML databases has not mentioned the preservation of constraints when updates are performed. In particular the referential integrity constraint is not supported in any XML database [20] including Ipedo [63], Tamino [120] and X-Hive [139] which are commercial XML databases; moreover, they cannot perform joins between documents in update commands.

Two possible reasons for the immaturity of the XML update area are as follows: firstly XQuery has not provided update features because W3C wanted to release the standard of XQuery as soon as possible [29]. Thus this research area has not been investigated as it ought to have been; however there is a suggestion from W3C [29] for the future release of an update version in XQuery. Secondly, existing work is focused on updating XML by employing a native XML database. Thus a host of work such as preserving constraints must be created from scratch which can take a long time.

To conclude, our research is motivated by the immaturity of the XML update area exposing several weaknesses such as preserving constraints, data inconsistency, low performance and the joins of documents in updating; thus this thesis has proposed a solution for updating XML documents to advance understanding of the area.

1.2 Problem Statement

As mentioned earlier, research in the area of updating XML is not fully-fledged. Thus several problems in updating XML documents have been exposed. This thesis concentrates on five main problems as follows.

1.2.1 Preserving constraints

As in traditional databases such as (object-)relational database, the schema of XML can be defined to control the structure of XML documents and the constraints of XML can be specified in the schema. Currently, several schemas such as DTD (Document Type Definition), XML Schema and XDR (XML-Data Reduced schema) are proposed but only two schemas, DTD and XML Schema, are recommended by W3C. Usually, when XML documents are created, their contents conform to the constraints in their schemas; the ideal is that these constraints on XML documents should be preserved when such documents are updated. However, the existing work can update XML documents but only without checking constraints. Even commercial products cannot guarantee the integrity in the database when XML data is updated [12].

1.2.2 Data inconsistency and low performance

Normally, all XML data is kept in one document; thus data redundancy may occur. This can lead to data inconsistency and low performance when updates are performed. Two possible approaches to solve this problem are as follows: the first approach is used for storing XML in an RDB [11, 31] by reducing data redundancy during mapping XML to an RDB. With this method, data in XML documents differs from data in the RDB; thus it becomes difficult to maintain different data between the two storage systems. The second approach [27] is splitting an XML document into several documents to store non-redundant data in these documents; inevitably though this leads to the necessity for joins of documents in updating.

1.2.3 Join of documents in updates

When non-redundant data is kept in several separate XML documents, joins of XML documents in updating are necessary since data in these separate documents have a relationship with one another (data in one document which will be updated may relate to data in other documents). Presently, joins between XML documents in querying can be performed in XQuery [27]; however XQuery can only join by values as joins by pointer cannot be performed. In updating XML documents, no XML update language supports joins of XML documents both in the case of joins by values and of joins by pointer.

1.2.4 Updating XML whose structure is partially known

Frequently the XML structure is unknown or only partially known; thus regular path expressions must be used rather than the full paths of XML documents in XML query or update languages. Using regular path expressions, especially a descendent path expression ('//'), can slow the process of querying or updating data drastically [137] because the query engine must traverse all possible paths in XML documents to find target data. Hence if a database engine does not provide a good optimisation technique or other techniques to handle the regular path expressions, an unacceptable performance can occur.

1.2.5 Updating recursive XML documents

Sometimes the structure of XML documents can be recursive. In XQuery, there is no specific facility to query data whose structure is recursive; however the effect can be achieved by creating a recursive user-defined function. For applying conventional databases to handle XML documents, until now no technique has been proposed to translate a recursive feature of XQuery into SQL. Furthermore, some researchers [86, 87] indicate this recursion is an untranslatable feature. Thus for updating XML data whose structure is recursive, none of the work in this area can do this task.

To illuminate, an example related to these five problems is described as follows. We keep the data of publications in an XML document as shown in Figure 1.1 by using a

native XML database (such as X-Hive). The schema of the document is shown in Figure 7.1 (page 156).

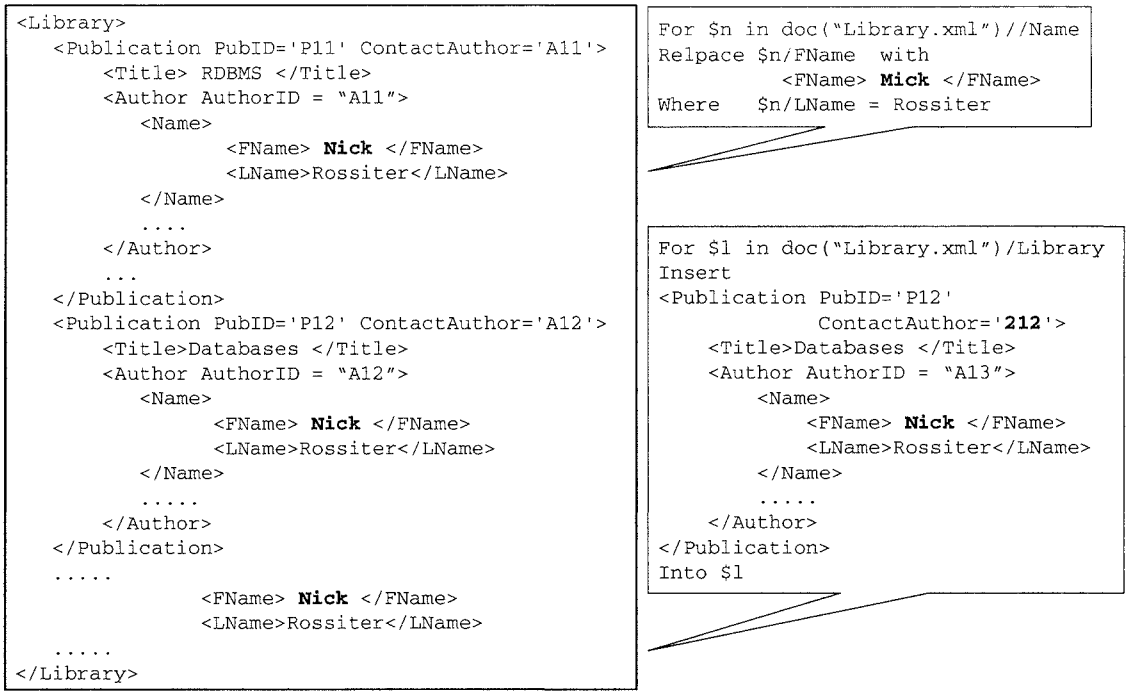

Figure 1.1 A single XML document storing redundant data and updating the data

From Figure 1.1, it is possible that one author can write several publications; thus the data of an author is kept redundantly. When updating the redundant data of an author by using a descendant path ('//'), it is necessary to search all possible paths and update every place that keeps the same data of the author; thus it takes more time in updating. To solve the data redundancy problem, one possible way is to keep non-redundant data in several separate documents; however no XML update language can join documents; thus the problem cannot be solved by this method at the moment. From the Figure, when a new publication with the out-of-date data of the author is inserted into the XML document, the data of the author will be inconsistent and the referential integrity constraint is violated (contactAuthor refers to non-existing author) since existing XML databases cannot check this constraint. Moreover when we want to update all direct and indirect references of a publication, we have to use several commands in a sequence to update the direct references first then update the indirect references in a chain until no more indirect references are found. This is because there is no XML update language that can update data whose structure is recursive.

The outline of our approach to solve these problems is presented in the next section.

1.3 Our Approach

Our approach is not just for updating semi-structured data such as XML documents but for solving the five problems addressed in section 1.2 as well.

According to existing literature, updating XML has not been investigated as it should have been. XML updating has been relatively well-researched in the area of native XML databases whereas in the area of applying conventional databases to manage XML, only one work [124] has presented an XML update language for updating XML data. This work employed an RDB but only the syntax and semantics of the language are presented. In our solution, the ORDB technology providing the newer object features is exploited to update XML documents.

The purpose of using a traditional database, ORDB, in this research is different from that of other work. The previous work uses an OODB [2, 146], RDB [40, 74, 118] or ORDB [75, 96, 108, 127] as a database of XML documents to store and query XML data; thus the order of XML elements in the database must be maintained. On the other hand, our approach uses an ORDB as a mechanism to preserve XML constraints during updating and to indicate the target-elements in XML documents which should be updated. Therefore when updates are performed on XML documents it is not necessary to maintain the order of XML elements in the ORDB and users can query data from XML documents instead of the ORDB.

In our solution, we update XML documents via ORDB and let the database engine handle the preservation of constraints. Both the structure and constraints of XML documents are mapped to an ORDB. Instead of XML Scheme, DTD is used in this research since presently many XML documents still stick to DTD and DTD is widely supported by many tools; furthermore, DTD uses a very compact syntax.

To solve the problem of data redundancy, non-redundant data is kept in multiple separate XML documents so avoiding the storage of redundant data in one single XML document; then these separate documents are linked together by a mechanism called 'rlink'. This mechanism is mapped to the ORDB. To update XML data, an XML update language, as an extension to XQuery, is proposed and this language will be translated into SQL to update XML data stored in the ORDB. Then the changes in the ORDB will be propagated to the XML documents.

In our approach, users update XML data on a view of XML documents by using our XML update language. The processes performed on an ORDB are hidden; thus to the users it appears that they are operating on XML documents rather than the ORDB.

By applying an ORDB to update XML, the problems can be solved as follows. Firstly, the preservation of XML constraints is pushed to an ORDB engine. Secondly, data inconsistency and low performance caused by data redundancy are solved by storing non-redundant data in multiple linked XML documents. Thirdly, joins of XML documents in XML update commands are translated into joins of table in SQL commands. Fourthly, the path in a regular path expression is determined by using information derived from mapping XML to ORDB; thus fields or tables involved in the path can be tackled in a short time. Finally, the recursive function is translated into a series of SQL commands equipped with a programming capability.

1.4 The research question and hypotheses

The research question of this research is derived from problems found in existing literature. Thus our research question is as follows:

Can we use a conventional database such as an ORDB to update XML documents to solve the problems as stated in section 1.2 ?

The research question is used to develop the hypotheses tested by the experimental study. Thus the hypotheses focus on the outcomes of the research, both on the conceptual design of a technique for updating XML documents and on the performance of systems built using the technique:

1. That a rigorous technique for updating XML documents with an ORDB can be designed.
2. That such a technique yields an adequate performance, in comparison with other methods for updating XML documents.

These hypotheses are tested by the design developed in Chapters 3-5 and the implementation and experimental study described in Chapters 6-7.

1.5 Contributions

The aim of this research is to devise a solution for updating XML as the representation of semi-structured data to solve the problems stated in section 1.2. This aim leads to major contributions as follows:

- A review of existing semi-structured and XML data management techniques. The capabilities and limitations for XML mapping techniques, XML query and update languages and XML query language translation techniques are analysed to determine unresolved problems.
- An approach to update XML documents via ORDB. This solution helps to answer the unresolved problems addressed in section 1.2.
- A prototype to realise the proposed techniques. This prototype helps to validate the approach and provides the fundamental basis for conducting experimental study.
- A series of experimental results. These are used to indicate the performance of the prototype in different cache states, to indicate the effect of data redundancy, to indicate the performance among update features and among update operations, and to demonstrate the correctness of translating the XML update language into SQL.

Besides the major contributions, the minor contributions stemming from this thesis are as follows:

- Updating XML data through ORDB not only makes an advance in the XML research area but also responds to the requirement of business. In business applications, XML can be used as a database; thus updating XML data is necessary.
- A standard for updating XML documents has not been proposed by W3C and the existing XML databases and XML update languages have limitations in their capability for updating data. Thus, in our technique, the ORDB technology is exploited to increase the capability of existing XML update approaches in the aspect of controlling constraints during updating XML data, making it easier to join XML documents in updating and improving the performance of the updates by using regular path expressions.

- The ability to join XML documents makes it possible to store non-redundant data in multiple XML documents and thus data inconsistency will not occur when XML documents are modified.

1.6 Outline of the remainder of the thesis

The remainder of the thesis is focused on the solution for updating XML data via ORDB. The diagrammatic overview of the thesis structure is shown in Figure 1.2.

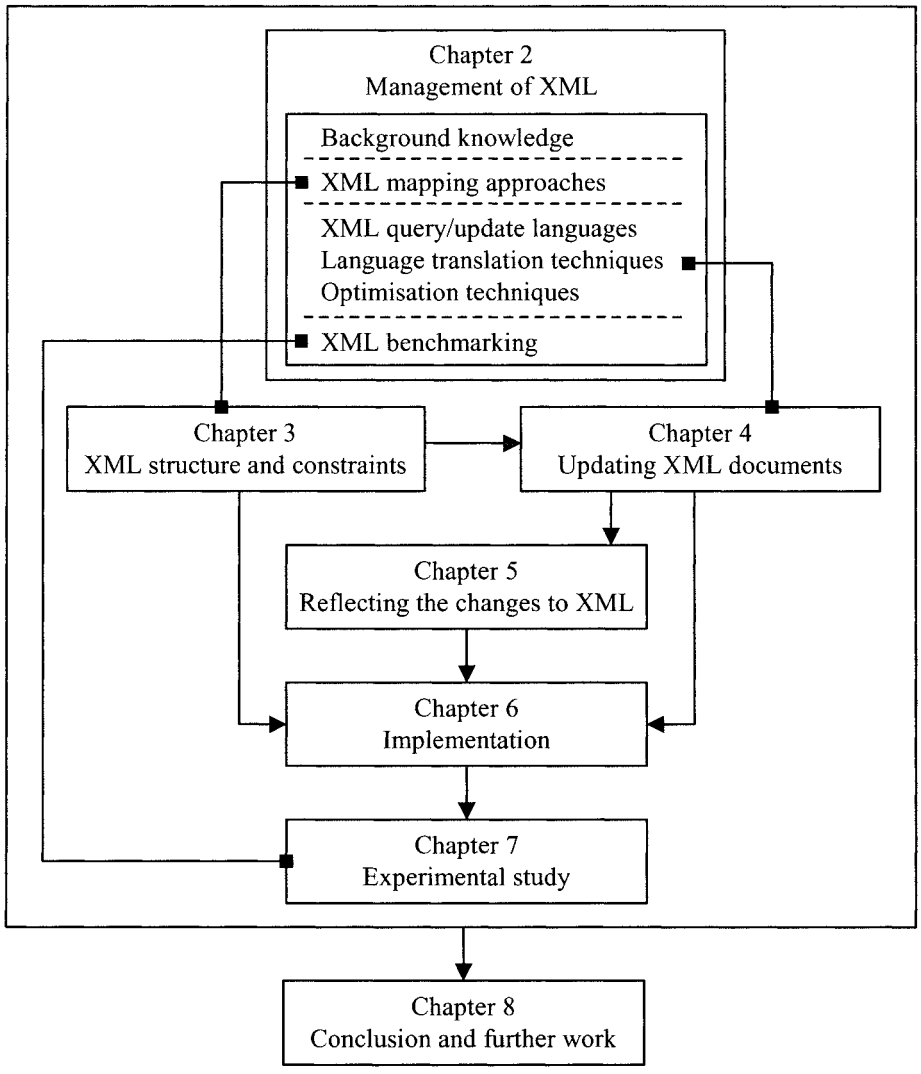

Figure 1.2 Overview of the thesis structure

In Chapter 2, the literature related to the XML update solution in this thesis is explored. The literature covers the following issues: (a) techniques for mapping XML to a conventional database, (b) XML query languages and XML update languages, (c) techniques for translating XML query languages into SQL, (d) optimisation techniques

in querying XML documents, and (e) techniques for benchmarking XML database. Overviews of semi-structured data and XML are also provided in Chapter 2.

Issue (a) is used as information to choose the mapping method designed in Chapter 3 and to show some open problems of existing mapping methods that will be solved. Issues (b) & (c) show some drawbacks of existing XML update languages and XML query language translations that will be improved in Chapter 4 while issue (d) is used as a guidance for devising optimisation techniques. Issue (e) is applied for benchmarking the XML update processing in Chapter 7.

In Chapter 3, the constraints of XML are categorised according to the corresponding constraints in an (O)RDB. A method for mapping the structure and constraints of linked XML documents to the structure and constraints of an ORDB is presented respectively.

Chapter 4 presents the design of an XML update language and the techniques for translating this language into SQL for updating XML data stored in an ORDB according to the mapping method presented in Chapter 3.

Chapter 5 presents a technique for propagating the changes in an ORDB to XML documents. The change in the ORDB is caused by the execution of SQL translated from the XML update language as described in Chapter 4.

Chapter 6 explains how the prototype has been developed. The tools used in creating the prototype are discussed and the architecture of the prototype is depicted. The architecture of the prototype is developed from the design in Chapters 3-5. Then the components of the prototype and their functions are explained and how the components communicate to each other is clarified.

Chapter 7 presents how the experimental study is conducted to assess the performance of the prototype. The correctness of translating the XML update language into SQL is checked by comparing the results of translating the XML update language obtained via the prototype and the expected results.

The conclusion of the thesis and some possible further work are presented in Chapter 8.

Chapter 2: Management of XML

Chapter 1 gave the overview of our solution to the update of XML documents in section 1.3: this chapter describes the literature related to the solution presented in this thesis including an overview of semi-structured data and XML management systems. In a similar way to the general steps for querying XML by using traditional databases, the general steps for updating XML via conventional databases are: (i) mapping the XML schema to the database schema and (ii) translating an XML update language into SQL executed on the database. This thesis describes a technique for updating XML via ORDB; hence related literature includes mapping XML to conventional databases, the languages for querying and updating XML, the translation of the languages into SQL, optimisation techniques in querying data and benchmarks for XML database. This chapter is organised as follows.

- Section 2.1 provides an introduction to semi-structured data;
- Section 2.2 gives an overview of XML management systems;
- Section 2.3 investigates existing techniques/approaches for mapping XML documents to conventional databases;
- Section 2.4 presents a review of existing XML query languages and XML update languages;
- Section 2.5 provides a review of techniques/approaches for translating XML query languages into SQL;
- Section 2.6 provides a review of techniques used for optimisation in querying XML documents;
- Section 2.7 describes techniques for benchmarking XML databases that will be adapted to our experimental study;
- Section 2.8 provides a summary of the chapter and points to what follows next in the thesis.

2.1 Introduction to semi-structured data

The large volume of data on the Web is growing continuously. The forms of Web data range from unstructured data such as image or sound to highly structured data such as sales data and invoice data. This diverse information proliferates over the Internet; hence data integration becomes indispensable, and this motivates the research area of semi-structured data because the semi-structured data model provides a highly flexible data structure that may be used to capture most types of data.

There are several major characteristics of semi-structured data [1] as follows. Firstly, semi-structured data has an irregular structure and the data is not strongly typed: the values of the same elements may be represented in different types. Secondly, the schema of semi-structured data may or may not be defined and sometimes the schema may be larger than its data since the schema must contain enough data formats for the data coming from various heterogeneous sources. Thirdly, its schema is implicit because it needs a parsing mechanism to isolate pieces of data and detect relationships between them. Finally, the schema of data can change.

Research on semi-structured data focuses on extending database management techniques to handle data having an irregular or unknown structure. Most research concentrates on data models or query languages for semi-structured data.

2.1.1 Data model for semi-structured data

The unifying idea for modelling semi-structured data is representing the data by a graph-like or tree-like structure. OEM (Object Exchange Model) [123] is the effective standard model for semi-structured data introduced for the Tsimmis [101] data integration project. OEM (based on the notion of objects) is a self-describing, nested object model. Data represented in OEM can be thought of as a labeled, directed graph, with objects representing elements as the vertices and labels on the edges representing the relationship between elements and sub-elements. In OEM, each object can either be atomic or complex. Every object has an object identifier (OID). An atomic object contains a value which is of a base type such as integer, string or real. A complex object contains a value which is a set of [label, sub-object] pairs. The label provides a textual description of the relationship between the complex object and its sub-object. In

other words, OEM is a graph whose nodes are the objects and the edges are labeled with the relationship between object and its sub-objects. Leaf nodes have associated atomic values as shown in Figure 2.1. From Figure 2.1, the root node '1' contains two sub-objects (nodes '2' and '3') that are 'Restaurants'.

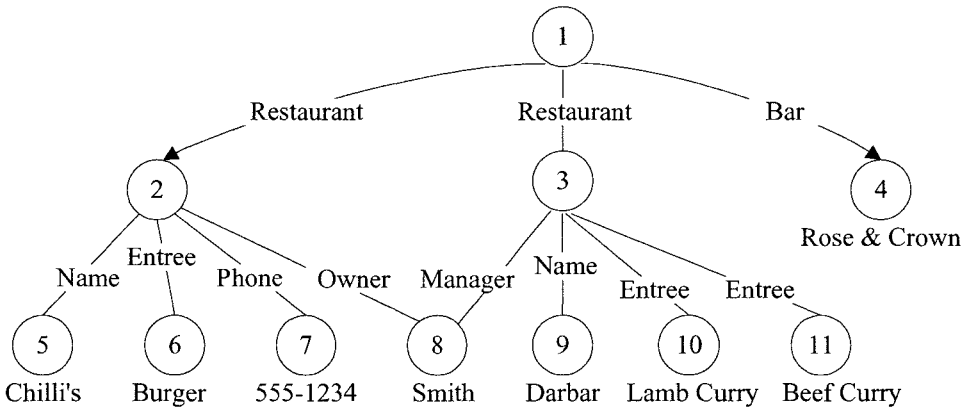

Figure 2.1 OEM of a database

2.1.2 Query languages for semi-structured data

Several query languages for semi-structured data have been proposed. The most distinct query languages are Lorel designed to query semi-structured data in the Lore system [89] and WebSQL [92] designed to query hypertext structures. Lorel [4] is OQL-like while WebSQL [92] is SQL-like. These two languages use OQL and SQL as a starting point and then insert some particular features appropriate for semi-structured data.

Lorel is an object-oriented query language [110]. An example for a Lorel command querying the titles of articles written in 2001 is as follows:

```
select A.title
from Publication.article A
where A.year = 2001
```

The query binds the variable A to an object in the *from* clause, tests the condition in the *where* clause and returns objects in the *select* clause. In Lorel, the *from* clause is optional. When the full path is specified, the *from* clause can be omitted. For example, the above can be rewritten as follows:

```
select Publication.article.title
where Publication.article.year = 2001
```

WebSQL is designed to query HTML. For example, if we want to find all HTML documents about "hypertext" in Web, a query will be written as follows:

```
SELECT d.url, d.title, d.length
FROM Document d
SUCH THAT d MENTIONS "hypertext"
WHERE d.type = "text/html";
```

The *FROM* clause is used to bind a variable to the *Document*, the condition based on the attribute of the *Document* is specified in the *WHERE* clause and the condition based on the content of the *Document* is specified in the *MENTIONS* clause.

2.1.3 DataGuide

Generally, semi-structured data has no predefined schema; thus queries may be evaluated inefficiently and it is hard to optimise querying. In fact, in the application of semi-structured data, frequently it is found that data has some regular structure [123]; thus some researchers exploit this regularity by specifying rigid schemas for the semi-structured data in advance. One most notable schema for semi-structured data is DataGuide [52]. The DataGuide is a graph where every label path in the database occurs exactly once in the DataGuide and all label paths in the DataGuide exist in the original database. The DataGuide is the summarisation of the database which is dynamically generated and maintained to always represent the current state of the database. The DataGuide is used in practice to browse structure and guide query formulation through a graphical interface for the Lore system. Figure 2.2 shows the DataGuide for the database in Figure 2.1.

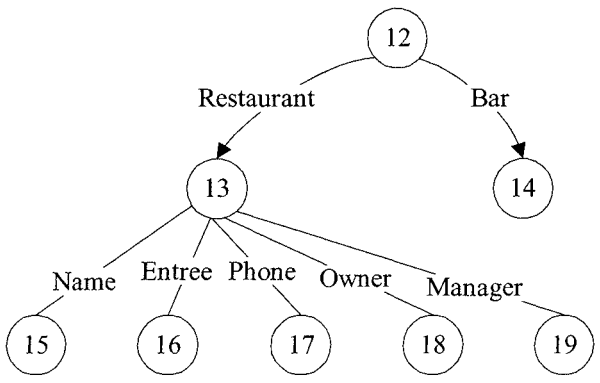

Figure 2.2 DataGuide for the database in Figure 2.1

Research in the area of semi-structured data is motivated by the need for data integration. However, with the advent of XML, a universal format for data interchange on the Web, much research currently focuses on managing XML as the standard representation of semi-structured data over the Internet.

2.2 XML management systems

Presently, there are two dominant approaches to manage XML repositories. The first approach is to design the specific native XML database. This approach can provide a more natural data model; nevertheless, a complete data management system must be built from scratch and searching data in this model must start from its root.

The second approach is applying conventional databases to manage XML data. One disadvantage of the approach is that it has a high cost of data conversion; however, it exploits maturity, stability, and scalability of traditional databases. This approach can be categorised into three types: schema-based method, schema-oblivious method and mixed method combining Shred and Edge methods together. The schema-based method is classified into Shred and Node methods whereas the schema-oblivious is classified into Edge and Node methods. The classification of systems for managing XML documents is shown in Figure 2.3. In the area of applying conventional databases, this figure expresses research that presents automatic mapping methods that are more complicated than those found in commercial (O)RDBs.

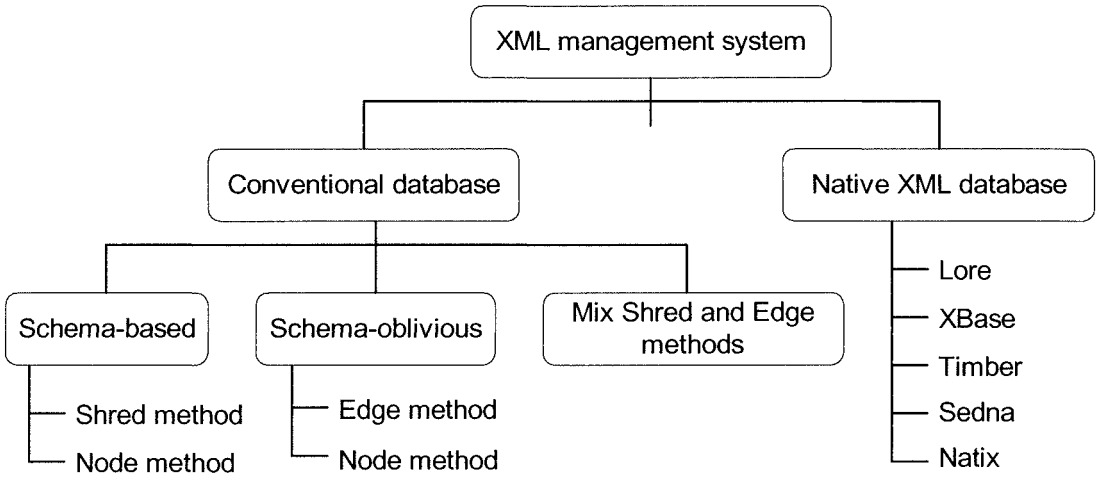

Figure 2.3 Systems for managing XML documents

2.2.1 Native XML database

A native XML database means a database designed specifically for the storage and retrieval of XML documents. The term 'native XML database' is used to contrast such a system with a database that merely provides an XML interface to data whose intrinsic data model is something different.

A common data model for XML documents kept in the native XML database is Document Object Model (DOM) [133]. In a DOM tree, each node has four filial pointers and two sibling pointers. The four filial pointers are the pointers linking to the first child, the last child, the parent and the root nodes. The two sibling pointers link to the previous and the next sibling nodes. To store a DOM tree physically, the nodes in the tree are serialised into disk pages by filial clustering: depth-first order or sibling clustering: breadth-first order. Lore [89, 90] and XBase [82] store XML documents in this manner whereas Timber [65] transforms each node of the tree into an internal representation and stores it into SHORE, a persistent object system [23]. For Sedna [55], a descriptive schema is dynamically generated from the XML data, that is paths of the document are kept in a descriptive schema.

Natix [45] keeps XML documents in the native storage format as follows. (Sub)trees of the original XML documents are kept together in a single (physical) page and the inner structures of (sub)trees are retained. If a document loaded into the database does not fit in a single page, it is split at an edge in the parse tree, turning it into two linked sub-documents.

Commonly, the data model of a native XML database is represented by a hierarchical graph or tree; thus the hierarchical nature of XML locks users into a single view of the data while most users always ask to see their data in many different ways. Although XSLT (Extensible Stylesheet Language Transformations) provides transformation, it cannot join data from multiple documents in a flexible manner while XQuery can join several documents but it cannot relate documents efficiently when parts of documents are required in a form different from that in which they are stored.

2.2.2 Conventional database applications

When compared with the native XML database, a relational database was designed to support large amounts of interrelated data; thus it can produce an almost infinite number of views of data. Relational databases have been researched for more than 35 years; therefore the mechanisms for preserving constraints, security, concurrency control and so on are mature and thereby it looks promising to handle XML by exploiting the relational technology.

Currently, besides relational database (RDB) and object-relational database (ORDB), the object-oriented database (OODB) is applied to handle XML documents too because the object model can hold the hierarchical structure of XML; thus it seems that XML also fits well in an object-based model.

Our solution uses ORDB since ORDB inherits relational mechanisms such as constraints and update statements from RDB and inherits OO features from OODB.

When XML data is kept in a conventional database (RDB, ORDB and OODB) then a database schema must be defined. The database schema can be generated either with or without the use of an XML schema. If the database schema is generated by using an XML schema, it is called a schema-based method; on the other hand if the database schema is generated without an XML schema, it is called a schema-oblivious method.

Schema-based method: There are two approaches for this method: Shred and Node methods. With the Shred method, XML data is shredded into several tables derived from mapping XML schemas to database schemas. The general approach of mapping the schema is as follows. The complex elements are converted to tables while simple elements and attributes are converted to fields. The parent-child relationships, ID and IDREF(s) are converted to associations between primary keys and foreign keys. For the general approach for the Node method, nodes are stored in tables according to types of nodes such as elements and attributes. Paths are kept in another table. Each table has a path-id which is used to join the tables. The positions of nodes are used to indicate a parent-child relationship

Schema-oblivious method: This method is classified into Edge and Node methods. In the Edge method, an edge is identified by OIDs of two nodes identifying the parent-child relationship. One table will contain pairs of nodes whereas the values of nodes

and each type of nodes are kept in further tables. The OIDs of nodes are used to join the tables. The Node method has already been introduced in the Schema-based method.

Applying conventional databases to manage XML documents is quite pertinent to our research; thus the techniques for mapping XML schemas to database schemas will be described in detail in section 2.3.

2.2.3 Managing XML in commercial (O)RDBMS

Commercial (O)RDBMSs provide new database types, non-automatic mapping and/or automatic mapping for simple XML structures to store XML documents in the databases as follows.

Oracle 9i [100] stores XML documents in a BLOB/CLOB (Binary/Character Large Object) column, in XML type or by shredding XML documents to store them in tables. In BLOB/CLOB representation, a column in a database table contains a byte stream or a literal string of XML data; thus non-schema XML documents can be stored by this approach. However, the documents must be searched using a text search engine in contrast to SQL queries which can facilitate the functionality of query rewrites, indexes, and so on. Updating a document is performed by downloading the entire file, making the change, and replacing it. Moreover, the storage can consume considerable space since the names of all elements must be stored in the table.

The main purpose of the XML type is to encapsulate the CLOB storage and to provide XPath-based methods for ease of navigating XML documents. XMLType [99] provides methods for retrieving XML data. In the case of shredding XML documents, it can be performed automatically only if XML documents conform to a canonical mapping. The canonical mapping allows for only a simple XML structure having limited nesting level, which is a parent of elements, elements and attributes of elements.

DB2 XML extender [61] stores XML data in XML columns or XML collections. In XML columns, XML data is kept as `varchar` type for small documents and *CLOB* type for large documents. For XML collections, documents are broken down into relational tables. Both XML columns and XML collections use DAD (Data Access Definition) files to define mapping between XML elements/attributes and relational columns.

Microsoft SQL Server [94] uses a language called OpenXML to break down XML data into tables. OpenXML can be used for inserts, updates and `SELECT INTO` target tables.

To conclude, the commercial databases provide new database types keeping intact XML data, a non-automatic mapping requiring users to specify how to map XML and/or an automatic mapping supporting only a simple XML structure. Proprietary languages mixing SQL, XPath and some API methods are used to query or update XML data.

2.3 Storing XML data in conventional databases

There are three main approaches for generating traditional database schemas from XML documents and/or XML schemas: schema-based approach, schema-oblivious approach and mixed approach combining Shred and Edge methods. In these approaches, there are several techniques for mapping XML to a conventional database. The comparison of mapping XML to traditional database models is shown in Table 2.1 (page 29).

2.3.1 Schema-based approach

The schema-based approach is categorised into two sub-methods: Shred and Node methods. In this approach, all work uses the Shred method except Khan *et al* [73] who used the Node method. Thus the approach is categorised according to the database models instead of the sub-methods.

RDB: Shanmugasundaram *et al* [118] utilised a DTD graph to represent a DTD for generating a relational schema. The graph's nodes are elements, attributes and operators in the DTD. Each element appears exactly once in the graph while attributes and operators appear as many times as they appear in the DTD. Relations are created for root elements and elements having occurrence '*'. Attributes in the relation are named by the path from the root element of the relation. Each relation has an ID field that serves as the primary key while a foreign key is created from a path linking between the two relations. A rule is applied to simplify the DTD version conflicting with a constraint: the symbol "+" is converted to the symbol "*". Shanmugasundaram

et al can reduce the use of unnecessary tables that usually occurs when mapping XML to RDB. For example, a name consisting of fname and lname is mapped to the fields: name.firstname and name.lname. Unlike with the work of Lv *et al* [85], Shanmugasundaram *et al* destroy the semantics of constraints.

Zhou *et al* [145] adopted the technique proposed by Shanmugasundaram *et al* [118] for mapping schemas, thus both share the same strengths and weaknesses. However, in addition Zhou *et al* created a path directory table to keep the distinct paths of the document. The table is used to quickly locate all element nodes with the same path.

Varlamis *et al* [130] used an XML-Schema file to indicate how to map an XML structure to an RDB schema. While other researches such as [85, 118] support the mapping methods that are more complicated than those found in commercial (O)RDBs, the approach in this work is similar to that in Oracle 9i [100] supporting only a simple XML structure. The XML structure consists of only simple elements, (group of) attributes and complex elements containing the simple elements or the (group of) attributes. Varlamis *et al* preserved constraints derived from a parent-child relationship and IDREFs; nevertheless, they defined 'Delete Set Null' on a foreign key converted from IDREFs; thus this constraint will conflict in the case that IDREFs is REQUIRED.

Psaila [106] modeled an XML document in the manner of an Entity Relationship Diagram (ERD) for a relational database. In the model, an entity describes a complex element and a relationship describes correlations existing between entities. Attributes of XML elements correspond to attributes of entities. However, some problems are not identified, such as handling multiple-valued elements and recursion. Mapping XML to ERD makes it impossible to define the order of elements and constraints except the cardinality constraint whereas for the general approach presented by other work such as [7, 85], the order of elements and constraints can be specified by adding an order-column and mapping the constraints to the database constraints. However Psaila's method can define the cardinality constraint while the general approach cannot.

The method presented by Bohannon *et al* [15] is different from other mapping techniques in the case that they used cost-based methods to determine which mapping method is the best one. This is because one XML Schema can be mapped to several RDB schemas. For example, if a child element has occurrence * and both parent and child are converted to tables, then the PK (Primary Key) of the parent can be kept as an

FK (Foreign Key) of the child or it is possible that a separate table will be created to keep the PKs of the parent and the child together.

Khan *et al* [73] used both DTD and XML documents for mapping. They kept elements and paths of elements in one table and keep attributes and paths of attributes in another table. The reference between elements is represented by paths kept in the tables. This method uses DTD to check the conformity of XML documents whereas other work based on the schema-based approach uses DTD for mapping an XML structure to a databases structure. Thus optimisation cannot be performed for this method while other work can optimise queries by reducing the joins of tables.

ORDB: Runapongsa and Patel [108] employed XML data type of object-relational database to store a fragment (sub-tree) of an XML document. Nevertheless, if only a small part of a document is updated, the whole fragment of the document has to be updated and users have to know the structure of the sub-tree stored in XML data type. As in [118], the researchers convert the symbol "+" to the symbol "*" conflicting with a constraint. While most work applying an ORDB uses nested tables or the collection type, this work uses the XML data type storing a literal string of XML data; thus this work cannot use SQL to query data.

Klettke and Meyer [75] and Mo and Ling [96] showed that an XML hierarchy can be fitted well into an ORDB structure. They exploited a set/list-collection type and nested tables which are features of an ORDB in mapping but there is no any DBMS which supports both set/list and nested tables in one ORDB. Informix [62] supports list/set and Oracle [100] supports nested tables while PostgreSQL [105] supports array. Moreover, Klettke and Meyer [75] used several constraints such as domain and default constraints in their collection type whereas Mo and Ling [96] defined foreign keys in nested tables. Defining the full range of constraints in nested tables or a collection type is restricted in available object-relational DBMSs; thus indeed not all constraints can be enforced.

Pardede *et al* [102, 103] created a row type in Array and multiset; however, in the current ORDB technology, only a simple type can be defined in these types.

Tseng and Hwung [127] mapped an XML DTD to an object-relational schema; however, they assumed that XML structure is regular and can map one-to-one from XML element to table. Tseng and Hwung mapped every element to tables; thus it

produces many tables while other works [75] [96] [102, 103] tried to reduce the number of unnecessary tables. Tseng and Hwung solved this problem by using *classes* for implementing a prototype in UniSQL/X [129] opposed to other popular ORDBs such as Oracle or DB2.

Han *et al* [57] presented a mapping language based on the XML Schema language to map an XML structure to an OODB/ORDB schemas. The rules of this language allow four kinds of mapping: element to class, element to column, attribute to column, and relationship between two elements to relationship between two classes. However, users must specify which element will be mapped to a class or column in this language. Thus it is different from other work presenting automatic mapping techniques.

OODB: Abiteboul *et al* [2] mapped semi-structured data to an object model. Each element in the DTD is interpreted as a class, choice connector (|) is modelled by a union type, element components marked by "+" or "*" occurrence indicators are represented by lists and attributes are represented by private property of the class. However, mapping semantics (constraints) of semi-structured data is not proposed because of the limited constraints of an OODB [110].

Zwol *et al* [146] modelled XML documents with MOA, an object algebra with the capability of new basetype extensions. Zwol *et al* concentrated on `ANY` type of XML documents. To cope with `ANY` Type, MOA merely inherits all the base types of the system; therefore a new type has all the capabilities of all the base types. However this solution misuses the original inheritance semantics and it increases the size of the class hierarchy. The fundamental mapping rules of MOA are based on the rules presented by Christophides *et al* [33]: elements in the DTD are identified as class definitions and the XML elements are regarded as objects of those classes.

Fegaras and Elmasri [42] presented a framework for storing XML data in an OODB and for translating XML queries called XML-OQL into OQL queries. While Abiteboul *et al* [2] and Zwol *et al* [146] supported only schema-based XML data, Fegaras and Elmasri proposed a new XML type for describing XML data capturing both schema-less and schema-based XML data. Schema-less XML data is stored in a tree-like form while the content of elements of schema-based XML data is stored in records. Fegaras and Elmasri implemented a prototype and conducted an experiment. The result showed that a schema-less query takes about four times longer than a schema-based query.

The drawbacks of applying an OODB are that most XML constraints cannot be handled by the database engine and updating XML is not convenient since OODB does not provide an update language. The strength is that an XML hierarchy can be fitted well into an OODB structure.

A drawback of applying an RDB is that it cannot handle a multi-value attribute and multi-value simple element efficiently since it has to create a separate table to keep this data. The strength is that XML constraints can be handled by the database engine.

An ORDB inherits features from an RDB and an OODB; thus the strength of applying an ORDB is in combining the strengths of an RDB and an OODB together. However some constraints cannot be defined on object types of an ORDB.

2.3.2 Schema-oblivious approach

An (O)RDB is used to handle XML data by the schema-oblivious approach which is classified into two sub-methods: Edge and Node. However, one piece of research [40] cannot be classified as either Node or Edge method. Deutsch *et al* [40] used OEM and a relational database to manage XML data. An RDB schema is extracted from XML documents by using a data mining algorithm to identify supported patterns and using OIDs of the OEM to be primary keys and foreign keys. An overflow graph is used to keep the data whose elements/attributes have never appeared in the XML documents.

In the schema-oblivious approach, only Shimura *et al* [119] used an ORDB to store XML data; thus this approach is categorised according to the two sub-methods.

Edge method: Florescu *et al* [47] broke down XML data into edges and kept them in the table Edge(*Source*, *Ordinal*, *Target*, *Label*, *Flag*) and table Value(*oid*, *value*). The meaning of the fields of the tables Edge and Value is shown in Figure 2.4.

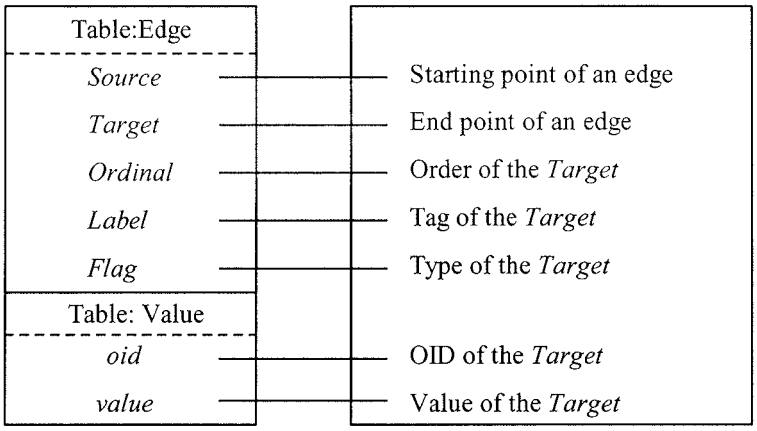

Figure 2.4 Fields of the tables Edge and Value and the meaning of the fields

For the table Edge, an edge is indicated by two nodes: the OIDs of *Source* and *Target* attributes. The *Label* attribute keeps the tag of the *Target* whereas the *Ordinal* attribute stores the order of the *Target* among all the child nodes of the *Source*. The *Flag* attribute specifies type of the *Target* such as int, string or ref (internal-node). For the table Value, it keeps the OID of *Target* and the *value* of the *Target*. The OIDs are used to reference between tables. Similarly, XParent [70] kept a parent-child relationship with node pairs: parent *(Source)* and child *(Target)* attributes in a table; however, XParent kept paths explicitly in another table instead of holding a tag as a *Label*.

Manolescu *et al* [87] shredded XML data into virtual generic schema tables keeping data based on node types such as element, attribute, namespace and value. In each table, the id of the node types is kept. Parent-id and child-id are kept in another table to keep a parent-child relationship and to join data in each table together.

XParent compares the path in a path expression with the path in the path table and then joins the path table to element and data tables. If there is no search condition, the number of joins of XParent is less than that of the method presented by Florescu *et al* [47]. However the number of joins of XParent is more than that of the method presented by Florescu *et al*, if there are several conditions in the search expression. This is because the path has to be split in XParent when a condition is found and all absolute paths (of each split path) compared with the path in the path table. Then the path table is joined with the DataPath table to indicate parent-child relationships and finally a further join is made with the element and data tables to retrieve data values or use the data values to compose conditions. Thus if there are several conditions, there are several split paths and the number of joins is increased.

In a similar way to the work by Florescu *et al*, the number of joins in the Edge method presented by Manolescu *et al* [87] depends on the length of the path in a path expression (Manolescu *et al* use the name of each element in the path to compare with the name of elements kept in the Element table). However Manolescu *et al* indicate the parent child relationship via the child table holding childID and parentID; thus there are more joins than the method of Florescu *et al* since the child table needs to be joined with the element and data tables.

Node method: Shimura *et al* [119] decomposed XML documents into the nodes and stored them in tables according to the node types. Thus XML documents are mapped to four tables: element, attribute, text, and path tables in an ORDB. Each table has a field called pathID used to reference each other whereas the positions of nodes kept in element, attribute, and text tables are used as implicit relationships between parent and child nodes. Actually, this method can use an RDB since it does not use any features of an ORDB. Yoshikawa *et al* [142] presented a very similar method to that of Shimura *et al* [119] except that the positions of nodes are kept separately as starting and end positions in different columns.

When compared with the node method presented by Khan *et al* [73] (page 23), the method of Shimura *et al* and Yoshikawa *et al* keeps the order of elements by using the position of elements, text and attributes; thus it can use `order by` in SQL whereas that of Khan *et al* [73] cannot. This is because Khan *et al* concatenate the order number of a path with the path-name; thus the order of the path is a lexicographical ordering instead of an integer ordering. The method of Khan *et al* consumes more space than that of Shimura *et al* and Yoshikawa *et al* since Khan *et al* keep all redundant paths in a table whereas Shimura *et al* and Yoshikawa *et al* keep only all unique paths. However only two tables are joined in SQL using the method of Khan *et al* while that of Shimura *et al* and Yoshikawa *et al* requires joins of more than two tables.

When comparing the Edge, Node and Shred methods, both Node and Edge methods have two weaknesses in common. The first weaknesses is that both Node and Edge methods consume more space than the Shred method. For instance Florescu *et al* [47] and Khan *et al* [73] both kept redundant element-names or paths. Although XParent [70] and Manolescu *et al* [87] also kept distinct paths or unique element names their data holds many IDs for linking tables and absolute paths; thus element-names occur redundantly in paths. The second weaknesses is that optimisation cannot be performed

and constraints cannot be preserved by the traditional database engine in the Edge and Node methods while, in the Shred method, optimisation can be performed to reduce joins between tables and XML constraints can be mapped to the database constraints.

2.3.3 Mixed approach

Amer-Yahia *et al* [7] combined the existing Shred and Edge methods. They proposed a non-automatic mapping, using the syntax of an XML Schema to specify a mapping from XML to an RDB and specifying choices of mapping such as mapping by Edge [47] or Shred [118] method. Both Edge and Shred methods can be used together in one document structure. They allowed an automatic mapping by using the technique of Shanmugasundaram *et al* [118]. This work used XPath to query XML data; however, it does not support updating or preserving constraints.

When querying XML data, mapping the XML by a schema-based approach outperforms mapping the XML by a schema-oblivious approach. This is indicated by several experimental studies. Florescu *et al* [46] evaluated a number of schema-oblivious methods and showed that the best overall schema-oblivious method with respect to performance is the attribute method categorised as the Node method (schema-oblivious method). Tian *et al* [126] identified that the attribute method has a poorer performance than the DTD method (schema-based method). Fegaras and Elmasri [42] and Tatarinov *et al* [125] indicated that a schema-based method yields a far better performance than a schema-oblivious method.

Table 2.1 Comparison of mapping XML documents to traditional database models

Research/Proposal	Model	Type of schema	Mapping method	Query Language	XML Update	Constraint Handling	Order kept	Recursion
Khan *et al* [73]	Relational	DTD	Node	XPath	No	No	No	No
Psaila [106]	Relational	DTD	Shred	None	No	No	No	No
Shanmugasundaram *et al* [118]	Relational	DTD	Shred	XML-QL Lorel	No	No	No	Yes
Zhou *et al* [145]	Relational	DTD	Shred	None	No	No	No	Yes
Lv *et al* [85]	Relational	DTD	Shred	None	No	Partial	No	No
Bohannon *et al* [15]	Relational	XML Schema	Shred	XQuery	No	No	No	Yes
Varlamis and Vazirgiannis [130]	Relational	XML Schema	Shred	DB. Command	No	Partial	No	No
Amer-Yahia *et al* [7]	Relational	XML Schema	Shred Edge	XPath	No	No	Yes	Yes
Florescu and Kossmann [47]	Relational	None	Edge	None	No	No	Yes	Yes
Jiang *et al* [70]	Relational	None	Edge	XPath	No	No	Yes	Yes
Manolescu *et al* [87]	Relational	None	Edge	QUILT	No	No	No	Yes
Yoshikawa *et al*[142]	Relational	None	Node	XPath	No	No	Yes	Yes
Deutsch *et al* [40]	Relational	None	Data mining	User defined language	Yes	No	Yes	Yes
Shimura *et al* [119]	OR	None	Node	XQL	No	No	Yes	Yes
Klettke and Meyer [75]	OR	DTD	Shred	None	No	Partial	No	Yes
Runapongsa and Patel [108]	OR	DTD	Shred	SQL	No	No	Yes	Yes
Tseng and Hwung [127]	OR	DTD	Shred	None	No	No	No	No
Mo and Ling [96]	OR	DTD, XML Schema	Shred	None	No	Partial	No	No
Pardede *et al* [102, 103]	OR	XML Schema	Shred	None	No	Partial	No	No
Han *et al* [57]	OO/OR	XML Schema	Shred	None	No	No	No	No
Abiteboul *et al* [2]	OO	DTD	Shred	OQL extension	No	No	No	No
Fegaras and Elmasri [42]	OO	DTD	Shred	XML-OQL	No	No	No	No
Zwol *et al* [146]	OO	DTD	Shred	Algebra	No	No	No	No

From Table 2.1, the two main problems of mapping XML are constraint handling and update features of XML query languages. Although, some work handles constraints in mapping, not all constraints are handled. Moreover, Klettke and Meyer [75] and Mo and Ling [96] defined constraints that are impracticable in the current technology. Furthermore Varlamis and Vazirgiannis [130] defined constraints conflicting with each

other: they defined 'Delete Set Null' on a foreign key converted from IDREFs; thus this constraint will conflict in the case that IDREFs is REQUIRED. Pardede *et al* [102, 103] presented only the constraint mapping a key in XML Schema to a primary key. Lv and Yan [85] preserved the cardinality constraint in the case of converting elements $(e_1, e_2)+$ to (e_1+, e_2+) by checking cardinality(e_1) = cardinality(e_2); nevertheless, if e_1 or e_2 is converted to a table and every row of e_1 and e_2 is deleted, this solution cannot check that it should have at least one row of e_1 or e_2 in the table. Furthermore, it cannot check the cardinality constraint in the case that parent and child elements are converted to tables. Until now, none of the existing work covers all the constraints that are found in DTDs.

Although Lee *et al* [78, 79] proposed mapping XML constraints to RDB constraints, some constraints are not mentioned. For example, preserving constraints of a choice of groups of elements and preserving constraints (such as delete cascading in the case of the parent-child relationship) during updating have not been addressed. For the cardinality constraint, in the case that two elements are converted to tables, instead of checking the cardinality of elements, the referential integrity constraint is checked instead. To summarise, although Lee et al try to handle XML constraints during the mapping of XML schemas to RDB schemas, some constraints are still not handled. Moreover the referential integrity constraint is interpreted imprecisely as the cardinality constraint. Lee et al implemented a prototype and conducted an experimental study. The experimental results showed that ID and IDREF(s) are seldom used in the DTDs whereas the occurrence indicator: '?', '*' and '+' is much used.

For the update feature, STORED [40] uses Lorel [4] as its update language; thus it supports updating the value of existing nodes but it does not support an insert operation.

In the cases of keeping order and of recursion, several works support these aspects. The order of elements is used for sorting the result derived from querying. The recursion involves mapping the recursive structure to a database schema which will be used in translating an XML query/update language into SQL.

2.4 Querying and updating XML documents

When compared to the lower-level APIs provided by DOM (Document Object Model) or SAX (Simple API for XML), a query language or update language is a more convenient way to navigate or update XML documents, especially for non-programmers. Most of these languages are declarative i.e. users specify what information they want to extract from XML documents whereas in functional languages and APIs, users specify how the information should be extracted algorithmically. In this section, XML query languages will be described first and then XML update languages will be described later.

2.4.1 XML query language

Many XML query languages have been proposed; however, only seven well-known XML query languages are compared namely Lorel [4], XML-QL [39], XML-GL [26], XSLT [36], XQL [107], Quilt [30] and XQuery [27, 28]. This is because these languages except XML-GL were implemented and used in the real world. Although, XML-GL has not been implemented, it is unique among other XML query languages since it provides a graphical interface. The comparison of these languages is provided in Table 2.2 where the information on these languages except XQuery is derived from [18, 19]. XPath is not included here since it is a navigational language designed to be used as a part of other XML standards such as XSLT, XPointer and XQuery.

The seven query languages can be classified into two categories: declarative languages and functional languages. Lorel, XML-QL, XML-GL, XSLT and XQL are declarative languages whereas Quilt and XQuery are functional languages. Normally, a declarative language is easier than a functional language to understand but it has less expressive power. However in the real world, which language is easy or difficult to use depends on the experience or background knowledge of users.

Table 2.2 Comparison of the seven XML Query Languages

Languages / Functionalities	Lorel	XML-QL	XML-GL	XSLT	XQL	Quilt	XQuery
Specific Data Model	✓	✓	✓	✓	✓	✓	✓
Joins	✓	✓	✓	✓	✓	✓	✓
Partially specified path expr.	✓	✓	Partial	✓	✓	✓	✓
Halt on matching cyclic data	✓	×	×	×	×	✓	✓
Existential quantification	✓	✓	✓	✓	✓	✓	✓
Universal quantification	✓	×	×	×	✓	✓	✓
Negation	✓	×	✓	✓	✓	✓	✓
Construct new elements	✓	✓	✓	✓	×	✓	✓
Support for aggregation	✓	×	✓	Partial	Partial	✓	✓
User defined function	×	×	×	✓	×	✓	✓
Nested queries	✓	✓	×	✓	✓	✓	✓
Set Operation	✓	Partial	✓	✓	✓	✓	✓
Sorting results	✓	✓	✓	✓	×	✓	✓
Querying the schema order	×	✓	×	×	×	×	×
Querying the instance order	✓	×	×	✓	✓	✓	✓
Support for type coercion	✓	×	×	×	Partial	×	Partial
Support environment	×	×	×	×	×	×	✓
Support XML Schema	×	×	×	×	×	✓	✓
Support of namespaces	×	✓	×	✓	✓	✓	✓
Tag variables	✓	✓	×	×	×	✓	✓
Update feature	✓	×	✓	×	×	×	×

To make an extension to an existing query language for our solution, the expressive power is a major factor which is considered next while the simplicity is considered later.

Expressive power: Expressive power means features that a language supports. Thus the expressive power of various XML query languages is ordered according to the number of features that they support.

Simplicity: Simplicity indicates how easy a language is to understand or learn. Here, the simplicity of the XML query languages is ordered according to our background knowledge. Of the seven languages, XQuery has the most expressive power but it is the most difficult to learn. Quilt is the second most difficult to understand since it uses syntax similar to XQuery while it provides less expressive power than XQuery. For the five declarative languages, XML-GL and Lorel are the two most easy to understand while XSLT is the most difficult to understand. XML-GL provides a graphical interface while Lorel is an OQL-like language. XQL is another language that is quite easy to

understand since its syntax is very similar to XPath. XML-QL seems to have a difficulty in understanding intermediate to that between XQL and XSLT; however, XML programmers may consider it quite easy to understand since the syntax of XML-QL is similar to XML.

The order of simplicity and expressive power is summarised as follows:

Ordering	1	2	3	4	5	6	7
Simplicity	XML-GL	Lorel	XQL	XML-QL	XSLT	QUILT	XQuery
Power	XML-GL	XML-QL	XQL	XSLT	Lorel	QUILT	XQuery

1 means easiest or least powerful and 7 means most difficult or most powerful.

XQuery is the most powerful but also the most difficult to use. Lorel is reasonably powerful and is also fairly simple to use.

2.4.2 XML update language

In this section, the capabilities of existing XML update languages are discussed. Some update languages are extensions to XQuery whereas others are XML query languages including update features in themselves.

Presently, there is no standard for an XML update language. Even though several languages have been proposed, their capabilities are quite limited. A comparison of XML update languages is shown in Table 2.3.

Table 2.3 Comparison of XML Update Languages

Features \ Language	XUpdate	SixDML	Lorel	XML-GL	XML-RL Update Language	XML Update Extension
Updating	elements	elements	nodes	nodes	elements	elements
Order Preserving	✓	✓	✓	×	✓	×
Recursion	×	×	×	×	✓	×
Joins of documents based on values	×	×	×	×	×	×
Joins of documents based on pointers	×	×	×	×	×	×
Update multiple linked documents	×	×	×	×	×	×
A sequence of updates	✓	✓	×	×	✓	✓
Implementation/prototype	✓	×	✓	×	×	×
Origin of Language	XPath	XQuery	Lorel	XML-GL	XML-RL	XQuery

XUpdate [140] proposed by the XMLDB group provides creation, insertion, deletion and value updating. Its query language is based on XPath to select elements for updating and for conditional processing. Except for order preserving and a sequence of updates, it does not support other capabilities specified in Table 2.3.

SiXDML [97], an extension of XQuery, provides functionalities to insert, delete elements and update value with the condition using the "IF/THEN/ELSE" expression. SiXDML uses the "FOR" expression for binding variables to elements or attributes. Again, it does not provide other capabilities apart from order preserving and a sequence of updates. This language has not been implemented.

Lorel [4] is the first XML query language supporting XML updating. It only supports creation and deletion operations, and value updating for existing nodes on a Lore data graph. Lorel does not provide an insert operator; hence if we want to insert any object, two steps must be taken: creating a new object and updating the existing graph with the new object. For instance, inserting an author element which is a sub-element of a book element requires two steps as follows:

```
name author := "Tomas"     // To create a new node
update book.author += author  // To update existing graph
```

Other capabilities except order preserving have never been mentioned.

XML-GL [26] only provides the functions for updating values of nodes and creating new nodes. It does not support any capability specified in Table 2.3. There is no evidence to show that XML-GL has been implemented.

XML-RL update language [83] is an extension to XML-RL [81]. For this XML update language, an XML document is viewed as a complex object model instead of a tree model. It can preserve the order of elements in updating, handle recursion and perform a sequence of updates. However, it lacks features for updating multiple linked documents and joins of documents in updates. There is no implementation for this language.

XML update extension [124] is another extension of XQuery proposed to update XML data kept in an RDB. A sequence of updates is supported; however, other features in Table 2.3 are ignored. There is no implementation for this language.

From Table 2.3, we see that all languages can perform updating only on a particular document. None of these update languages can join documents in update commands or

update multiple linked documents. Only XML-RL can update data whose structure is recursive but this language has never been implemented. Only a few XML update languages are implemented such as XUpdate and Lorel. The preservation of constraints is not included in XML update language features since usually it is handled by the DBMS, not the language; thus the constraint facility is not included in Table 2.3. Consequently, existing XML databases such as X-Hive [139], eXist [91] and Xindice [8] using XUpdate or other XML update languages to update XML documents, cannot preserve constraints during updating.

2.5 Translating XML query language into SQL

By applying an (O)RDB to manage XML documents, XML data must be converted to keep it in the database. To query XML data in an XML view, an XML query language must be translated into SQL to retrieve XML data stored in an (O)RDB [117]. Much work has been done applying an (O)RDB to XML; thus several techniques for translating diverse XML query languages into SQL have been presented. These techniques may be classified according to the method for representing XML to the database. There are three methods for representing XML to the database: by the shred approach, as a view created from the database and by the edge or node approach. A comparison for translating XML query languages into SQL is shown in Table 2.4 (page 39).

2.5.1 Translation based upon representing XML by the Shred method

Translation based upon representing XML to the database by the Shred method is in effect translating an XML query language into SQL to query XML data in an (O)RDB whose schema is derived from mapping XML to the (O)RDB by the Shred method.

Fong *et al* [48] translated XQL to SQL by converting the XQL query graph to the SQL query graph. The steps of the conversion are as follows. Firstly, decompose XQL by parsing its XPath according to the path of child (/) and descendent (//) and then create a query graph of XQL through the nodes' navigation paths. For each node, its elements can be mapped to corresponding tables; thus the corresponding query graph of SQL can

be created from the path of the joined tables. Finally, the SQL syntax is parsed, then the FROM clause is created from the tables in the SQL graph, the WHERE clause is created from the condition nodes in the XQL graph and the SELECT clause is created from the selected nodes in the XQL graph.

Shanmugasundaram *et al* [118] presented general rules for translating Lorel into SQL as follows. For queries with simple path expressions, converting the queries to SQL is performed by identifying relations from path expressions and joining these relations together. In the case of queries with recursive path expressions, the researchers suggest that the technique for converting a recursion can be separated into two steps: identify the fix-point in initialisation of recursion (the starting point for querying data) and identify the actual recursive path expression. Shanmugasundaram *et al* showed the result of translating a query with a recursive path expression into an SQL command whose syntax is based on DB2; however they did not show the exact steps for the translation.

Jain *et al* [67] presented an algorithm for translating XSLT programs into SQL queries and assumed the XML document to be unordered. The translation is performed as follows: the parser translates the XSLT program into a directed acyclic graph of templates (of XSLT). Each path of the graph is translated into a Query tree corresponding to an SQL query. Each Query tree is converted to an SQL query and finally every SQL query is combined together. Jain *et al* conducted the experiments for the translation. The results showed that overall, the algorithm generated efficient SQL queries in most cases. However some inefficient SQL queries were generated because of the semantic mismatch between XSLT and SQL in the case of GROUP-BY and the presence of a nested IN query.

Krishnamurthy *et al* [76] proposed an algorithm to generate an SQL command from a path expression query querying data whose structure is recursive. However, a path expression query has less power than a recursive function of XQuery since the recursive function of XQuery can be used along with other features such as a conditional expression whereas a path expression query cannot be used along with other features. However, Manolescu *et al* [86, 87] in translating Quilt and XQuery into SQL determined that the recursion is an untranslatable feature.

2.5.2 Translation based upon representing XML as a view

Sometimes the original data is stored in an (O)RDB but users may want to query data in an XML view; thus the XML view is created from the (O)RDB, users then use an XML query language to query the (O)RDB via this XML view and, in the background process, this language is translated into SQL to retrieve (O)RDB data.

In SilkRoute [43], XQuery is translated into SQL. The researchers use a view forest to represent XQuery to SQL translation. Semantically, a view forest defines a mapping from a relational database to an XML document. Usually a user has a canonical XML view created from a relational database. The canonical XML view is the XML view which is created from a relational database and which has never been materialised. To translate an XQuery expression, the XQuery expression querying the canonical XML view is rewritten as a view forest over the canonical XML view. Then each node in the view forest is associated with a unique identifier based on the Dewey encoding [98]. This encoding allows us to check the parent-child relationship easily. The SQL fragments are stored in each node of the view forest.

Fernandez *et al* [44] translates XML-QL querying over an XML view into SQL. XML-QL consists of patterns, tag-path conditions and constructors. First, the patterns are evaluated on the view definition to obtain a modified XML view, then the tag-path conditions and constructors are evaluated by using the modified view.

In XPeranto [116], XQuery is translated into SQL as follows. Firstly, an XQuery query is parsed and converted to an internal query representation called XML Query Graph Model (XQGM), an extension of a SQL internal query representation. Secondly, the query graph is composed with the XML views to which it refers. Thirdly, optimisations are performed by eliminating the construction of intermediate XML fragments and pushing down predicates. Finally, the modified XQGM is translated into a single SQL query. The experimental result shows that the translation time is always less than half a second.

2.5.3 Translation based upon representing XML by the Edge or Node method

Translation based upon representing XML to the database by the Edge or Node method is in effect translating an XML query language into SQL to query XML data in an (O)RDB whose schema is also derived from mapping XML to the (O)RDB by the Edge or Node method.

Storing XML by the Edge method: Jiang *et al* [70] translated XPath into SQL. Firstly, an XPath query is represented as a logical query plan. Then paths in the query plan are used as conditions for comparison with paths kept in the *Path* table. *PathID* is used to join the *Data* table to select values or to use values of elements/attributes as conditions. The *IDs* of parent and child kept in the *Edge table* are used to join other tables and to indicate the parent-child relationship.

Manolescu *et al* [87] proposed a method for translating Quilt queries into SQL queries. In order to translate queries, they are normalised into a form easy to translate into SQL by using predefined rules of the researchers. Then, the normalised queries are translated into SQL. In translation, names of elements/attributes in paths of the normalised queries are used as conditions by SQL for comparison with the name kept in the *Attributes/Elements* table. The conditions based on values in the normalised queries are converted to the conditions in SQL by comparing query values with the values kept in the *Value* table. Then the *IDs* of these tables are joined to *parentID* and *childID* of the *Child* table, keeping the parent-child relationship. However, the researchers indicate that recursion is an untranslatable feature since Quilt uses a recursive function to query data whose structure is recursive in the same way as in XQuery. Quilt has very similar constructs to XQuery; thus in Agora [86], the researchers used the same method to translate XQuery.

Storing XML by the Node method: In XREL [119], XQL is translated into SQL. In translation, paths in XQL are used as conditions to be compared with paths kept in the *Path* table. *PathID* is used to join the *Element/Attribute* table to select values or to use values of elements/attributes as conditions. The positions of elements/attributes are used as conditions to indicate the parent-child relationship. If XQL queries contain a path operator '//', each occurrence of '//' will be superseded with '%/' by using a LIKE

predicate in the WHERE clause of SQL because simple paths of XML documents are kept in the *Path* table and thus a LIKE predicate can be used to select all simple paths. Yoshikawa *et al* [142] uses the same method as Shimura *et al* [119] to translate XPath into SQL.

Khan *et al* [74] translated XPath into SQL. The translation starts from creating a condition in SQL by comparing paths in the XPath with paths kept in the *attribute/element* table to select values or to use values of elements/attributes as conditions in SQL. The *path* is used to join the *position* table to indicate the parent-child relationship.

Table 2.4 Comparison of techniques for translating XML query into SQL

Researchers	Recursion	Optimisation	Represent XML to Database	Language / expression	Database
Fong and Dillon [48]	N	N	shredding	XQL	Relational
Shanmugasundaram *et al* [118]	N	N	shredding	Lorel	Relational
Jain *et al* [67]	N	Y	shredding	XSLT	Relational
Krishnamurthy *et al* [76]	Y	N	shredding	Path expressions	Relational
Fernandez *et al* [43]	N	Y	XML V.	XQuery	Relational
Fernandez *et al* [44]	N	Y	XML V.	XML-QL	Relational
Shanmugasundaram *et al* [116]	N	Y	XML V.	XQuery	Relational
Khan and Rao [74]	N	N	Node A.	XPath	Relational
Shimura *et al* [119]	N	N	Node A.	XQL	OR
Yoshikawa *et al* [142]	N	N	Node A.	XPath	Relational
Manolescu *et al* [87]	N	N	Edge A	Quilt	Relational
Manolescu *et al* [86]	N	N	Edge A.	XQuery	Relational
Jiang *et al* [70]	N	N	Edge A.	XPath	Relational

From Table 2.4, there is only one work [76] that translates a path expression query in the presence of recursive structure in DTD into SQL. However, from the XQuery, no work translates the recursive feature into SQL. The reason for this will be specified in section 4.3 in Chapter 4.

2.6 Optimisation for querying XML documents

The objective of optimisation for XML databases is to perform queries against XML documents efficiently. Various optimisation techniques have been presented. Some techniques focus on reducing the search space in XML data whereas others concentrate on indexing data or the structure of XML documents. Several techniques are based upon optimisation by using algebra. Thus, in this thesis, optimisation techniques are classified into three categories: optimisation by eliminating unnecessary paths, optimisation by indexing and optimisation by algebra.

2.6.1 Optimisation by eliminating unnecessary paths

Eliminating paths means eliminating full paths or eliminating partial paths or nodes in full paths. Eliminating (partial) paths or nodes can be performed on the XML data model and XML structures.

Krishnamurthy *et al* [77] proposed two optimisation techniques based on storing XML data in a relational database: eliminating redundant prefixes and grouping multiple paths. Optimisation is performed during the translation of XML into SQL. Eliminating redundant prefixes involves the removal of the prefix of SQL at any node appearing more than one time in a schema graph. The prefix ranges from the root of the schema graph to the least common ancestor of the two nodes. Grouping multiple paths is the combining of two of the same sequences of nodes. To make it clearer, suppose that two tables: BOOK, and BOOK-AUTHOR are received from the graph in Figure 2.5. A cheap book has price < 50; otherwise it is a costly book.

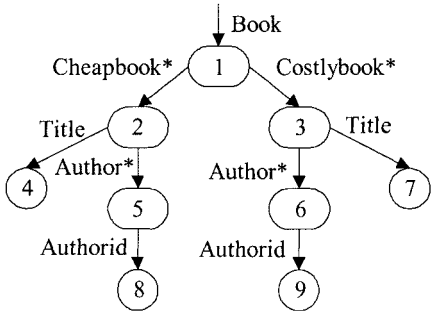

Figure 2.5 Schema graph

To query //Author/Authorid, a naïve translation will yield the following SQL query

```
Select BA.authorid
From BOOK B, BOOK-AUTHOR BA
Where B.bookid = BA.bookid  And B.price < 50
UNION
Select BA.authorid
From BOOK B, BOOK-AUTHOR BA
Where B.bookid = BA.bookid  And B.price >= 50
```

To eliminate the redundant prefix, BOOK is the least common ancestor; thus BOOK can be eliminated. To group the multiple paths, two of the same sequences of branch paths are BOOK-AUTHORs; thus BOOK-AUTHOR from the above two queries can be combined.

The optimised SQL query will be: `Select authorid From BOOK-AUTHOR.`

Wang *et al* [137] proposed two optimisation techniques for regular path expressions. The two techniques are path-shortening and path-complementing. Path-shortening reduces the number of joins by shortening path expressions. For example, from the Graph in Figure 2.6, 'description' which is the end node in the path `/site/closed_auctions/closed_auction/annotation/description`

can be reached by the path

`/site/open_auctions/open_auction/annotation/description`

as well. To shorten the path expression, the unique path is indicated first and then the long path is shortened. In this case, the unique path is `/site/closed_auctions/closed_auction` and thus the shorter path is `closed_auction/annotation/description`. Path-complementing is comparing the cost of an equivalent complementary path with the original path specified in a query. For example, the query "find all names of items of all regions" is expressed as `/site/regions/*/item/name`. From the graph, the name can be also reached by path `site/people/person/name`; thus path `/site/regions/*/item/name` can be evaluated by subtracting the result of the path `site/people/person/name` from path `/site//name`. Then the cost of the complementary path is compared with the original path. The path having the lower cost is chosen.

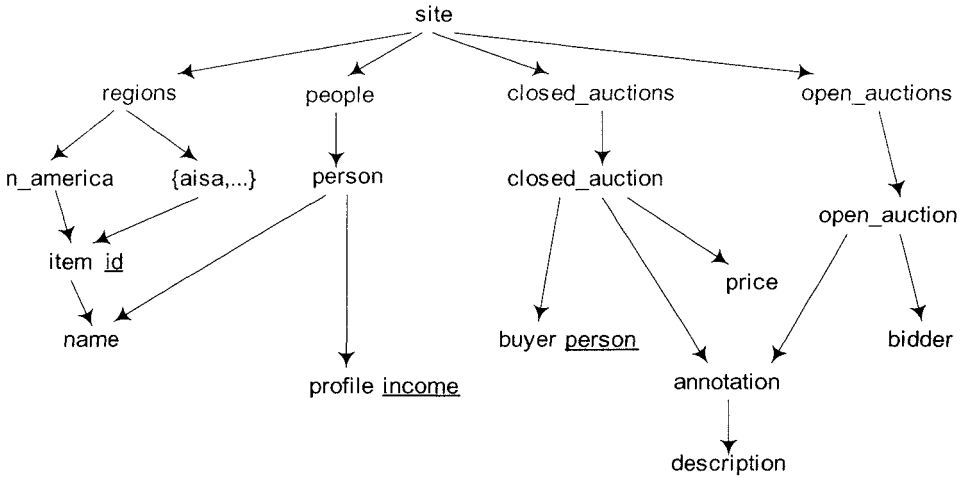

Figure 2.6 XML Schema Graph

Chung *et al* [34, 35] proposed two optimisation techniques: NodeInfo and MergeNodeInfo for querying XML documents based on Object Exchange Model (OEM). An example for OEM is shown in Figure 2.7.

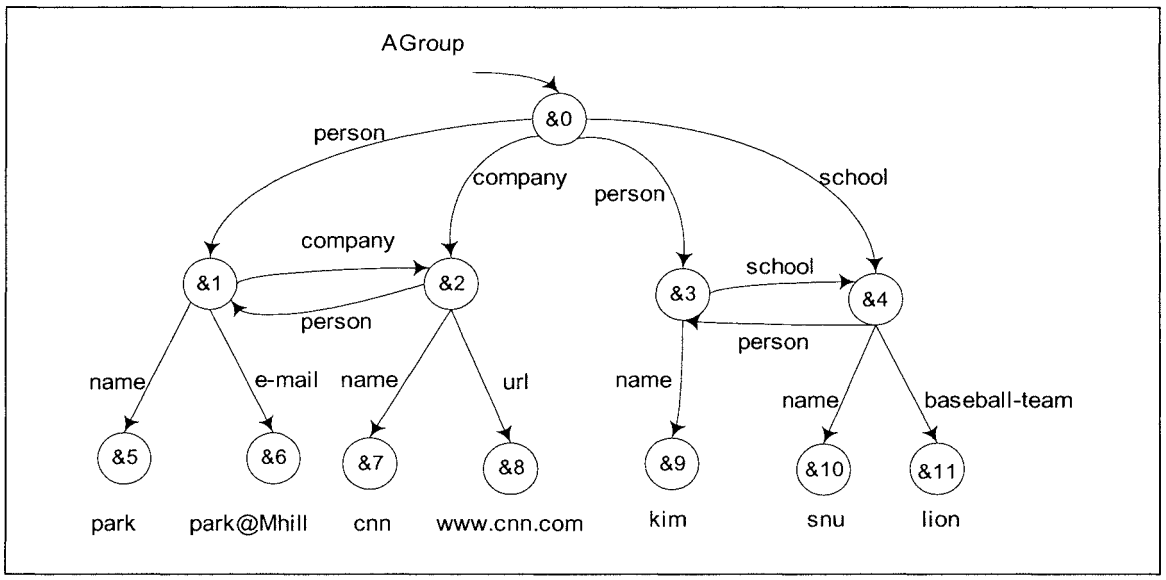

Figure 2.7 OEM Graph

NodeInfo and MergeNodeInfo are used to reduce the search space in an OEM graph. NodeInfo keeps information of all objects that are reachable directly from the target object. For example, for object &1 of OEM keeping NodeInfo 1:{e-mail, company}, when we want to query information of a person having an e-mail and working for a company, only objects that are children of object &1 will be searched. MergeNodeInfo is similar to NodeInfo. MergeNodeInfo keeps information on all objects that are reachable directly and indirectly from the target object (MergeNodeInfo is union of

NodeInfo of its descendants). For example, by using MergeNodeInfo, object &1 will keep {e-mail, company, url}.

McHugh and Widom [90] presented three methods for processing path expression queries: top-down, bottom-up and hybrid methods. The hybrid method can reduce the space for navigating the OEM database. For example, from an OEM database in Figure 2.8, a Lorel query is as follows:

```
select x from DBGroup.Member x Where exists y in x.Age: y<30
```

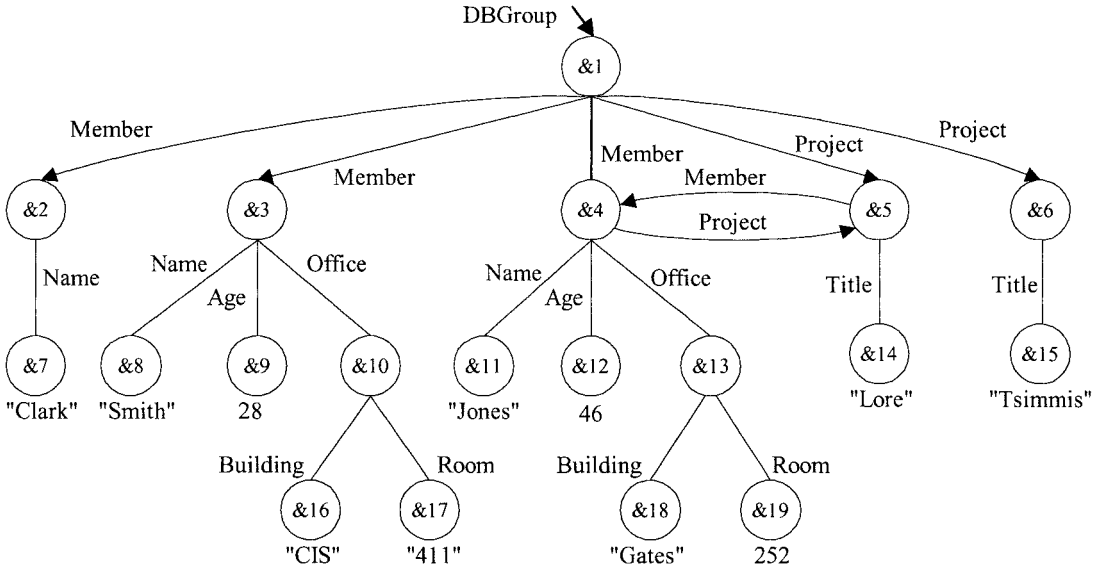

Figure 2.8 An OEM database

The top-down approach starts from DBGroup then fully explores all Member and Age to find a value for age which is less then 30. The bottom-up approach, firstly, identifies all objects that satisfy the "y<30" predicate and then traverses from children to parents. The advantage of the bottom-up approach is that it starts from an object guaranteed to satisfy the `Where` predicate and thus it does not need to explore paths through the data that do not satisfy the predicate.

However, the bottom-up approach is not always better than the top-down approach; for example when very few paths satisfy the path expression but a lot of objects satisfy the predicate. To solve this problem the hybrid approach, a combination between top-down and bottom-up, is proposed.

The steps of the hybrid approach are as follows. (i) Evaluate some but not all of a path expression top-down and create a temporary result of objects that satisfy the query

path. (ii) Identify all objects that satisfy the `Where` predicate and traverse up to the same point as the top-down exploration and create a temporary result of satisfying objects. (iii) Perform a join between the two temporary results.

From the example, firstly, only the paths from DBGroup to Member are explored by the top-down approach, and then the bottom-up approach is applied to find an Age satisfying the "y<30" predicate. Finally, the results from both approaches are joined together. Thus there is no need to explore paths from Member to Age where Age's value does not satisfy the predicate.

2.6.2 Optimisation by indexing

Similar to optimisation by eliminating unnecessary paths, both XML structure and XML data can be indexed.

Li *et al* [80] proposed a system for storing XML data based on a numbering scheme for elements and then proposed three index methods: element index, attribute index and structure index. The numbering scheme is defined as follows: for two given nodes x and y of a tree T, x is an ancestor of y if and only if order(x) < order(y) <= order(x) + size(x). Size(x) can be an arbitrary integer larger than the total number of the current descendents of x. The difference between size(x) and the total number of the current descendents of x is termed the preserve size. The preserve size indicates the number of new nodes that can be inserted as descendents of x. This numbering scheme associates each node with a pair of numbers <order, size>; however, a disadvantage of this scheme is that when the number of new nodes which are inserted is more than the number for preserve size, the current size of nodes need to be modified. An example of this numbering scheme is shown in Figure 2.9.

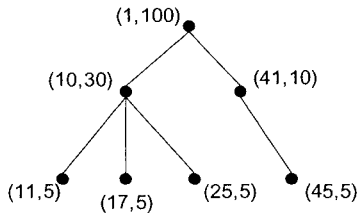

Figure 2.9 Numbering scheme using <order, size> pair

The three index approaches are used along with the numbering scheme for storing XML data as follows. Element index and attribute index use the name of an element (or attribute) as the key. Each entry points to a set of fixed-length records for elements (or attributes) having an identical name string, grouped by the document to which they belong. Each element (or attribute) record includes an <order, size> pair and other related information of the element (or attribute) as shown in Figure 2.10.

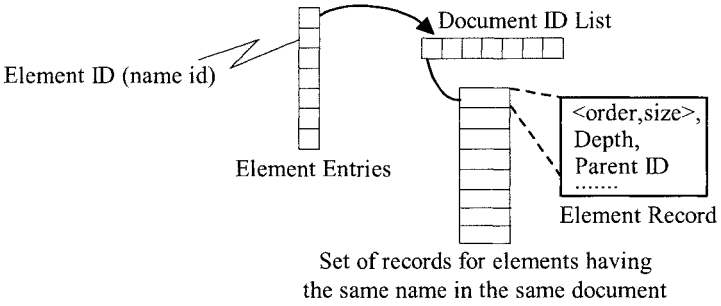

Figure 2.10 Element Index

For structure index, all elements and attributes of one document are kept in one array. Within an array, the elements and attributes are together sorted by their order value in preorder traversal. Each record of an array stores name identifier, order values of the first sibling, first child, and the first attribute and so on as shown in Figure 2.11.

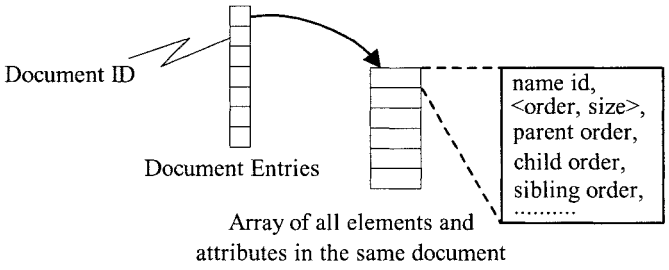

Figure 2.11 Structure Index

Seo *et al* [115] proposed an inverted index for XML query processing using an RDB. The researchers presented the inverted index kept in four tables as follows:

Path(path, pathID), PathIndex(pathID, docID, begin, end),

Term(term, termID), TermIndex(termID, docID, pathID, position)

To process containment queries, normally only two join operations will be performed: one join between the table Term and the table TermIndex, and one join between the table Path and the table TermIndex.

For example, to process the query:

/books/book/summary/keyword/'XML', the operations will be:

(I) extract the pathID of "/books/book/summary/keyword/" from table Path and extract termID of 'XML' from the table Term

(II) extract from the table TermIndex, tuples which have the same pathID and termID by a join with the table Path and the table Term respectively.

The researchers identified that their approach outperforms the 2-INDEX approach [144]. The 2-INDEX approach is T-index for indexing text words and E-index for indexing elements. Two tables for indexing are as follows:

Texts(word, docno, wordno, level), Elements(element, docno, begin, end, level).

A serious disadvantage of the 2-INDEX approach is that the number of joins relies on the path length of queries since it requires a self-join operation on the table Element for processing every containment relationship between two elements.

For example, to process the query: /books/book/summary/keyword/'XML', it requires four joins since path length of the query is four.

Schoning [114] presented text index and structure index. In the text index, the words contained in an element or attribute are indexed. Many elements/attributes may contain the same word; thus the word is indexed to point to all these elements/attributes. For the structure index, it keeps information of all paths of a particular document. This is used to optimise querying for optional elements/attributes: elements with occurrence '?' or '*' and #IMPLIED attributes. If an element/attribute does not appear in the document, this can be determined by using the path information.

2.6.3 Optimisation by algebra

Similar to the relational algebra, the purpose of the XML Query Algebra is twofold [131]. First of all, the algebra is used to give semantics for the query language and, then the algebra is used to support optimisation. Most of optimisation rules in the XML query algebra are based on relational algebraic optimisation rules.

Frasincar *et al* [50] proposed an XML algebra called XAL and proposed optimisation rules based on relational algebraic optimisation rules such as decompose selection, commute selection and commutation of $\times$(cartesian product).

For example, a query is written in XQuery as follows:

```
<result>
{     For $i in doc("printers.xml")/painter,
          $j in doc("paintings.xml")/painting[author = $i/name],
          $k in doc("catalogue.xml")/item[paintingid = $j/id]
      Where $k/price/data() > 10000
      Return $i/name
}
</result>
```

The tree derived from the query is shown in Figure 2.12(a). By applying optimisation rules (i) decompose selection: $\sigma[c_1 \wedge \ldots c_n](e) = \sigma[c_1](\ldots(\sigma[c_n](e))\ldots)$ and (ii) commute selection: $\sigma[c_1](\sigma[c_2](e)) = \sigma[c_2](\sigma[c_1](e))$, the tree is changed to the tree shown in Figure 2.12(b). Finally, (iii) apply commutation of $\times$: $(e_1 \times e_2 \times e_3 = e_3 \times e_2 \times e_1)$ so that the most restrictive selections will be performed first. The final tree is shown in Figure 2.12(c). Note that $\sigma[c_1 \wedge \ldots c_n](e) = \sigma[c_1](\ldots(\sigma[c_n](e))\ldots)$ means in selecting data if there are several conditions linked by 'and' operators, then the conditions can be decomposed and each condition can be evaluated separately on the selected data.

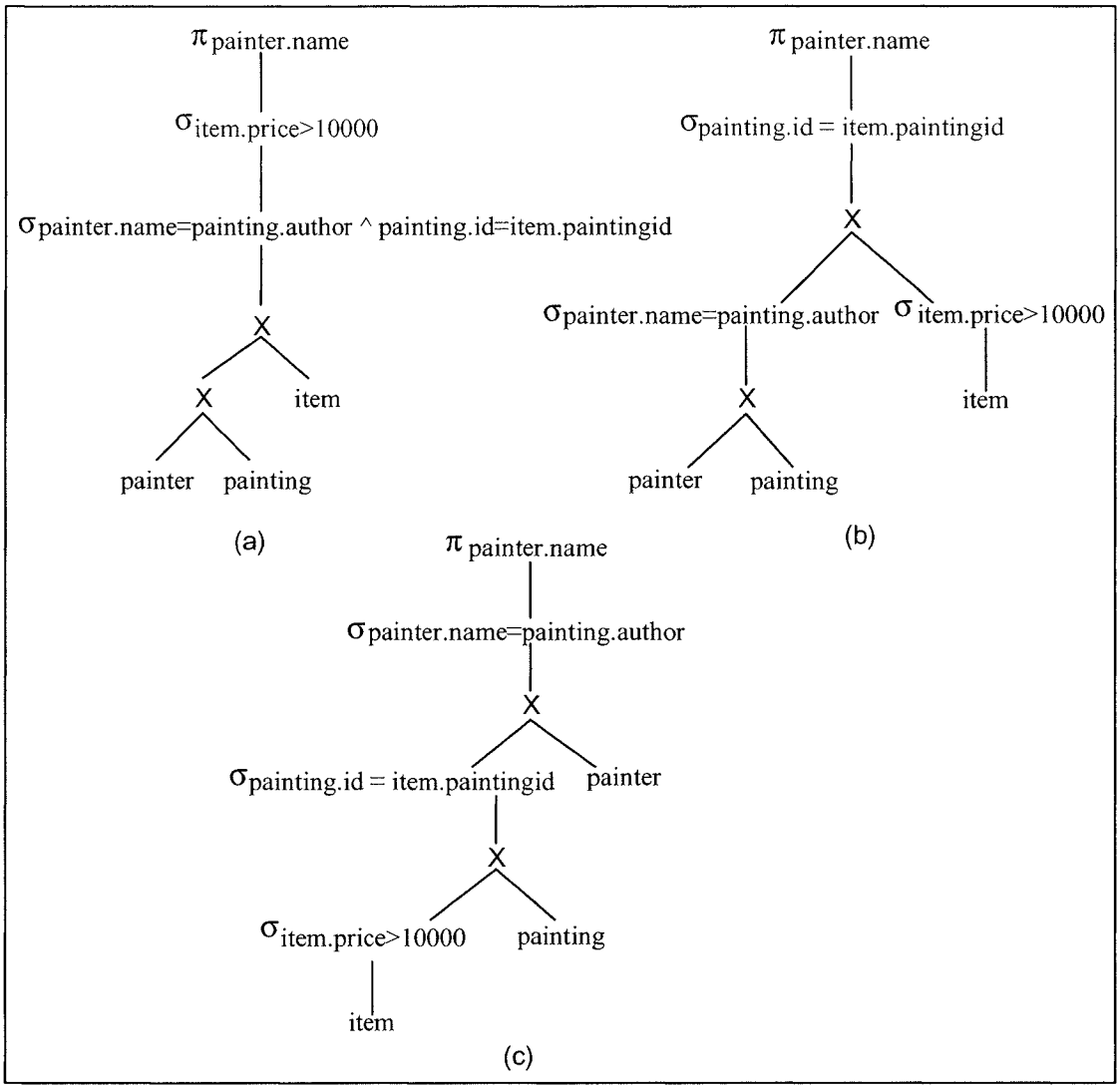

Figure 2.12 query tree

Sartiani *et al* [112] presented an XML algebra, an extension of the common object-oriented query algebra. Algebra operators are classified into two groups: basic operators and operators extended for use in XML. Two algebra-operators are extended for XML: *path* and *return*. The *path* operator is used to extract data and to build variable binding. The *return* operator is used to produce new XML documents. The basic operators are very similar to the operators of the relational and object-oriented algebra. As with XAL [50], the researchers proposed rules for decomposition based upon optimisation rules in the relational algebra.

Beeri *et al* [13] proposed an XML algebra called SAL (Semistructured Algebra). Only one optimisation, pushdown selection is mentioned in this work. The principle of this technique is changing the order of operations to apply predicates as soon as possible.

This technique is the same as the technique proposed by Grinev *et al* [54, 56] and in turn the same as relational algebraic optimisation rules.

Although currently various techniques of optimisation for querying XML documents are proposed, most of the work focuses on the optimisation for querying XML whose model is the tree model, whereas there is not much work presented on the optimisation for querying XML whose models are based on traditional database models.

2.7 XML benchmarking

In this section, general concepts about benchmarks are reviewed first and then specific benchmarking for XML is explored.

2.7.1 Benchmark

A benchmark is used to test the performance of a system. Different systems have different variables which are important to the systems. For example, in a network system, transaction cost is critical whereas in a database system, processing time is crucial. Thus a benchmark must capture the major characteristics of the system to be measured. Gray [53] defined four key criteria for a domain-specific benchmark:

- Relevance
  The benchmark must capture the characteristics of the system to be measured
- Portability
  The benchmark should be able to be implemented in different systems
- Scalability
  The benchmark should still be applicable as the data size increases.
- Simplicity
  The benchmark must be understandable; otherwise it will not be credible.

There are several domain-specific benchmarks such as Wisconsin [14] which is widely used to test the performance of a relational query system. OO7 [24] is utilised to test an OODB and BUCKY [25] is employed to test an ORDB. These benchmarks mainly evaluate query processing performance.

2.7.2 XML management system benchmarks

It is difficult for a benchmark to capture all possible uses of XML; hence most benchmarks focus on a particular aspect of XML such as the performance of XML query processing.

The performance of XML query languages depends greatly upon their expressive powers, which are the functionalities they provide. The W3C [134] has published a list of "must have" requirements for XML query languages. The requirements specify that an XML query language must provide data-centric, document-centric and navigational capabilities. An XML query language must be as expressive as a structured query language (such as SQL for an RDB). Data-centric capabilities include join operations, aggregation and so on as found in an RDB. Ordering elements in queries is classified as a document-centric capability while traversal of an XML document structure using references or links is called a navigational capability.

There are several XML benchmarks proposed such as XOO7 [21], XMark [113], XMach-1 [17], Michigan [109] and XBench [141]. The details of these benchmarks are as follows.

XOO7 [21] is an XML version of OO7 with new elements and queries added to test the features that are unique in XML. The basic data structure in XOO7 is derived from OO7. To test XML query processing, the OO7 schema and instances are mapped to a DTD and the corresponding XML data sets respectively. The eight OO7 queries are translated into two XML query languages: Lorel [51] and Kweelt [111]. The data size can be changed in two directions: depthwise and breadthwise. Testing consists of eight queries taken from OO7 and an additional five queries designed for special XML aspects. XOO7 gives simple queries that test one or two functionalities; thus it is easy to analyse the results of the queries.

XMark [113] tests the performance of XML processing based on an application scenario that models an Internet auction site by using twenty XQuery commands. XMark uses many references in the test data such as IDREF in an auction element and the item's ID in an item element. The data size can be changed with a scaling factor of 1.0. Although XMark queries will cover most of the functionalities of the XML query language, the majority of the queries are complex and cover several features that may lead to difficulty in analysing the results of a query because it may be ambiguous which

feature is responsible for most of the response time. Moreover, it is possible that some queries cannot be executed since the systems under test may support only a subset of the complex features.

XMach-1 [17] is a multi-user benchmark designed for Web applications. XML data to be tested with XMach-1 has a small size and simple data structure. The benchmark supports both schema-based and schema-less XML management systems. The size of data is controlled by the number of files and the size of files; however, XMach-1 assumes that the size of the data files exchanged in the Web applications will be small (1-14 KB). The benchmark consists of only eight queries and two update operations; hence when compared with other XML benchmarks, it covers the fewest functionalities. As in XOO7, XMach-1 gives simple queries that test one or two functionalities so that it is easy to analyse the results.

While Wisconsin [14] is designed for measuring the performance of a relational DBMS, Michigan [109] is a comparable benchmarking tool for XML data management systems. In Wisconsin, the primary interesting data characteristics are the distribution and domain of the attribute values, the cardinality of the relations and the tuple/attribute size but in Michigan, besides these characteristics, several other characteristics such as tree fanout and tree depth related to the structure of XML documents are of interest. The benchmark uses a single large document tree as the default data set. The benchmark evaluates the cost of individual XML query operations instead of composite performance of complex queries with several functionalities; thus the number of tested commands rises to 49 queries and 7 update operations to test most of the essential functionalities of an XML query language.

XBench [141] classifies XML database applications into two dimensions: application characteristics and data characteristics. Two types of application characteristics are data-centric and text-centric while two classes of data characteristics are single and multiple documents. Thus XBench tests performance of XML processing based on four cases as follows: data-centric/single document, data-centric/multiple documents, text-centric/single document and text-centric/multiple documents. Examples of these four cases are E-commerce catalogues, online dictionaries, digital libraries and transactional data respectively. Twenty queries covering twelve functionalities of XQuery specification are tested in XBench.

Most XML benchmarks are used for measuring the performance of XML query processing. The techniques of these benchmarks can be exploited to measure the performance of XML update processing. This is because before updating data, the data must be searched by the query engine.

2.8 Summary

This thesis presents updating XML through ORDB; thus the general steps for this solution are mapping XML schemas to the database schemas and translating an XML update language into SQL executed on the database. Thus relevant literature covers not only the languages for querying/updating XML and optimisation techniques but also XML mapping approaches and language translation techniques.

The related literature has been surveyed and it has been shown that the research area of updating XML documents is still immature, and therefore some problems are exposed. The comparisons of capabilities and limitations for mapping techniques, query languages, update languages and language translation techniques for XML are shown in Table 2.1-2.4. Five major points from the literature are summarised as follows.

Mapping XML to conventional databases: Not many pieces of research handle XML constraints during mapping XML to the databases; however, those that do - except [85, 103] - define constraints conflicting with each other or which are impracticable in the current technology. Other researchers ignore the XML constraints or define just some XML constraints. In addition, there is just one work [40] supporting simple update in the query language.

XML query and XML update languages: XQuery, a standard from W3C, is the most powerful of the existing XML query languages; thus some work has made extensions to it while other work proposed new query languages supporting the update feature. Nonetheless, none can update multiple linked documents and join documents in updates. Only one work [83] can update data whose structure is recursive but this work has never been implemented. The existing XML databases cannot guarantee the preservation of constraints when the updates are performed.

Translating XML query language into SQL: None of existing work translates the recursive feature of XQuery into SQL. Several other features of XQuery such as aggregate functions and quantifier have never been translated into SQL.

Optimisation for querying XML: We classify optimisation techniques for XML into three categories: optimisation by eliminating unnecessary paths, optimisation by indexing and optimisation by algebra. There is not much work presenting the optimisation techniques for XML whose models are based on traditional database models. Most of the work focuses on the optimisation for XML whose model is the tree model.

XML benchmark: Measuring XML query processing is related to the design of data sets and sets of queries. The size of data can be scaled by several options such as changing depth and fanout. Sets of queries are categorised according to the functionalities provided by the XML query language. The techniques for benchmarking XML query processing can be applied to measuring the performance of XML update processing.

Chapter 3, the next chapter, presents mapping the XML schema to the ORDB schema. In addition, a method for handling the cardinality constraint in XML is proposed as well.

Chapter 3: XML structure and constraints

Chapter 1 introduced problems in the updating of XML and our approach to solve them while related literature to our approach was surveyed in Chapter 2. This chapter describes the mapping of XML documents to an ORDB, the first step in our approach. Recall for our approach, the overview is illustrated in Figure 3.1.

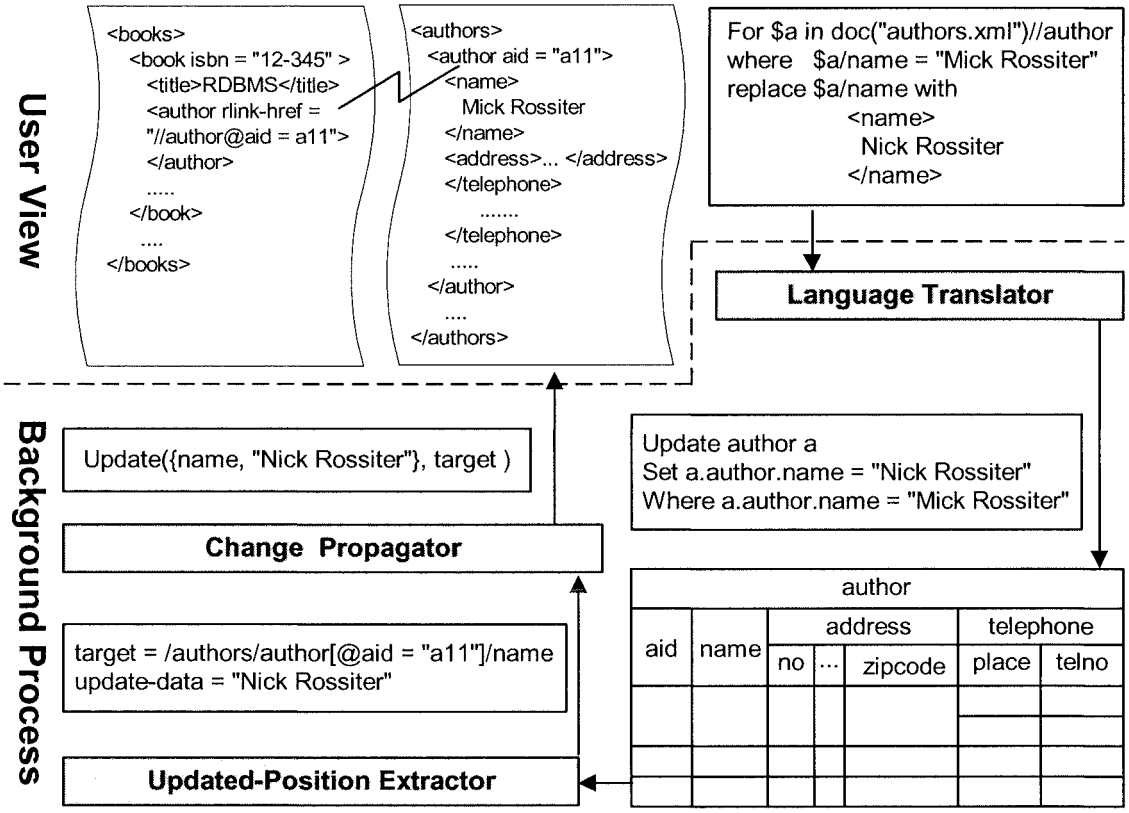

Figure 3.1 Overview of the approach to updating XML documents

To avoid storing redundant data in one single XML document, non-redundant data is kept in several XML documents instead. The elements in these documents will be linked together by a mechanism called `rlink`. An attribute of `rlink` is rlink-href whose value is an XPath expression used to associate a relationship between elements in different documents. To update these linked XML documents via ORDB and let a

database engine handle the preservation of constraints, the XML structure and `rlink` including the constraints are first mapped to the ORDB, then XML data is loaded into the database according to the XML structure. Next, users update XML data using a view of XML documents and the XML update language, an extension to XQuery. Running as a background process, the XML update language is then translated into SQL to update XML data stored in an ORDB. Finally, the change in the ORDB is propagated to the XML documents as follows. The positions of elements or attributes in XML documents that should be updated are indicated by the values for keys of updated rows in the ORDB. The data for updating XML documents is derived directly from the XML update language. The positions and data for updating XML documents are sent to propagate functions as their parameters. The propagation functions provide the process for updating XML documents.

In this thesis, instead of RDB and OODB, ORDB is used for mapping XML. The structure of XML documents is hierarchical whereas the RDB structure is flat; therefore mapping XML to an RDB is not a good fit, producing many unnecessary tables. Usually XML documents contain multi-value elements which can only be mapped to RDB by putting the data into separate tables. When the data is queried, several joins of tables are performed; thus low performance of querying can occur. On the other hand, an OODB can handle multi-value elements and hierarchical structure since it provides collection types such as list and set, and a container class which can contain other classes; thus it seems that the OODB is suitable for keeping XML data. However, some constraints such as, not null, delete cascade or even trigger are not supported in the OODB; moreover, the OODB does not provide update statements. This thesis focuses on updating XML data; thus an ORDB should be more suitable for storing XML data. This is because in the ORDB, constraints and update statements are inherited from the RDB and the OO features are inherited from the OODB. In order to update XML documents and preserve constraints during updating, storing XML documents in the ORDB should be a better approach. In this research, both structure and constraints of (linked) XML documents are mapped to an ORDB with practical issues in mind to suit the available ORDB technologies.

There are three major methods for mapping XML to conventional databases: schema-based, schema-oblivious and mixed methods. Without an XML schema, XML will never reach its full potential since the meaning of tags used as an agreement among

applications can never be expressed. In others word, the Web community is very excited about XML because it is anticipated that the vast majority of files on the Web will be XML conforming to some standardised schemas; thus applications can interpret XML files by consulting their schemas. To exploit the advantage of the XML schema, our mapping is based on the Shred method (the schema-based method). Querying data stored in a database by a schema-based method can yield better performance than querying data stored in a database by a schema-oblivious method since the schema enables the query to be optimised.

Many XML schemas have been proposed but only two, DTD and XML Schema, are recommended by W3C. Rather than XML Schema, DTD is used in our mapping since the syntax of DTD is more compact than that of XML Schema and an investigation of 200,000 XML documents by Mignet *et al* [95] indicates that 48% of XML documents on the Web link to DTD whereas only 0.09% of the XML documents use XML Schema.

The rest of this chapter is organised as follows. Section 3.1 describes a mechanism used to link XML documents together. Section 3.2 presents rules for translating an XML structure to an ORDB schema. Section 3.3 presents types of constraints in an ORDB and also presents rules for mapping XML constraints to ORDB constraints, including the preservation of the cardinality constraint during the updating of XML. Section 3.4 gives a case study for mapping linked XML documents to an ORDB. Section 3.5 summarises this chapter.

3.1 A mechanism for linking XML documents

Data redundancy can occur when all data is kept within one XML document. Thus as in an RDB where it is necessary to keep non-redundant data in several tables, non-redundant data can be stored in several XML documents with these documents linked together by some mechanisms.

In an (O)RDB, foreign keys are employed to represent inter-table references; thus to model linked XML documents, a mechanism for linking XML documents can be translated into foreign keys in an (O)RDB. For XML documents, XLinks (XML Linking Language) [132] and XInclude [135] are mechanisms for linking the documents together. However, we do not wish to make any extension to such

mechanisms since XLink and XInclude are not designed from a database viewpoint; they do not provide enough information for linking from a database viewpoint. The major purpose of XLink is to link XML documents on the Web while the main purpose of XInclude is to build a large XML document out of smaller XML documents. XInclude [59] does not allow circular reference (recursion). XLink is supported only by Mozilla and its derivatives such as Netscape but the support is incomplete [60]; moreover, no XML query language supports XLink.

In this thesis, a mechanism called `rlink` is presented. Its purpose is to associate the relationship between elements from different XML documents. `rlink` provides information to identify the document and element to which a link is made. `rlink` consists of two attributes: rlink-relationship and rlink-href. The rlink-relationship indicates which document and which element are involved in `rlink` whereas the rlink-href indicates the links to the document and the element indicated by the rlink-relationship.

`rlink` will only serve the function of linking elements in XML documents; thus an element containing `rlink` must be EMPTY and have no other attributes except the rlink-relationship and the rlink-href. The element containing `rlink` is called 'rlink-element'. In a DTD, the format of a value assigned to the rlink-relationship is LinkedDocument::LinkedElement and its property is FIXED while the value type of rlink-href is CDATA. In XML documents, the format of the value assigned to the rlink-href is an XPath expression which links to the document and element specified by the rlink-relationship in the DTD. To illustrate, an example of using the rlink-relationship and the rlink-href in DTDs and XML documents is shown in Figure 3.2.

```
<!DOCTYPE Publications[
<!ELEMENT Publications(Publication*)>
<!ELEMENT Publication (Title, Year, Author+)>
<!ATTLIST Publication  PubID ID #REQUIRED>
<!ELEMENT Title       (#PCDATA)>
<!ELEMENT Year        (#PCDATA)>
<!ELEMENT Author      EMPTY>
<!ATTLIST Author rlink-href CDATA #REQUIRED
 rlink-relationship #FIXED "Authors::Author">
]>

<Publications>
<Publication PubID = "P111">
 <Author rlink-href="//Author[@AutID='A222']">
 </Author>
    ...
</Publication>
</Publications>
```

```
<!DOCTYPE Authors[
<!ELEMENT Authors (Author*)>
<!ELEMENT Author(Name, Email?)>
<!ATTLIST Author AutID ID>
<!ELEMENT Name  (FName, LName)>
<!ELEMENT FName (#PCDATA)>
<!ELEMENT LName (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
]>

<Authors>
<Author AutID = "A222">
   <Name>...</Name>
      ...
      ...
</Author>
 ...
</Authors>
```

Figure 3.2 Publications.xml and Authors.xml are linked together by `rlink`

From Figure 3.2, two XML documents, Publications.xml and Authors.xml are linked together by `rlink`. The Author element in Publications.xml uses the rlink-href as an attribute to link information of the Author whose AutID = 'A222' from Authors.xml.

Besides using `rlink` to represent the relationship between elements across XML documents, the attribute 'rlink-relationship' is applied to IDREF(s) in a DTD to indicate the involved document and element. This enables IDREF(s) to refer to an element in a different document since usually IDREF(s) can refer only to elements in the same document.

3.2 Mapping XML structure to ORDB structure

As mentioned earlier, DTDs are used in mapping XML to an ORDB. Normally, there are two forms of recursion in the DTD but a third appears in the present work. The first form comes from a recursive structure: an element contains its ancestor elements as child elements. The second form stems from IDREF(s). Because `rlink` is proposed, the third form of recursion results in the case of two XML documents referencing each other by `rlink`.

This section describes the rules for mapping an XML structure and `rlink` to ORDB. Before applying these mapping rules, three collections of simplifying rules are applied. The second and third collections of simplifying rules are derived from Lu *et al* [84].

The symbol ➔ stands for 'is converted to'.

The first collection of simplifying rules for groups of elements:

(1) $(e_1 |\ldots| e_n)^* \rightarrow (e_1^* |\ldots| e_n^*)$

(2) $(e_1|\ldots| e_n)+ \rightarrow (e_1+ |\ldots| e_n+)$

(3) $(e_1|\ldots| e_n)? \rightarrow (e_1? |\ldots| e_n?)$

The second collection of simplifying rules:

(1) $(e_1,\ldots,e_n)^* \rightarrow (e_1^*,\ldots,e_n^*)$

(2) $(e_1,\ldots,e_n)+ \rightarrow (e_1+,\ldots,e_n+)$

(3) $(e_1,\ldots,e_n)? \rightarrow (e_1?,\ldots,e_n?)$

The third collection of simplifying rules:

(1) e**, e*+, e*?, e+*, e+?, e?*, e?+ → e*

(2) e++ → e+

(3) e?? → e?.

From the first collection of rules, each group of elements has a subtly different meaning as follows:

Group of elements	Meaning
$(e_1 \|\ldots\| e_n)^*$	$e_1,\ldots,e_n$ can occur zero or more times and in any order.
$(e_1\|\ldots\| e_n)+$	$e_1,\ldots,e_n$ can occur one or more times and in any order.
$(e_1^* \|\ldots\| e_n^*)$	A single element of $e_1\|\ldots\|e_n$ can occur zero or more times.
$(e_1+ \|\ldots\| e_n+)$	A single element of $e_1\|\ldots\|e_n$ can occur one or more times.
$(e_1\|\ldots\| e_n)?$ $(e_1? \|\ldots\| e_n?)$	A single element of $e_1\|\ldots\|e_n$ can occur zero or one time.

When XML constraints are checked, the group of elements before applying the simplifying rules will be considered whereas during mapping an XML structure to an ORDB structure, the group of elements after applying the simplifying rules will be considered.

The second collection of simplifying rules, $(e_1,\ldots,e_n)^*$, $(e_1,\ldots,e_n)+$ and $(e_1,\ldots,e_n)?$ mean that the number of occurrences for $e_1,\ldots,e_n$ must be equal. Thus before inserting or deleting these elements, a check will be made as follows. If one element is inserted or deleted, other elements must be inserted or deleted as well. For example, if an instance of e_1 is inserted or deleted then an instance of $e_2\ldots e_n$ must be inserted or deleted as well

After applying the simplifying rules, the mapping rules will be applied. A diagram for mapping an XML structure and `rlink` to an ORDB is shown in Figure 3.3 (page 62). The mapping rules are based on two criteria: performance and the available technologies. For the performance, we limit the number of object layers: a nested table and an ADT cannot contain another nested table or ADT, otherwise unlimited object layers can occur causing the problem of low performance since when the condition in a command is tested, objects must be unnested first. For the available technologies, the mapping rules are based on the technology of Oracle since its ORDB has more capability than other ORDB technologies.

3.2.1 Rules for mapping XML structure to ORDB structure

In our mapping rules, three features of the object-relational technology, abstract data type (ADT), object table and nested table, will be used. In the rules, elements whose type is #PCDATA and without attributes are called simple elements whereas elements consisting of child-elements and/or attributes are called complex elements.

1. Complex elements which do not correspond to the rules 2-5 are converted to object tables.
2. Complex elements having only one complex child-element are converted to object tables and their complex child-elements are converted to ADTs.
3. Complex elements which have occurrence ? or 1, have siblings, have all children as simple elements whose occurrence is ? or 1 and have attributes whose type is not IDREF(s) are converted to ADTs.

   Note: there is no single symbol for occurrence meaning one, from now on we will use the symbol '1' as a canonical short label.
4. Complex elements having only a number of complex child-elements are ignored in the conversion: that is, if all the child elements are converted to object tables, they are not converted to anything in an ORDB.
5. Complex elements which have occurrence * or +, have siblings and which comply with the following conditions, are converted to nested-tables:
   - All children are simple elements whose occurrence is ? or 1 and all attributes whose type is not IDREF(s).
   - There is no reference to other elements and from other elements to them.

This rule is to make sure that nested tables do not contain references since the referential integrity constraint cannot be defined in nested tables; however, other constraints such as domain constraint and default constraint can be defined in nested tables.

6. Multi-valued simple elements (simple element having occurrence + or *) are converted to nested tables.
7. Simple elements, attributes whose type is not IDREFs, optional simple elements, choice of simple elements and elements with type `ANY` are converted to fields.
8. Optional complex elements and choice of complex elements are converted to (nested or object) tables or abstract data types according to the rules 1-5.
9. For a choice of groups of elements in which some of the elements in each group are the same, duplicate elements are eliminated and then rules 1-8 are applied.
10. For a parent-child relationship and recursive structure, the primary key of the table of the parent-element is copied to the table of the child-element.
11. For IDREFs, a separate table will be created to keep the primary keys of tables of referencing and referenced elements.

3.2.2 Rules for mapping the rlink to ORDB structure

The relationship between XML documents specified by `rlink` is similar to the relationship specified by IDREF(s) in the same document; thus their mapping rules are similar too.

1. If the occurrence of an rlink-element is 1 or ?, the primary key of the table of a referenced element is added to the table of a referencing element.
2. If the occurrence of an rlink-element is + or *, a separate table will be created to keep the relationship between XML documents; thus the separate table consists of the primary keys of the table of a referencing element and a referenced element.
3. For a recursive structure: an rlink-element in a referenced document refers back to an element in a referencing document, such an element is considered in the same way as an element under rules 1-2.

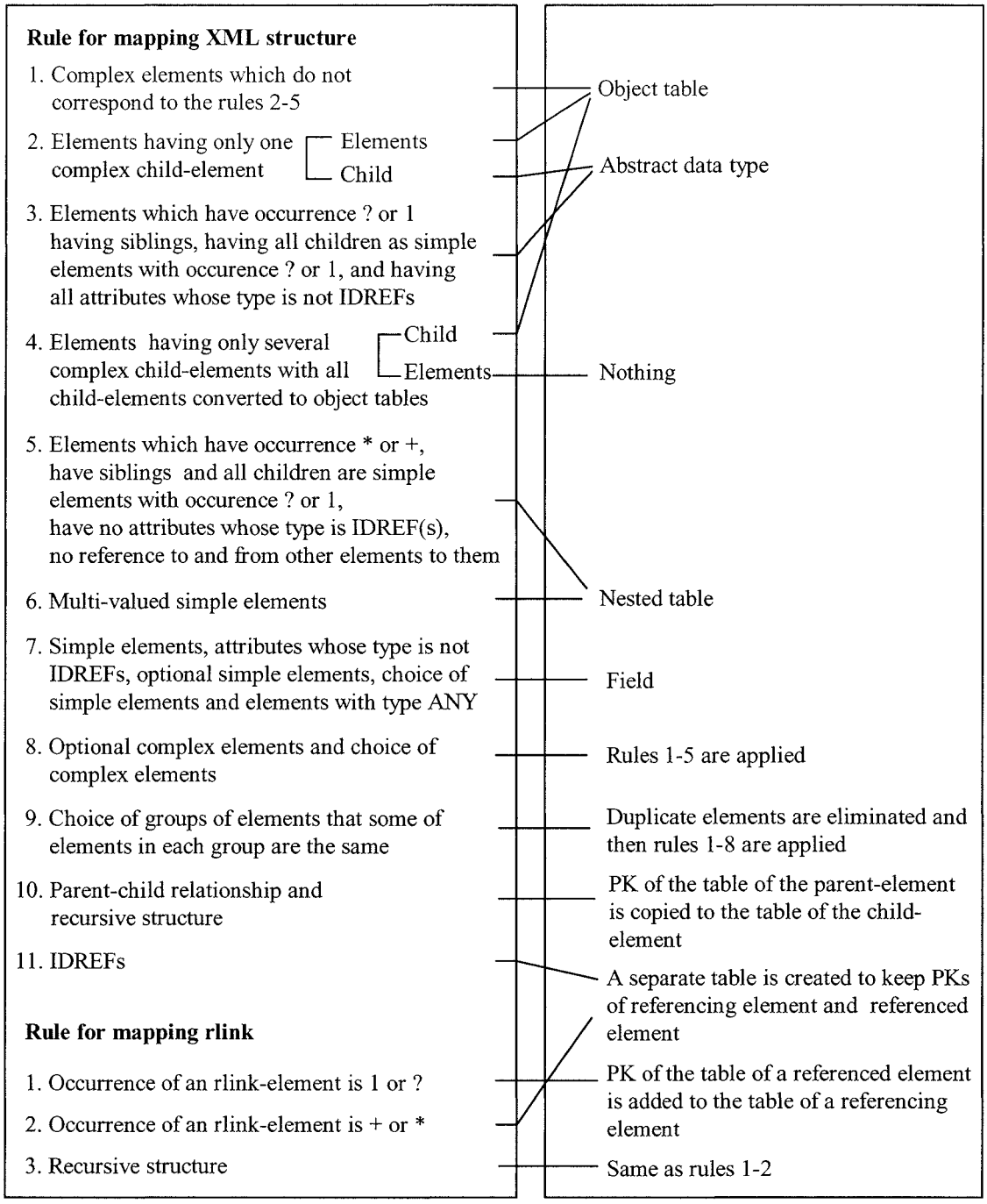

Figure 3.3 Mapping XML structure and rlink to ORDB structure

3.3 Constraint Rules

Constraints of an XML document are defined in DTDs; thus when XML data is stored in an ORDB, mapping XML constraints to object-relational constraints is necessary. This section starts with a reminder about the types of constraints in an (O)RDB and then presents the rules for mapping XML constraints to ORDB constraints. Finally supplementary rules for mapping constraints resulting from `rlink` to ORDB constraints are presented. A diagram for mapping XML constraints and `rlink` constraints to ORDB constraints is shown in Figure 3.4 (page 67). The constraint rules are derived from translating the meaning of XML constraints to those of ORDB constraints.

3.3.1 Type of constraints

The constraints in an (O)RDB from a data-oriented viewpoint can be categorised into three types [128] as follows.

1. Row constraints: these constraints are related to exactly one table and can be evaluated independently for each row in that table. Constraints of this type include a check (null value) constraint, a domain constraint and a default value constraint.
2. Table constraints: evaluating these constraints is associated with at least two rows in the same table. These constraints include a primary key constraint, a unique constraint and a cardinality constraint.
3. Inter-table constraints: these constraints involve rows from at least two tables. This constraint type is a foreign key constraint (referential integrity constraint) including cascade rules.

Currently, the cardinality constraint is not available in any (O)RDBMS.

3.3.2 Rules for mapping XML constraints to ORDB constraints

Some constraints in a DTD can be easily mapped to (O)RDB constraints; thus a number of the rules proposed here are the same as some rules proposed in other work [78, 79]. However, our work can extract more constraints from a DTD than in previous work and some of our rules are different from the rules proposed in previous work; in particular no work has proposed preserving the cardinality constraint when updates are performed. In this section, we will organise constraints in a DTD according to constraint types found in an (O)RDB as follows.

Row constraints:

1. #REQUIRED attributes and elements with occurrence 1 or + are converted to a not null constraint.
2. #IMPLIED attributes and elements with occurrence ? or * have a default null constraint.
3. The default value is translated into a default value constraint.
4. The choice of attribute values is converted to a domain constraint.
5. For a choice of elements such as <!ELEMENT e (s_1 | s_2)>, meaning can have either s_1 or s_2 but not both simultaneously, then the constraint will be:

   ```
   Check ((s1 is not null AND s2 is null) OR
          (s1 is null AND s2 is not null))
   ```

   Thus, for <!ELEMENT e(s_1|…|s_n)>, the constraint is:

   ```
   Check((s1 is not null AND s2 is null AND … AND sn is null)
         OR … OR (s1 is null AND s2 is null AND…AND
                  sn-1 is null AND sn is not null))
   ```

   For <!ELEMENT e(s_1?|...|s_n?)>, the constraint is:

   ```
   Check((s1 is not null AND s2 is null AND … AND sn is null)
         OR … OR (s1 is null AND s2 is null AND…AND
                  sn-1 is null AND sn is not null)
         OR(s1 is null AND s2 is null AND … AND sn is null))
   ```

6. For a choice of groups of elements, in which some of the elements in each group are the same but their constraints may be different, the constraints are converted with the following rules.

   Firstly, every element+ or element in each group is converted to "AND element is not null". Secondly, every element* or element? in each group is ignored (is not converted to anything). Thirdly, if there are some elements in one group which do not appear in other groups, the "AND these elements are null" is added to the

groups for which these elements do not appear. Finally, the OR operation is performed between groups.

For example: (name, telephone+) | (name, telephone*, email), the constraint is:

```
Check ((name is not null AND telephone is not null
        AND email is null)
      OR (name is not null AND email is not null))
```

Table constraints:

1. ID of an element is converted to a primary key constraint except when the table of the element has a primary key already and the element is converted to ADT. In this case the ID of the element is converted to a unique constraint.
2. Occurrence of elements converted to nested or object tables should be converted to the cardinality constraint; however, no (object-)relational technology provides this constraint; hence this constraint will be checked by the rules for preserving the cardinality constraint in section 3.3.4.

Inter-table constraints:

This type of constraint is related not only to IDREF(s) but also to the parent-child relationship; ergo we characterize these constraints according to types of relationship in XML documents as follows:

1. Parent-child relationship (1 to many relationship)
2. Parent-child relationship (1 to 1 relationship)
3. Referencing from descendants to ancestors by recursive structure with occurrence ? or 1
4. Referencing from descendants to ancestors by recursive structure with occurrence * or +
5. Referencing from descendants to ancestors by IDREFs or IDREF (recursive)
6. Referencing from ancestors to descendants by IDREFs or IDREF
7. Elements (siblings or relatives) referencing each other by IDREFs or IDREF

From parent-child relationships and recursive structures (1-4), the primary key of the parent-element table is copied to the child-element table as a foreign key constraint and a delete cascade is defined on this constraint.

From referencing by IDREFs (5-7), a new separate table will be created to hold relationships of references. This separate table consists of two fields derived from the

primary keys of the tables of a referencing element and a referenced element. These fields are set as a composite key and as foreign keys for this table. The foreign key derived from a referencing element is defined with a delete cascade whereas the foreign key derived from a referenced element is defined without a delete cascade.

From referencing by IDREF (5-7), IDREF is converted to a foreign key of the table of a referencing element without a delete cascade to point to the primary key of a referenced element.

Note: For an object table without a primary key, the RowID automatically created in an object table will be used as the primary key.

3.3.3 Additional rules for mapping constraints resulting from the rlink

The supplementary constraint rules for mapping `rlink` are similar to the constraint rules of IDREF(s). The rules are as follows.

1. In the case that the occurrence of an rlink-element is 1 or ?, the primary key of the table of a referenced element will be held in the table of a referencing element as a foreign key without a delete cascade.
2. In the case that the occurrence of an rlink-element is * or +, a separate table is created to hold the two fields derived from the keys of the tables of referencing and referenced elements. These two fields are set as a composite key and foreign keys but only the foreign key derived from the table of a referencing element is defined with a delete cascade whereas the foreign key derived from the table of a referenced element is defined without a delete cascade.
3. For a recursive structure, constraints are considered in the same way as rlink-elements using rules 1, 2 and 4.
4. Not null is defined on a foreign key in the case that the occurrence of an rlink-element is 1.

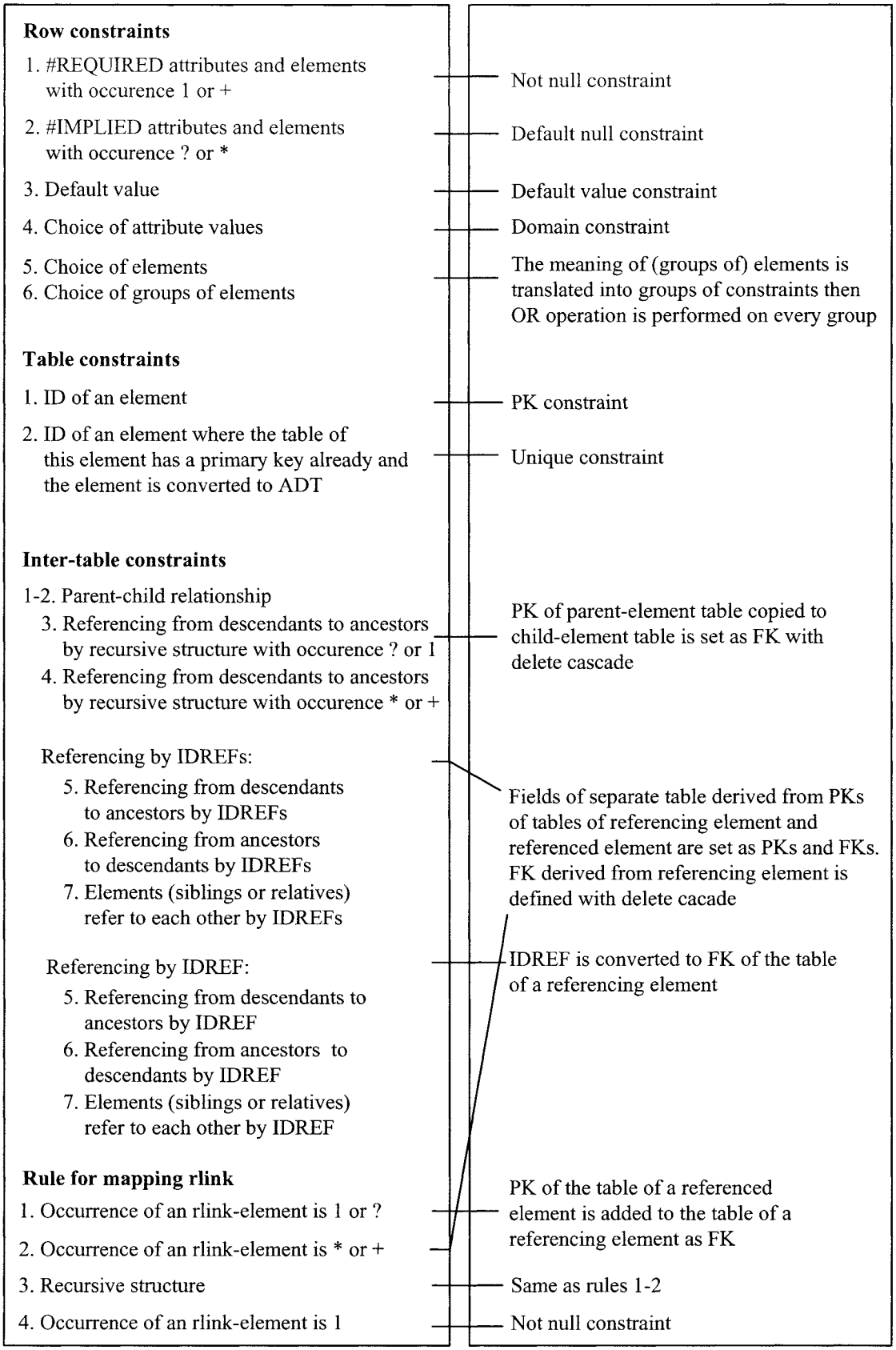

Figure 3.4 Mapping XML constraints and rlink constraints to ORDB constraints

3.3.4 Rules for preserving the cardinality constraint when updates are performed

This section will describe how to preserve the cardinality constraint when updates are performed. Nowadays, no (O)RDBMS can handle the cardinality constraint; hence a particular method is needed to manage it. The cardinality is the number of occurrences of elements. The number of occurrences or cardinality of elements is affected when new elements are inserted or existing elements are deleted. Usually one element will be referenced by another element or will be a child of another element except for the root element. Thus the cardinality constraint is the number of occurrences of an element in another element or the number of references from one element to another element. Thus, the cardinality constraint is related to the relationship of two elements.

In this section, basic rules for checking the cardinality constraint of an individual element will first be presented according to the types of relationship between elements. Next, in the case of a group of elements where each element is connected by '|', the cardinality constraint of each element in the group will be checked by applying the basic rules for counting the number of occurrences for an individual element.

Basic rules for checking the cardinality constraint of an individual element

The cardinality constraint of an individual element is categorised according to the types of relationship in the XML documents as given below.

1. Parent-child relationship (1 to many relationship) including recursive structure

   The child is an element converted to (i) an object table, (ii) a nested table or (iii) an ADT without siblings. The cardinality constraint will be checked as follows:

Delete child elements converted object tables in the case of occurrence +

```
Select count (*) as count1 from child-element table
Where FK (PK of parent-element table) = $PK;

Select count (*) as count2
from child-element table, tables related to delete-conditions
Where delete-conditions
And joins of tables;

If  count2 >= count1  then
    Do not allow deletion
End If
```

Delete child elements converted to nested tables in case of occurrence +

```
Select count (*) as count1 from table(select child-nested-table
                                      from parent-table
                                      where PK = $PK);

Select count (*) as count2 from table(select child-nested-table
                                      from parent-table
                  where delete-conditions not on child-nested-table)

Where delete-conditions on fields-of-child-nested-table;

If  count2 >= count1  then
    Do not allow deletion
End If
```

Delete child elements converted to ADTs in case of occurrence +

```
Select count (*) as count1 from parent-element table;

Select count (*) as count2 from parent-element table
Where delete-conditions;

If  count2 >= count1  then
    Do not allow deletion
End If
```

*Delete child elements in the case of occurrence **

No need to be checked.

*Insert children in the case of occurrence + or **

No need to be checked since any number of children is allowed when inserting

2. Parent-child relationship (1 to 1 relationship) including recursive structure

   The child is a complex element converted to (i) an object table or (ii) an ADT without siblings. The cardinality constraint will be checked as follows:

Delete/insert child elements in the case of occurrence 1

Do not allow deletion/insertion.

Delete child elements in the case of occurrence ?

No need to be checked.

Insert child elements converted object tables in the case of occurrence ?

```
If the number of child element that will be inserted > 1 then
   Do not allow insertion
End If

Select count(*) as count1 from child-element table
Where FK (PK of parent-element table) = $PK;

If count1 > 0 then
   Do not allow insertion
End If
```

Insert child elements converted to ADTs in the case of occurrence ?

```
If the number of child element that will be inserted > 1 then
   Do not allow insertion
End If

Select count(*) as count1 from parent-element table;

If count1 > 0 then
   Do not allow insertion
End If
```

3. Parent-child relationship (1 to 1 relationship)

   In the case that the child is an element whose occurrence is 1 or ?, or an attribute whose type is not IDREFs, the child is converted to a field, a field of ADT, a field of a nested table or an ADT having siblings.

Delete a child element converted to a field, a field of ADT, a field of a nested table or an ADT having siblings in case of occurrence 1 or ?

No need to be checked since the constraint of field is defined with not null already

Insert a child element converted to a field, a field of ADT, a field of a nested table or an ADT having siblings in case of occurrence 1

Do not allow insertion

Insert a child element converted to a field, a field of ADT or an ADT having siblings in case of occurrence ?

```
If the number of child element that will be inserted > 1 then
   Do not allow insertion
End If

Select count(*) as count1 from table containing the field
where PK = $PK
and the field is not null;

If count1 > 0 then
    Do not allow insertion
End If
```

Insert a child element converted to a field of a nested table in case of occurrence ?

```
If the number of child element that will be inserted > 1 then
   Do not allow insertion
End If

Select count(*) as count1 from table(
                              select nested-table
                              from table containing the nested table
                              where PK = $PK)
where condition based on fields of nested-table
and the field is not null;

If count1 > 0 then
    Do not allow insertion
End If
```

4. Referencing from descendants to ancestors by IDREFs (recursive)
5. Referencing from ancestors to descendants by IDREFs
6. Elements (siblings or relatives) referring to each other by IDREFs
7. Referencing between XML documents by using `rlink` where the occurrence of an rlink-element is + or *

For cases 4-7 above, in the case of referencing by IDREFs or by the rlink-element whose occurrence is + or *, a separate table is created to hold the relationship between the referencing and referenced elements. The cardinality constraint will be checked as follows:

Delete values in #REQUIRED IDREFs or delete rlink-elements whose occurrence is +

```
Select count (*) as count1 from separate table Where PK1 = $PK1;

Select count (*) as count2
from separate table, tables related to delete-conditions
Where delete-conditions
And joins of tables;

If  count2 >= count1  then
    Do not allow deletion
End If
```

*Delete values in #IMPLIED IDREFs or delete rlink-elements whose occurrence is **

No need to be checked.

Insert values to IDREFs or insert rlink-elements

No need to be checked.

Checking the cardinality constraint of a group of elements connected by '|'

Each type of a group of elements such as (e_1|...| e_n)+ and (e_1+ |...| e_n+) has subtly different meaning as mentioned in section 3.2.1. To check the cardinality constraint of a group of elements connected by '|', sometimes the number of occurrences of all elements in one group must be considered together since the appearance of one element may affect the appearance of another element in the same group. The basic rules for checking the cardinality constraint of an individual element are applied to counting the number of occurrences of each element in a group. Which rule will be used for each element in a group depends on the type of relationship between elements as specified in the basic rules.

1. (e_1|...| e_n)+ and (e_1|...| e_n)*

   (e_1|...| e_n)+ means that e_1,...,e_n can occur one or more times and in any order whereas (e_1 |...| e_n)* means that e_1,...,e_n can occur zero or more times and in any order. The cardinality constraint will be checked as follows:

Delete e_1|...| e_n in the case of (e_1|...| e_n)+

```
tot1 = 0;
tot2 = 0;
For i = 1 to n
      /* Applying the basic rules for count( ) */
      count(occurrence of ei) as count1;
      count(occurrence of ei) as count2 where delete-condition;
      tot1 = tot1 + count1;
      tot2 = tot2 + count2;
End For
If  tot2 >= tot1  then
      Do not allow deletion
End If
```

*Delete e_1|...| e_n in the case of (e_1|...| e_n)**

No need to be checked

*Insert e_1|...| e_n in the case of (e_1|...| e_n)+ and (e_1|...| e_n)**

No need to be checked

2. $(e_1+|\ldots|\ e_n+)$ and $(e_1*|\ldots|\ e_n*)$

$(e_1+\ |\ldots|\ e_n+)$ means that a single element of $e_1|\ldots|e_n$ can occur one or more times. For example, if e_1 occurs, other elements cannot occur. $(e_1*\ |\ldots|\ e_n*)$ means that a single element of $e_1|\ldots|e_n$ can occur zero or more times. The cardinality constraint will be checked as follows:

Delete e_i in the case of $(e_1+|\ldots|\ e_n+)$

```
/* Applying the basic rules for count( ) */
count(occurrence of ei) as count1;
count(occurrence of ei) as count2 where delete-condition;
If  count2 >= count1  then
   Do not allow deletion
End If
```

Delete e_i in the case of $(e_1*|\ldots|\ e_n*)$

No need to be checked

Insert e_i in the case of $(e_1+|\ldots|\ e_n+)$ and $(e_1*|\ldots|\ e_n*)$

```
count(occurrence of ei) as count1;
If count1 = 0 then  //if do not have ei, must check for other e
   For j = 1 to n       //check that no other e exists
      If j!= i then     //count other e except ei
         count(occurrence of ej) as count1;
         If count1 > 0
            Do not allow insertion //if have other e, cannot insert
            Break For
         End If
      End If
   End For
End If
```

In the case that a replace operation is achieved with a sequence of delete and insert operations, the cardinality constraint rules will be ignored.

3.4 A case study for mapping linked XML documents

To elucidate, we will demonstrate how to map linked XML documents to an ORDB. We suppose that four XML documents, Publications.xml, References.xml, Authors.xml and Publishers.xml, are linked together by the rlink-element consisting of two attributes: the rlink-relationship and the rlink-href as shown in Figures 3.5-3.8 respectively.

To gain more understanding, the Schema graph of DTDs in Figure 3.9 and the resulting tables in Figure 3.10 should be considered along with the description below.

```
<!DOCTYPE Publications [
<!ELEMENT Publications(Publication*)>
<!ELEMENT Publication (Title, Author+,Year, Publisher, Reference?, Cost,
                       SpecialCost?, ShippingCost?)>

<!ATTLIST Publication PubID ID #REQUIRED>
<!ATTLIST Publication PubType (book | article | journal) #IMPLIED>

<!ATTLIST Publication ContactAuthor IDREF #REQUIRED
                      rlink-relationship CDATA #FIXED "Authors.xml::Author">

<!ELEMENT Title (#PCDATA)>

<!ELEMENT Author EMPTY>
<!ATTLIST Author rlink-href CDATA #REQUIRED
                 rlink-relationship CDATA #FIXED "Authors.xml::Author">

<!ELEMENT Year (#PCDATA)>

<!ELEMENT Publisher EMPTY>
<!ATTLIST Publisher rlink-href CDATA #REQUIRED
                 rlink-relationship CDATA #FIXED "Publishers.xml::Publisher">

<!ELEMENT Reference EMPTY>
<!ATTLIST Reference rlink-href CDATA #REQUIRED
                rlink-relationship CDATA  #FIXED "References.xml::Reference">

<!ELEMENT Cost (#PCDATA)>
<!ELEMENT SpecialCost (#PCDATA)>
<!ELEMENT ShippingCost (#PCDATA)>

]>

<Publications>
    <Publication PubID = "P111">
      ...
      <Author rlink-href = "//Author[@AuthorID = 'A111']"></Author>
      <Publisher rlink-href="//Publisher[PName='Prentice Hall']"></Publisher>
      <Reference rlink-href = "//Reference[@RefID = 'R111']"></Reference>
    </Publication>
    <Publication PubID = "P222">
      ...
    </Publication> ...
</Publications>
```

Figure 3.5 Publications.xml

```
<!DOCTYPE References [
<!ELEMENT References (Reference*)>
<!ELEMENT Reference (Publication+)>
<!ATTLIST Reference RefID ID #REQUIRED>

<!ATTLIST Reference RefType (References | Bibliography) "References">

<!ELEMENT Publication EMPTY>

<!ATTLIST Publication rlink-href CDATA #REQUIRED
            rlink-relationship CDATA #FIXED "Publicastions.xml::Publication">

]>

<References>

   <Reference RefID = "R111" RefType = "Miscelleneous">
      <Publication rlink-href="//Publication[@PubID = 'P222']"></Publication>
   </Reference>  …
</References>
```

Figure 3.6 References.xml

```
<!DOCTYPE Authors [
<!ELEMENT Authors (Author*)>
<!ELEMENT Author (Name, Email?, Telephone*) >
<!ATTLIST Author AuthorID ID #REQUIRED>
<!ELEMENT Name  (FName, MName?, LName)>
<!ELEMENT FName (#PCDATA)>
<!ELEMENT MName (#PCDATA)>
<!ELEMENT LName (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Telephone (Location, TelNo)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT TelNo (#PCDATA)>
]>
<Authors>
    <Author AuthorID = "A111"> … </Author>  …
</Authors>
```

Figure 3.7 Authors.xml

```
<!DOCTYPE Publishers [
<!ELEMENT Publishers (Publisher*)>
<!ELEMENT Publisher (PName, Address?, SetupYear)>
<!ATTLIST Publisher PID ID #REQUIRED>
<!ELEMENT PName (#PCDATA)>
<!ELEMENT Address (No, Street, City, Country, Zipcode)>
<!ELEMENT No (#PCDATA)>
<!ELEMENT Stree (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Zipcode (#PCDATA)>
<!ELEMENT SetupYear (#PCDATA)>
]>
<Publishers>
    <Publisher> …<Publisher>
    …
</Publishers>
```

Figure 3.8 Publishers.xml

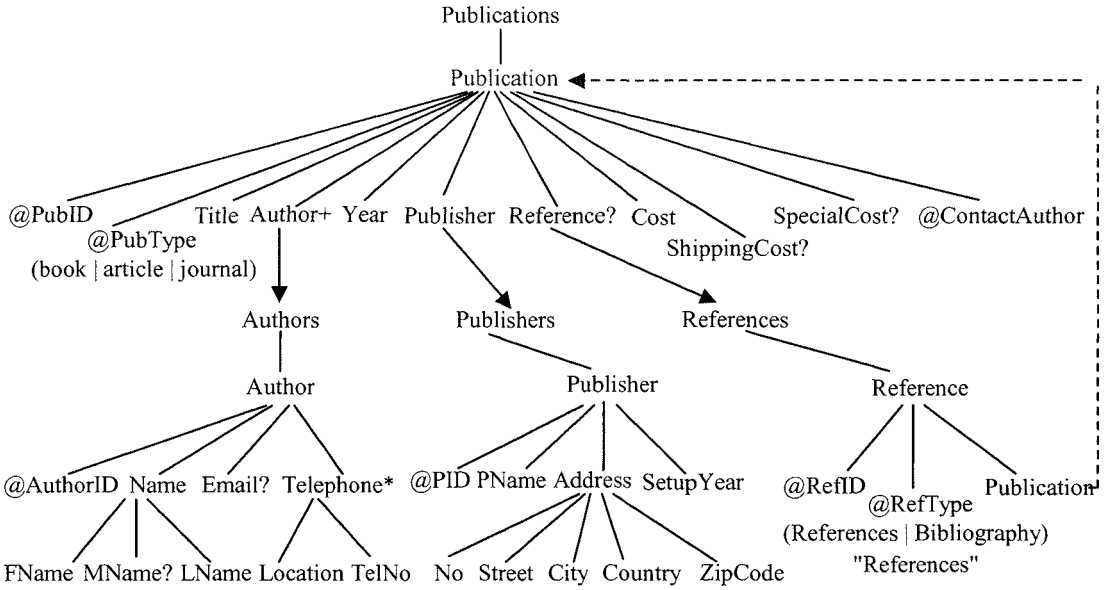

Figure 3.9 Schema graph of DTDs: a dashed-line denotes a recursion while an arrow denotes an rlink

Authors

Author						
AuthorID	Name			Email	Telephone	
	FName	MName	LName		Location	TelNo

Publications

Publication									
PubID	PubType	Title	Year	PID	RefID	Cost	ShippingCost	SpecialCost	ContactAuthor

ReferencePublication

RefID	**PubID**

PublicationAuthor

PubID	**AuthorID**

References

Reference	
RefID	RefType

Publishers

Publisher							
PID	PName	Address					SetupYear
		No	Street	City	Country	Zipcode	

Figure 3.10 Tables derived from the mapping rules

Firstly, *Publications* has only one complex child element: *Publication*. Thus *Publications* is converted to an object table and *Publication* is converted to an ADT (Mapping Rule 2). *Publication* has three attributes: *PubID*, *PubType* and *ContactAuthor* and five simple elements: *Title*, *Year*, *Cost*, *ShippingCost* and *SpecialCost*; these eight are converted to simple fields (Mapping Rule 7). For the constraints, *Title*, *Year* and *Cost* have occurrence 1 and *ContactAuthor* is #REQUIRED; thus the not null constraint is applied (Row constraint rule 1). *PubID* is set as a primary key (Table constraint rule 1). *PubType* has a choice of attribute values; so the domain constraint is applied (Row constraint rule 4). For *Publication**, the rules for preserving the cardinality constraint are applied when updates are performed.

Secondly, three rlink-elements which are children of *Publication*: *Author*, *Publisher* and *Reference*, are handled as follows:

- Since the occurrence of *Author* is +, a separate table is created. This table consists of the primary keys of *Publication* and *Author* set as a composite key and foreign keys. *PubID* is defined with a delete cascade (Mapping rule 2 and Constraint rule 2 of `rlink`).
- Since the occurrences of *Publisher* and *Reference* are 1 and ? respectively, the primary keys of *Publisher* and *Reference* are added to *Publication* and are set as foreign keys (Mapping rule 1 and Constraint rule 1 of `rlink` are applied). The occurrence of *Publisher* is 1; thus the not null constraint is applied to the foreign key derived from *Publisher* (Constraint rule 4 of `rlink`)

Thirdly, *Authors* has only one complex child element: *Author*. Thus *Authors* is converted to an object table and *Author* is converted to an ADT (Mapping Rule 2). *Author* consists of one attribute: *AuthorID* and one simple element: *Email*; so these two are converted to simple fields (Mapping Rule 7). *Name* has siblings, occurrence 1, and all children as simple elements with occurrence ? or 1; thus *Name* is converted to an ADT (Mapping Rule 3). *Telephone* has siblings, occurrence *, all children as simple elements with occurrence 1 and there is no reference; hence *Telephone* is converted to a nested table (Mapping Rule 5). For the constraints, *AuthorID* is set as a primary key (Table constraint rule 1), *Name*, *FName*, *LName*, *Location* and *TelNo* have occurrence 1; thus the not null constraint is applied (Row constraint rule 1). For *Author** and *Telephone**, the rules for preserving the cardinality constraint are applied.

Fourthly, *Publishers* has only one complex child element: *Publisher*. Thus *Publishers* is converted to an object table and *Publisher* is converted to an ADT (Mapping rule 2). *Publisher* consists of one attribute: *PID* and two simple elements: *PName* and *SetupYear*; so these three are converted to simple fields (Mapping Rule 7). A complex child element, *Address,* has siblings, occurrence 1, and all children as simple elements with occurrence 1; thus *Address* is converted to an ADT (Mapping rule 3). For the constraints, *PID* is set as a primary key (Table constraint rule 1), *PName* and *SetupYear* have occurrence 1; thus the not null constraint is applied (Row constraint rule 1). For *Publisher**, the rules for preserving the cardinality constraint are applied.

Finally, *References* has only one complex child element: *Reference* so *References* is converted to an object table and *Reference* is converted to an ADT (Mapping Rule 2). *Reference* has two attributes: *RefID*, *RefType*, and one rlink-element: *Publication*, with occurrence +, referencing back to *Publication* in the document which cites it; thus *RefID* and *RefType* are converted to simple fields (Mapping Rule 7) and the not null constraint is applied (Row constraint rule 1). *RefType* has a default value and a choice of attribute values; so the default value and domain constraints (Row constraint rules 3-4) are applied and *RefID* is set as a primary key (Table constraint rule 1). *Publication* is the rlink-element whose occurrence is +; hence a separate table is created and this separate table consists of primary keys of *Reference* and *Publication* set as a composite key and foreign keys. The foreign key derived from *Reference* is defined with a delete cascade (Mapping rule 3 and Constraint rule 3 of `rlink` are applied). For *Reference**, the rules for preserving the cardinality constraint are applied.

A schema derived from our rules is created in Oracle 9i, assuming that the length of fields which are primary keys and foreign keys is 9 characters and the length of other fields is 30 characters. The schema shown in Figure 3.11 is generated manually.

```
Create Type TName as object (
  FName varchar2(30),
  MName varchar2(30),
  LName varchar2(30)
);

Create Type TTelephone as object
( Location varchar2(30),
  TelNo varchar2 (30)
);

Create Type NTTelephone as
table of TTelephone;

Create Type TAuthor as object (
  AuthorID varchar2(9),
  Name TName,
  Email varchar2(30),
  Telephone NTTelephone
);

Create Type TAuthors as object (
  Author TAuthor
);

Create Table Authors of TAuthors (
  Primary key (Author.AuthorID),
  Check (Author.Name is not null),
  Check (Author.Name.FName is not null
    and Author.Name.LName is not null)
)
nested table Author.Telephone STORE AS
Telephone_TAB(
  (CHECK  (Location is not null),
   CHECK (TelNo is not null))
);

Create Type TAddress as object (
  No varchar2(30) ,
  Street varchar2(30) ,
  City varchar2(30) ,
  Country varchar2(30),
  Zipcode varchar2(9)
);


Create Type TPublisher as object (
  PID varchar2(9),
  PName varchar2(30),
  Address TAddress,
  SetupYear varchar2(30)
);


Create Type TPublishers as object (
  Publisher TPublisher
);

Create Table Publishers of TPublishers
(
  Primary key (Publisher.PID),
  Check (Publisher.PName is not null
  and Publisher.SetupYear is not null)
);

Create Type TReference as
object (
  RefID varchar2(9),
  RefType varchar2(30)
);

Create Type TReferences as object (
  Reference TReference
);

Create Table References of TReferences (
  Primary key (Reference.RefID),
  Reference DEFAULT TReference
            (null, 'References'),
  CHECK (Reference.RefType IN
        ('References','Bibliography'))
);

Create Type TPublication as object (
  PubID varchar2(9),
  PubType varchar2(30),
  Title varchar2(30),
  Year varchar2(30),
  PID varchar2(9),
  RefID varchar2(9),
  Cost varchar2(30),
  ShippingCost varchar2(30),
  SpecialCost varchar2(30),
  ContactAuthor varchar2(9)
);

Create Type TPublications as object (
  Publication TPublication
);

Create Table Publications of TPublications (
  Primary key (Publication.PubID),
  CHECK (Publication.PubType IN
        ('book', 'article', 'journal')),
  CHECK (Publication.Title is not null and
         Publication.Year is not null and
         Publication.PID is not null and
         Publication.Cost is not null),
  Constraint p_a
      Foreign Key(Publication.ContactAuthor)
      references Authors(Author.AuthorID),
  Constraint p_p
      Foreign Key(Publication.PID)
      references Publishers(Publisher.PID),
  Constraint p_r
      Foreign Key(Publication.RefID)
      references References(Reference.RefID)
);

Create Table PublicationAuthor (
  PubID varchar2(9) references
  Publications(Publication.PubID)
               on delete cascade,
  AuthorID varchar2(9)
           constraint pa_a references
  Authors(Author.AuthorID),
  Primary key (PubID, AuthorID)
);

Create Table ReferencePublication (
  RefID varchar2(9) references
  References (Reference.RefID)
              on delete cascade,
  PubID varchar2(9)
        constraint rp_p references
  Publications (Publication.PubID),
  Primary key (RefID, PubID)
);
```

Figure 3.11 Schema generated in Oracle 9i

3.5 Summary

The aim of this chapter has been to present rules for mapping XML structure and constraints to an ORDB. In section 3.1, the `rlink` mechanism is shown to represent the relationship between elements from different XML documents. Thus non-redundant data can be kept in multiple separate documents. In section 3.2, firstly the collections of simplifying rules are described and, secondly, the rules for mapping an XML DTD and `rlink` to ORDB model are proposed. In section 3.3, mapping XML constraints and `rlink` constraints to ORDB constraints is categorised according to the types of constraints found in an (O)RDB. A method for handling the XML cardinality constraint is developed since no (O)RDB technology supports the cardinality constraint. Section 3.4 provides a case study to illustrate how to map XML structure and constraints to an ORDB by using our rules. The ORDB schema derived from our mapping rules is generated in Oracle 9i to show that the rules are practicable in the available ORDB technology.

In Chapter 4, an XML update language, an extension to XQuery, and its semantics will be presented. Then translation using the XML update language into SQL will be proposed.

Chapter 4: Updating XML documents

Chapter 3 described an approach for mapping XML schema to ORDB schema to store XML documents in an ORDB. In this chapter, updating XML documents which are kept in an ORDB will be presented. XML data is stored in the ORDB while users work on an XML document view; thus users should be able to update XML documents by an XML update language without the need to know how the XML data is stored in the ORDB. As a background process, this XML update language can be automatically translated into SQL executed on the ORDB.

When compared to existing XML query languages, XQuery is the most powerful, providing many features and it became the standard XML query language as announced by W3C [134]; however currently, there is no standard for an XML update language. Thus we design an XML update language, an extension to XQuery. In traditional database terminology e.g. SQL it is assumed that a query language includes update facilities but this is not the case with XML. Our XML update language supports some features that have not been found in existing XML update languages such as joins of documents in updating, updating multiple documents and updating data whose structure is recursive. However, the preservation of constraints is handled by the ORDB engine.

For translating XQuery into SQL, most existing work translates only the For-Let feature of XQuery into SQL. It has been determined that the recursive features of Quilt and XQuery cannot be translated into SQL [86, 87]. In this thesis, six important features inherited from XQuery will be translated into SQL: FLW(R|I|D) an abbreviation for a For-Let-Where-(Replace|Insert|Delete) expression, conditional expression, quantifier, aggregate functions, (non-recursive) user-defined function and recursive function.

The rest of this chapter is organised as follows. Section 4.1 presents an XML update language. Section 4.2 proposes techniques for translating our XML update language into SQL and section 4.3 presents how to translate the recursive function into SQL

procedure: a series of SQL commands equipped with a programming capability. Section 4.4 summarises this chapter.

4.1 An XML update language

In this thesis, update means to modify the content of XML documents not the schema; therefore any operation which changes the XML schema will be neglected (such as insertion of attributes or elements that have never been defined in the XML schema). This capability is the same as that in RDB which allows users to update data but not the underlying RDB schema. In this section, firstly, the syntax of the XML update language, an extension to XQuery is displayed and secondly the semantics of the five update operations supported by the update language is presented. Finally, some examples of the XML update language are shown.

4.1.1 The Syntax of the XML update language

The syntax of our XML update language is adapted from Tatarinov *et al* [124] and that of XQuery [134]. The syntax shown in Figure 4.1 is defined in EBNF (Extended Backus Naur Form).

```
(ForClause | LetClause)+
WhereUpdateClause | IfUpdateClause
where each clause is:
ForClause          ::= For $var in XPath(,$var in XPath)*
LetClause          ::= Let $var := XPath(,$var := XPath)*
WhereUpdateClause  ::= WhereClause? UpdateClause
WhereClause        ::= Where Condition
UpdateClause       ::= DeleteClause | ReplaceClause | InsertClause
DeleteClause       ::= Delete node WhereClause? (, Delete node WhereClause?)*
ReplaceClause      ::= Replace node with content WhereClause?
                       (, Replace node with content WhereClause?)*
InsertClause       ::= Insert content Into node WhereClause?
                         (Before|After condition basedon_XPath)?
                       (,Insert content Into node WhereClause?
                          (Before|After condition basedon_XPath)?)*
IfUpdateClause     ::= If Condition Then UpdateClause
                       (ElseIf Condition Then UpdateClause)*
                       (Else UpdateClause)?
```

Figure 4.1 The syntax of our XML update language

Our XML update language supports five update operations shown in the syntax of the language as follows:

1. Insert content Into node

   This operation will insert content into the target parent-node. The content can be elements, simple attributes or IDREF(s), whereas the target parent-node can be only elements.
2. Insert content Into node Before condition

   This operation will insert content into the target parent-node while the reference position of insertion can be specified by the Before clause. The content can be elements or IDREFs whereas the target parent-node can be only elements. The condition specified at the Before clause is an XPath expression.
3. Insert content Into node After condition

   This operation is the same as Insert...Before except that the reference position of the insertion is specified by the After clause.
4. Delete node

   This operation performs deep deletion which means that all descendants of the target deleted-node will be deleted too. The target deleted-node can be elements, simple attributes or IDREF(s). This operation does not allow deleting the root document since the document will not then be well-formed.
5. Replace node With content

   This operation will replace content of the target node with new content. The target node can be elements, simple attributes or IDREF(s).

4.1.2 Semantics of the XML update language

Before describing the semantics of the five update operations, the notations employed in the semantics will be introduced as follows.

Notations used in the semantics for the XML update language

1. **IDs**

   Stand for all IDs which are presented in XML documents with their names defined in DTD.
2. **Attrs**(node), **Eles**(node)

Stand respectively for names of all attributes of the node and names of all elements which are children of the node defined in DTD. The node can be only an element.

3. **name**(node)

   Stands for name of the node where the node can be an attribute, element and **IDs**.

4. **val**(node)

   Stands for value of the node where the node can be an attribute, element and **IDs**.

5. **IDval**(node), **IDREFval**(node)

   Stand for ID value and IDREF(s) value of the node respectively. The node can only be an element.

Semantics of the five update operations in the XML update language

1. Insert N Into K

   N can be an element, a simple attribute or an IDREF(s) whereas K can only be an element which is the target parent-element into which N will be inserted. We assume that K' is the result from K after the operation is performed.

In the case that N is an attribute whose type is not IDREFs
If $K = \{a_1, ..., a_n, e_1, ... , e_m\}$
where $a_1, ..., a_n$ are attributes and $e_1, ..., e_m$ are elements, and
N is an attribute = a_{n+1} and **name**(a_{n+1}) $\in$ **Attrs**(K)
Then $K' = \{a_1, ..., a_n, a_{n+1}, e_1, ..., e_m\}$

In the case that N is an attribute whose type is IDREFs
If $K = \{a_1, ..., C,..., a_n, e_1, ..., e_m\}$
where $a_1, ..., C,..., a_n$ are attributes and $e_1,..., e_m$ are elements and
C and N are attributes whose type is IDREFs and
If **name**(N) = **name**(C), **val**(C) $=\{id_1,..., id_k\}$,
val(N) = id_{k+1}, **val**(N) $\notin$ **val**(C) and **val**(N) $\in$ **val(IDs)**
Then **val**(C') = $\{id_1,..., id_k, id_{k+1}\}$
where C' is the result of C after the operation is performed
Then $K' = \{a_1, ..., C', ..., a_n, e_1, ..., e_m\}$

In the case that N is an element

If $K = \{a_1, ..., a_n, e_1, ..., e_m\}$

where $a_1, ..., a_n$ are attributes and $e_1, ..., e_m$ are elements and

N is an element = e_{m+1},

name(e_{m+1}) ∈ **Eles**(K), **IDval**(e_{m+1}) ∉ **val(IDs)** and

IDREFval(e_{m+1}) ∈ **val(IDs)**

Then $K' = \{a_1, ..., a_n, e_1, ..., e_m, e_{m+1}\}$

2. Insert N ... Into K ... Before M

N can be an element or an attribute whose type is IDREFs whereas K can be only an element which is the target parent-element into which N will be inserted. M is a sibling of N. We assume that K′ is the result from K after the operation is performed.

In the case that N is an attribute whose type is IDREFs

If $K = \{a_1, ..., C, ..., a_n, e_1, ..., e_m\}$

where $a_1, ..., C, ..., a_n$ are attributes and $e_1, ..., e_m$ are elements

M, C and N have type: IDREFs and

If **name**(M) = **name**(C) = **name**(N),

val(C) = $\{id_1, ..., id_i, ..., id_k\}$ and **val**(M) = id_i

Then **val**(M) ∈ **val**(C)

If **val**(N) = id_{i-1} and

val(N) ∉ **val**(C) and **val**(N) ∈ **val(IDs)**

Then **val**(C′) = $\{id_1, ..., id_{i-1}, id_i, ..., id_k\}$

where C′ is the result of C after the operation is performed

Then $K' = \{a_1, ..., C', ..., a_n, e_1, ..., e_m\}$

In the case that N is an element

If $K = \{a_1, ..., a_n, e_1, ..., e_i, ..., e_m\}$

where $a_1, ..., a_n$ are attributes and $e_1, ..., e_i, ..., e_m$ are elements

M is an element = e_i, N is an element = e_{i-1},

name(e_{i-1}) ∈ **Eles**(K), **IDval**(e_{i-1}) ∉ **val(IDs)** and

IDREFval(e_{i-1}) ∈ **val(IDs)**

Then $K' = \{a_1, ..., a_n, e_1, ..., e_{i-1}, e_i, ..., e_m\}$

3. Insert N ... Into K ... After M

N can be an element or an attribute whose type is IDREFs whereas K can be only an element which is the target parent-element into which N will be inserted. M is a sibling of N. We assume that K′ is the result from K after the operation is performed.

In the case that N is an attribute whose type is IDREFs

If $K = \{a_1,..., C, \ldots, a_n, e_1, ..., e_m\}$

where $a_1, ...C, \ldots, a_n$ are attributes and $e_1,\ldots, e_m$ are elements

M, C and N have type: IDREFs and

If **name**(M) = **name**(C) = **name**(N) and

val(C) $=\{id_1,\ldots, id_i,\ldots, id_k\}$ and **val**(M) = id_i

Then **val**(M) ∈ **val**(C)

If **val**(N) = id_{i+1} and

val(N) ∉ **val**(C) and **val**(N) ∈ **val(IDs)**

Then **val**(C′) $=\{id_1, \ldots, id_i, id_{i+1}, \ldots, id_k\}$

where C′ is the result of C after the operation is performed

Then $K' = \{a_1, ..., C', \ldots, a_n, e_1, ..., e_m\}$

In the case that N is an element

If $K = \{a_1, ..., a_n, e_1, ..., e_i, \ldots, e_m\}$

where $a_1, \ldots, a_n$ are attributes and $e_1, ..., e_i, \ldots, e_m$ are elements

M is an element = e_i, N is an element = e_{i+1},

name(e_{i+1}) ∈ **Eles**(K), **IDval**(e_{i+1}) ∉ **val(IDs)** and

IDREFval(e_{i+1}) ∈ **val(IDs)**

Then $K' = \{a_1, \ldots, a_n, e_1, \ldots, e_i, e_{i+1}, \ldots, e_m\}$

4. Delete N

N can be an element, a simple attribute or an IDREF(s). We assume that K is the target parent-element from which N will be deleted. K′ is the result from K after the operation is performed.

In the case that N is an attribute whose type is not IDREFs

If $K = \{a_1, ..., a_{n-1}, a_n, e_1, \ldots, e_m\}$

where $a_1, ..., a_{n-1}, a_n$ are attributes and $e_1, ..., e_m$ are elements and

N is an attribute = a_n

Then $K' = \{a_1, \ldots, a_{n-1}, e_1, ..., e_m\}$

In the case that N is an attribute whose type is IDREFs

If $K = \{a_1, ..., C, \ldots, a_n, e_1, ..., e_m\}$

where $a_1, ...C, \ldots, a_n$ are attributes and $e_1, \ldots, e_m$ are elements and

N and C have type: IDREFs and

If **name**(N) = **name**(C), **val**(C) $=\{id_1, \ldots, id_{k-1}, id_k\}$ and **val**(N) = id_k

then **val**(N) $\in$ **val**(C) and **val**(Ć) = $\{id_1, \ldots, id_{k-1}\}$

where C' is the result of C after the operation is performed

Then $K' = \{a_1, ..., C', \ldots, a_n, e_1, \ldots, e_m\}$

In the case that N is an element

If $K = \{a_1, ..., a_n, e_1, ..., e_{m-1}, e_m\}$

where $a_1, ..., a_n$ are attributes and $e_1, ..., e_{m-1}, e_m$ are elements and

N is an element = e_m

Then $K' = \{a_1, \ldots, a_n, e_1, \ldots, e_{m-1}\}$

5. Replace N with M

N can be an element, a simple attribute or an IDREF(s). N and M have the same type. We assume that K is the parent-element of N which will be updated. K' is the result from K after the operation is performed.

In the case that N is an attribute whose type is not IDREFs

If $K = \{a_1, ..., a_{i, \ldots,} a_n, e_1, \ldots, e_m\}$

where $a_1, ..., a_i, \ldots, a_n$ are attributes and $e_1, \ldots, e_m$ are elements, and

val(K) = $\{x_1, \ldots, x_i, \ldots, x_n, y_1, \ldots, y_m\}$ and

N is an attribute = a_i , **name**(N) = **name**(M), **val**(N) = x_i and **val**(M) = v_i

Then **val**(K') = $\{x_1, \ldots, v_i, \ldots, x_n, y_1, \ldots, y_m\}$

In the case that N is an attribute whose type is IDREFs

If $K = \{a_1, ..., C, \ldots, a_n, e_1, ..., e_m\}$

where $a_1, ..., C, \ldots, a_n$ are attributes and $e_1, \ldots, e_m$ are elements

N, C and M have type: IDREFs and

If **name**(N) = **name**(C) = **name**(M) and

val(C) $=\{id_1, \ldots, id_i, \ldots, id_k\}$ and **val**(N)= id_i then **val**(N) $\in$ **val**(C)

If **val**(M)= id_x and $id_x \in$ **val(IDs)**

then **val**(C') = $\{id_1, \ldots, id_x, \ldots, id_k\}$

where C' is the result of C after the operation is performed

Then $K' = \{a_1, ..., C', \ldots, a_n, e_1, \ldots, e_m\}$

In the case that N is an element

If K = $\{a_1, ..., a_n, e_1, ... e_{i,...}, e_m\}$

where $a_1, ... , a_n$ are attributes and $e_1, ..., e_i, ..., e_m$ are elements and

val(K) = $\{x_1,..., x_n, y_1, ..., y_i, ..., y_m\}$ and

N is an element = e_i, **name**(N) = **name**(M), **val**(N) = y_i and **val**(M) = w_i

Then **val**(K′) = $\{x_1,..., x_n, y_1, ..., w_i, ..., y_m\}$

Note: If N contains an ID then **val**(ID) will not be changed and will be the **val**(ID) of M.

4.1.3 Examples of the XML update language

There are two new operators used in the XML update language for `rlink` as follows. The first operator is the link operator (~>) used to dereference `rlink` when elements are navigated across documents. For example, navigating *Author* in *Authors.xml* referenced by *@rlink-href* of *Author* in *Publications.xml* is expressed as follows:

doc("Publications.xml")//Author~>//Author

The second new operator is the rlink( ) operator. This is used for the convenience of assigning an XPath expression to the *@rlink-href*. Thus in the XML update language,

element@rlink-href = "XPath" is rewritten as rlink(element, XPath)

For example

$p/Author@rlink-href = "//Author[@AuthorID = 'A222']" is written as:

$p/rlink(Author, //Author[@AuthorID = "A222"])

Some examples of using the XML update language for updating a single XML document, based on the DTD in section 7.2.1 and multiple linked XML documents, based on the DTDs in section 3.4 are shown in Figures 4.2 and 4.3 respectively.

```
For $p in doc ("Library.xml")/Library/Publication
Where $p/Title = "Java 2"
and   $p/Author/Name/LName = "Smith"
Replace $p/Author with
          <Author>
                  <Name>                                          (a)
                        <FName>Mc</FName>
                        <LName>Donal</LName>
                  </Name>
                  ........
          </Author>
-------------------------------------------------------------------
For $p in doc ("Library.xml")//Publication,
    $r in $p/Reference
Where $p/Title = "Java"                                           (b)
Delete $p/Year,
Delete $r@RefPub
Where $r@RefPub = "P222"
-------------------------------------------------------------------
For $p in doc ("Library.xml")/Library/Publication,
      $rp in $p/Reference/@RefPub    Publication                  (c)
Where $p/Title = "XQuery"
Insert  <Year>2004</Year>  into $rp  →
```

Figure 4.2 Using the XML update language for a single XML document

```
For $p in doc ("Publications.xml")//Publication,
    $a in $p/Author ~>/Authors
Where   $p/Title = "Java 2"
and     $a/Author/Name/LName = "Smith"
Replace $a/Author with
          <Author>
                  <Name>                                          (a)
                        <FName>Mc</FName>
                        <LName>Donal</LName>
                  </Name>
                  ......
          </Author>
-------------------------------------------------------------------
For $p in doc ("Publications.xml")//Publication,
       $r in $p/Reference ~>//Reference
Where $p/Title = "Java"                                           (b)
Delete $p/Year,
Delete $r/Publication
Where $r/rlink(Publication, //Publication[@PubID = "P222"])
-------------------------------------------------------------------
For $p in doc ("Publication.xml")/Publications/Publication,
    $rp in $p/Reference~>//Reference/Publication~>//Publication
Where $p/Title = "XQuery"                                         (c)
Insert  <Year>2004</Year>  into $rp
```

Figure 4.3 Using the XML update language for multiple linked XML documents

The commands in Figures 4.2(a) and 4.3(a) replace an *Author*, whose LName is 'Smith', of the *Publication* whose *Title* is 'Java 2'. The commands in Figures 4.2(b) delete *RefPub* of the *Reference* (linking to *PubID* = 'P222') and delete *Year* of the *Publication* whose *Title* is 'Java'. The commands in Figures 4.3(b) delete an rlink-element: *Publication* of the *Reference* (linking to *PubID* = 'P222') and delete *Year* of the *Publication* whose *Title* is 'Java'.

The commands in Figure 4.2(c) and 4.3(c) insert *Year* into the *Publications* which are direct references to the *Publication* whose *Title* is 'XQuery'.

4.2 Update Language Translation

XQuery is the most powerful and hence it became the standard XML query language [134]; therefore our XML update language is an extension of XQuery. In this thesis, six important features of our XML update language inherited from XQuery will be translated into SQL: FLW(R|I|D) expression, conditional expression, quantifier, aggregate functions, non-recursive user-defined function and recursive function. This section will translate the first five features while the last feature, the recursive function, will be translated in the next section.

XQuery is a functional language whereas SQL is a declarative language; therefore translating the XML update language, an extension to XQuery, into SQL is not straightforward. Thus several techniques such as rewrite rules and optimisation will be used during the translation. In this section, four techniques for translating the XML update language will first be described. Then, the steps for translating the XML update language into SQL will be proposed and finally an example will be presented.

4.2.1 Four techniques for translating the XML update language

Our translation uses four main techniques: update/delete join commands (joins in update/delete commands), rewrite rules, graph mapping and optimisation rules for translating the XML update language into SQL. These four techniques are given below.

Update/delete join commands (joins in update/delete commands)

In the SQL standard, update/delete join commands cannot be performed; however, the translation of XML update commands can result in the joins of several tables. Therefore we translate XML update commands into update/delete join commands and then rewrite them as SQL with sub-query commands. The syntax of update/delete join commands is as follows:

- Syntax of joins in update command

  Update *table whose data will be updated*
  From *all related tables*
  Set *field1 = value1, field2 = value2, ...*
  Where *Condition*;

- Syntax of joins in delete command

  Delete *table whose data will be deleted*
  From *all related tables*
  Where *Condition*;

Rewrite Rules

There are seven categories of rewrite rules: FLW(R|I|D) expression, aggregate function, quantifier, conditional expression, (non-recursive) user-defined function, rlink and SQL rewrite rules. The first five categories are classified according to features of the update language and these features will be rewritten as SQL functions. The sixth category is used to rewrite `rlink` as SQL functions. The last category is used to rewrite update/delete join commands as SQL with sub-query commands. In this section, we explain the SQL functions and then describe the seven categories of the rewrite rules.

I) SQL Functions

In the translation of the language, all clauses of XML update commands must be rewritten as SQL functions. The SQL functions are conceptual functions representing the operations of SQL. The SQL functions are used to group XML update clauses and their conditions together since one XML update command can consist of several update clauses and each update clause can have its own condition. Thus these update clauses are grouped by using the function number (funcNo) which is a parameter of every SQL function. The funcNo 0 will be assigned to ForClause, LetClause and WhereClause of the ForClause and LetClause. These clauses will be shared clauses for the UpdateClause. Each update clause will have its own funcNo, a running number starting from 1. The update clause and its own condition will have the same funcNo.

The SQL functions are as follows:

```
1. bindF(path, $var, funcNo)
2. bindL(path, $var, funcNo)
3. insert (node, value | :funcNo, funcNo)
4. delete(node, funcNo)
```

```
5. update(node, value | :funcNo, funcNo)
6. where|LogicalOper(node, CompOper, value | :funcNo, funcNo)
7. group_by(node, funcNo)
8. aggFunc(node, funcNo)
   where aggFunc ::=max | min | count | avg | sum
9. having(aggFunc(node), CompOper, value | :funcNo, funcNo)
10. select(node, funcNo)
```

Four SQL functions, insert(), update(), where|LogicalOper() and having(), have the parameter *value | :funcNo* since sometimes the value in inserting, in updating or in the predicate, is not a constant but may come instead from selecting a value from other nodes. Hence in this case, funcNo has the same number as that for the funcNo of the select() function. The symbol ':' is used to differentiate the funcNo of an SQL function from funcNo as the value-parameter.

II) Rewrite rules for FLW(R|I|D)

The expression FLW(R|I|D) will be rewritten as SQL functions as follows:

1. `For $var in XPath` is rewritten as:
   `bindF(XPath, $var, funcNo)`
2. `Let $var := XPath` is rewritten as:
   `bindL(XPath, $var, funcNo)`
3. `Where predicate` is rewritten as:
   `where(node, ComparisonOperator, value|:funcNo, funcNo)`
4. `LogicalOper predicate` is rewritten as:
   `LogicalOper(node, ComparisonOperator, value|:funcNo, funcNo)`
5. $\texttt{For \$var in XPath}_{\texttt{predicate}}$ is translated into:
   `For $var in XPath Where predicate`
   Then this clause is rewritten as SQL functions according to rules 1, 3, 4.
6. $\texttt{Let \$var := XPath}_{\texttt{predicate}}$ is translated into:
   `Let $var := XPath Where predicate`
   Then this clause is rewritten as SQL functions according to rules 2-4.
7. `Select` *`node`* `| Return` *`node`* is rewritten as:
   `select(node, funcNo)`
8. `Replace` *`node`* `with` *`simple content`* is rewritten as:
   `update(node, content value, funcNo)`

9. Delete *node* is rewritten as:

```
delete(node, funcNo)
```

10. Insert *simple content* into *node* is rewritten as:

```
Insert(node/content-node, content value, funcNo)
```

11. Replace *node* with *complex content*

The *complex content* is shredded into many simple contents. The Replace command is rewritten in the form of the commands based on the simple contents which are in turn rewritten as SQL functions as follows:

Define: complex content $e_c = \{e_1, e_2,\ldots, e_{i-1}, e_i, a_1, a_2,\ldots, a_i\}$
where $e_1, e_2,\ldots, e_{i-1}, e_i$ are elements, $a_1, a_2,\ldots, a_i$ are attributes and

e_{i-1} is rlink-element, $e_i = \{e_{i1}\{e_{i2}..\{e_{ii}\}..\}, a_{i1}, a_{i2},\ldots, a_{ii}\}$ and

$v_{e1}, v_{e2},\ldots,v_{ei-1}, v_{a1}, v_{a2},\ldots, v_{ai}$ are values of $e_1, e_2,\ldots,e_{i-1}, a_1, a_2,..., a_i$ respectively and

$v_{eii}, v_{ai1}, v_{ai2},\ldots, v_{aii}$ are values of $e_{ii}, a_{i1}, a_{i2},\ldots, a_{ii}$ respectively.

Replace *node* with e_c is rewritten as:

```
update(ec, , funcNo)
update(ec/e1, ve1, funcNo)
update(ec/e2, ve2, funcNo)
...
replace ec/ei-1 with rlink(ei-1, vei-1)  (rlink rewrite rules are applied)
update(ec@a1, va1, funcNo)
update(ec@a2, va2, funcNo)
...
update(ec@ai, vai, funcNo)
update(ec/ei, , funcNo)
update(ec/ei/ei1, , funcNo)
update(ec/ei/ei1/ei2, , funcNo)
....
update(ec/ei/ei1/ei2/.../eii, veii, funcNo)
update(ec/ei@ai1, vai1, funcNo)
update(ec/ei@ai2, vai2, funcNo)
...
update(ec/ei@aii, vaii, funcNo)
```

12. Insert *complex content* Into *node*

The *complex content* is shredded into many simple contents. The Insert command is rewritten in the form of the commands based on the simple contents which are in turn rewritten as SQL functions as follows:

Define: complex content $e_c = \{e_1, e_2,\ldots, e_{i-1}, e_i, a_1, a_2,\ldots, a_i\}$
where $e_1, e_2,\ldots, e_{i-1}, e_i$ are elements, $a_1, a_2,\ldots, a_i$ are attributes and

e_{i-1} is rlink-element, $e_i = \{e_{i1}\{e_{i2}..\{e_{ii}\}..\}, a_{i1}, a_{i2},\ldots, a_{ii}\}$ and

$v_{e1}, v_{e2},\ldots,v_{ei-1}, v_{a1}, v_{a2},\ldots, v_{ai}$ are values of $e_1, e_2,\ldots,e_{i-1}, a_1, a_2,..., a_i$ respectively and

$v_{eii}, v_{ai1}, v_{ai2},\ldots, v_{aii}$ are values of $e_{ii}, a_{i1}, a_{i2},\ldots, a_{ii}$ respectively.

Insert e_c Into *node* is rewritten as:

insert(node/e_c, , funcNo)
insert(node/e_c/e_1, v_{e1}, funcNo)
insert(node/e_c/e_2, v_{e2}, funcNo)
...
insert e_c/rlink(e_{i-1}, v_{ei-1}) into node *(rlink rewrite rules are applied)*
insert(node/e_c@a_1, v_{a1}, funcNo)
insert(node/e_c@a_2, v_{a2}, funcNo)
...
insert(node/e_c@a_i, v_{ai}, funcNo)
insert(node/e_c/e_i, , funcNo)
insert(node/e_c/e_i/e_{i1}, , funcNo)
insert(node/e_c/e_i/e_{i1}/e_{i2}, , funcNo)
....
insert(node/e_c/e_i/e_{i1}/e_{i2}/.../e_{ii}, v_{eii}, funcNo)
insert(node/e_c/e_i@a_{i1}, v_{ai1}, funcNo)
insert(node/e_c/e_i@a_{i2}, v_{ai2}, funcNo)
...
insert(node/e_c/e_i@a_{ii}, v_{aii}, funcNo)

III) Rewrite rules for aggregate functions

1. Define: For $var1 in XPath1
   Let $var2 := $var1/XPath2
   Then: aggFunct($var2) is rewritten as:
   aggFunct($var2, funcNo)
   group_by($var1, funcNo)
2. Define: Let $var := XPath
   Then: aggFunct($var) is rewritten as:
   aggFunct($var, funcNo)
3. Define: For $var1 in XPath1
   Let $var2 := $var1/XPath2
   Then: Where aggFunc($var2) ComparisonOperator value is rewritten as:
   group_by($var1, funcNo)
   having(aggFunct($var2), ComparisonOperator, value, funcNo)

IV) Rewrite rules for quantifier

In XQuery, there are two quantifiers: existential quantifier (some) and universal quantifier (every). Both quantifiers can be translated into a count() function since the existential quantifier is used to test whether at least one item in a sequence satisfies the condition while the universal quantifier is used to test whether every item in a sequence

satisfies the condition; thus before rewriting these quantifiers to SQL functions, their meanings will first be translated and then rewritten as SQL functions as follows:

1. For $var1 in XPath1
   Where some $var2 in $var1/XPath2
   Satisfies (Condition) is translated into:

   For $var1 in XPath1
   Let $var2 := $var1/XPath2
   Where count($var2) > 0
   And Condition and is rewritten as:

   $var1 = bindF(XPath1, funcNo)
   $var2 = bindL($var1/XPath2, funcNo)
   where (node, ComparisonOperator, value, funcNo)
   (LogicalOper(node,ComparisonOperator, value, funcNo))*
   group_by($var1, funcNo)
   having(count($var2), >, 0, funcNo)

2. For $var1 in XPath1
   Where every $var2 in $var1/XPath2
   Satisfies (Condition) is translated into:

   For $var1 in XPath1
   Let $var2 := $var1/XPath2
   Where Condition
   And count($var2) = (For $var3 in XPath1
   Let $var4 := var3/XPath2
   Where $var3 = $var1
   Return count($var4)
) and is rewritten as:

   $var1 = bindF(XPath1, funcNo)
   $var2 = bindL(var1/XPath2, funcNo)
   where (node,ComparisonOper, value, funcNo)
   (LogicalOper (node, ComparisonOper, value, funcNo))*
   group_by($var1, funcNo)
   having(count($var2), =, :1, funcNo)
   $var3 = bindF(XPath1, 1)
   $var4 = bindL($var3/XPath2, 1)
   select(count($var4), 1)
   where ($var3, =, $var1, 1)
   group_by($var3, 1)

Besides 'some' and 'every' quantifiers, there are two functions: empty() and exists() which can be rewritten as a count() function. These functions and quantifiers can be used along with 'not'. To summarise, the meaning of these functions and quantifiers can be translated before rewriting as follows:

some	is translated into	$count_{predicate} > 0$
not (some)	is translated into	$count_{predicate} = 0$
every	is translated into	$count_{predicate} = count_{without\ predicate}$
not (every)	is translated into	$count_{predicate} < count_{without\ predicate}$ and $count_{predicate} > 0$
empty	is translated into	$count_{predicate} = 0$
not (empty)	is translated into	$count_{predicate} > 0$
exists	is translated into	$count_{predicate} > 0$
not (exists)	is translated into	$count_{predicate} = 0$

V) Rewrite rules for the conditional expression

The construction:

(ForClause|LetClause)+
If (Condition$_1$) then
UpdateStm$_1$
Else If (Condition$_2$) then
UpdateStm$_2$
…….
[
Else [If (Condition $_n$)]
UpdateStm$_n$
]

is translated into a series of commands as follows:

(ForClause|LetClause)+
Where Condition $_1$
UpdateStm$_1$
(ForClause|LetClause)+
Where Condition$_2$
And ¬(Condition$_1$)
UpdateStm$_2$
……
[
(ForClause|LetClause)+
[Where condition$_n$]
Where|And ¬ (condition$_1$)
And ¬ (condition$_2$)
…….
And ¬ (condition$_{n-1}$)
UpdateStm$_n$
]

The series of commands is then rewritten as SQL functions according to the category of expressions. The number of commands in the series corresponds to the number of conditions in if-then-else. The symbol ¬ stands for 'not'.

VI) Rewrite rules for non-recursive user-defined functions

Calls to non-recursive functions are replaced with the body of such functions and parameters of the function are replaced with the values of arguments. After such replacements, the update command is rewritten as SQL functions according to the category of expressions in the command.

VII) Rewrite rules for `rlink`

Define: rlinkEle is the rlink-element containing *@rlink-href* whose value is XPath
parentEle is the parent of the rlinkEle
linkedEle is the element referenced by XPath which is the value of *@rlink-href*

1. *Where* predicate based on rlink-element

   If there is only one predicate in XPath based on PK of linkedEle as follows:

   PK of linkedEle = value

   Then Where parentEle/rlink(rlinkEle, XPath) is rewritten as:

   where(parentEle/rlinkEle#linkedEle, =, value, funcNo)

   Else Where parentEle/rlink(rlinkEle, XPath) is rewritten as:

   where(parentEle/rlinkEle#linkedEle, in, :funcNo2, funcNo1)
   select(PK of linkedEle, funcNo2)
   where(condition in XPath, funcNo2)

2. *Insert* rlink-element

   If there is only one predicate in XPath based on PK of linkedEle as follows:

   PK of linkedEle = value

   Then Insert rlink(rlinkEle, XPath) Into parentEle is rewritten as:

   insert(parentEle/rlinkEle#linkedEle, value, funcNo)

   Else Insert rlink(rlinkEle, XPath) Into parentEle is rewritten as:

   insert(parentEle/rlinkEle#linkedEle, :funcNo2, funcNo1)
   select(PK of linkedEle, funcNo2)
   where(condition in XPath, funcNo2)

3. *Replace* rlink-element is translated into a sequence of *delete* and *insert* rlink-element with the sequence rewritten as SQL functions according to rules 1-2

   Replace parentEle/rlinkEle with rlink(rlinkEle, XPath1)
   [Where parentEle/rlink(rlinkEle, XPath2)] is translated into :

   Delete parentEle/rlinkEle
   [Where parentEle/rlink(rlinkEle, XPath2)],
   Insert rlink(rlinkEle, XPath1) Into parentEle

Then *Where* and *Insert* clauses are rewritten according to the rules 1-2.

VIII) SQL rewrite rules (rules for joins in update and delete commands)

These rules are used to rewrite joins in update commands and joins in delete commands as SQL with sub-query commands. The rewrite rules for joins in update commands and in delete commands are shown in Figures 4.4 and 4.5 respectively.

```
If   T1 is table                                                    Then
     Update T1
     From all related tables
     Set field1 = value, field2 = value, ...
     Where Condition                                              is rewritten to:

     Update T1
     Set field1 = value, field2 = value, ...
     Where PK(T1) in (select PK(T1) from all related tables where Condition)

Else If  T1 is separate table derived from IDREFs or rlink    Then
     Define:
     T2 is table containing primary key(PK) referenced by foreign key (FK1) of T1
     T3 is table containing primary key(PK) referenced by foreign key (FK2) of T1
     value1, value2 are constant values

     If  predicate of T1.FK1 is T1.FK1 = value1  OR
         predicate of T1.FK2 is T1.FK2 = value2               Then
          Update T1
          From all related tables
          Set FK1 = value, FK2 = value
          Where Condition
          And T1.FK1 = value1 | T2.PK
          And T1.FK2 = value2 | T3.PK                         is rewritten to:

          Update T1
          Set FK1 = value, FK2 = value
          Where T1.FK1 ( = value1 |
                       in (Select T2.PK From all related tables except T1
                           Where Condition without join to T1))
          And   T1.FK2 ( = value2 |
                       in (Select T3.PK From all related tables except T1
                           Where Condition without join to T1))

     Else If  predicates on T1.FK1 and T1.FK2 are not constant values  Then
          Update T1
          From all related tables
          Set FK1 = value, FK2 = value
          Where Condition
          And T1.FK1 = T2.PK
          And T1.FK2 = T3.PK                          is rewritten to:

          Update T1
          Set FK1 = value, FK2 = value
          Where T1.FK1 in (Select T2.PK From all related tables except T1 and T3
                           Where Condition without join to T1
                           and except predicates on T3)
          And   T1.FK2 in (Select T3.PK From all related tables except T1 and T2
                           Where Condition without join to T1
                           and except predicates on T2)
     EndIf
EndIf
```

Figure 4.4 Rewrite rules for joins in update commands

```
If   T1 is table                                                    Then
     Delete T1
     From all related tables
     Where Condition                                                is rewritten to:

     Delete From T1
     Where PK(T1) in (select PK(T1) from all related tables where Condition)

Else If  T1 is separate table derived from IDREFs or rlink    Then
     Define:
     T2 is table containing primary key(PK) referenced by foreign key (FK1) of T1
     T3 is table containing primary key(PK) referenced by foreign key (FK2) of T1
     value1, value2 are constant values

     If  predicate of T1.FK1 is T1.FK1 = value1  OR
         predicate of T1.FK2 is T1.FK2 = value2                 Then
          Delete T1
          From all related tables
          Where Condition
          And T1.FK1 = value1 | T2.PK
          And T1.FK2 = value2 | T3.PK                           is rewritten to:

          Delete From T1
          Where T1.FK1 ( = value1 |
                       in (Select T2.PK From all related tables except T1
                           Where Condition without join to T1))
          And   T1.FK2 ( = value2 |
                       in (Select T3.PK From all related tables except T1
                           Where Condition without join to T1))

    Else If  predicates on T1.FK1 and T1.FK2 are not constant values   Then
          Delete T1
          From all related tables
          Where Condition
          And T1.FK1 = T2.PK
          And T1.FK2 = T3.PK                                    is rewritten to:

          Delete From T1
          Where T1.FK1 in (Select T2.PK From all related tables except T1 and T3
                           Where Condition without join to T1
                           and except predicates on T3)
          And  T1.FK2  in (Select T3.PK From all related tables except T1 and T2
                           Where Condition without join to T1
                           and except predicates on T2)
     EndIf
EndIf
```

Figure 4.5 Rewrite rules for joins in delete commands

Graph Mapping

The purpose of graph mapping is to indicate the SQL functions performed on tables, (field of) nested table, (field of) abstract data type or simple fields of the table, so that SQL commands can be generated from the graph correctly.

The steps for graph mapping start from creating a graph whose paths correspond to paths in the SQL functions. Then the graph is mapped to the database schema graph to identify which node is a table, (field of) nested table, (field of) abstract data type or simple field. The keys for joins of tables and join symbols are then added to the graph and the SQL functions are mapped to the graph. Next, pushing the function down to

proper nodes of the graph and changing the meaning of update operations, are performed according to the update-rules in section 4.2.2. The graph may then be split into several sub-graphs. The number of sub-graphs is equal to the number of update operations performed on the tables which have a different function number.

Finally, optimisation rules are applied to the graph or the sub-graphs and SQL commands or update/delete join commands are generated from the graph or the sub-graphs.

Optimisation Rules

During the translation of the XML update language, the optimisation is performed to remove unnecessary tables in joins. The optimisation is based on ORDB structure types such as table and field and the appearance of SQL operations in these structure types.

There are three techniques for optimisation as follows:

1. Eliminate unnecessary prior nodes: this technique is performed by traversing from the root node of the graph until it finds the first predicate or update operation on a table or a field. Nodes which are prior to the table or the table of the field can then be eliminated from the graph.
2. Eliminate the join of any two contiguous tables:

   Define: T1 and T2 are two contiguous tables starting from the root of the graph.

   PK stands for primary key and FK stands for foreign key.

   On the graph, if T1 consists of only one field which is PK/FK linking to FK/PK of T2 and P is a predicate on PK/FK of T1, then P can be moved to FK/PK of T2 and T1 and its PK/FK can be eliminated from the graph.

   This means that:

   a. If the first table has a foreign/primary key linking to the primary/foreign key of the second table and
   b. If there is a predicate on the primary/foreign key of the first table but there is no other fields of the first table either related to predicates or acting as target fields which will be updated.

   Then the first table can be eliminated from the graph and the predicate on the primary/foreign key of the eliminated table is added to the corresponding foreign/primary of the second table.

3. Eliminate the join of any three contiguous tables:

   Define: T1, T2 and T3 are three contiguous tables starting from the root of the graph. PK stands for primary key and FK stands for foreign key.

   On the graph, if T2 consists of only one field which is PK/FK linking to FK/PK of T1 and FK/PK of T3, then T1 and T3 can be joined together directly and T2 and its PK/FK can be eliminated from the graph. If there is a predicate on PK/FK of T2 then the predicate will be moved to FK/PK of T3.

   This means that:

   a. If the second table has a primary/foreign key linking to the foreign/primary key of both the first table and the third table, and
   b. If there is no other fields of the second table either related to predicates or acting as target fields which will be updated.

   Then the second table can be eliminated from the graph, and the foreign/primary key of the first and the third table is used for direct join instead of joins via the primary/foreign key of the second table. If there is a predicate on the primary/foreign key of the second table then move the predicate to the foreign/primary key of the third table.

4.2.2 Steps for translating the XML update language into SQL

The steps for translating the XML update language into SQL are given below:

1. Rewrite every clause in the update command as SQL functions according to the rewrite rules.
2. Create a graph whose paths correspond to paths in the functions
3. Map the graph to the database schema graph to identify which node is a table, (field of) nested table, (field of) abstract data type or simple field.
4. Add key fields (PK and FK) which are used to join tables. However, key fields in the case of recursion on the path of the command (keys of elements referencing back to ancestors in the path of the command) will not be added. Then add the join symbols (L1, L2, ..., Ln) to indicate which pair of the key fields is used to join the tables.
5. Map the functions to the graph. The meaning of operations performed on the XML structure may be changed when the ORDB structure is mapped to the XML structure. In some cases, the functions are needed to push down to another node.

Update rules for changing meaning of the operations and pushing down the functions are defined as follows.

Update-rule 1: If a *delete* or *insert* function is performed on nodes converted to ADTs having siblings, fields of an ADT, fields of a table or fields of a nested table, without a *delete* or *insert* function on an ancestor-node converted to a table or an ADT without a sibling, then the function will be converted to an *update* function. If an *update* function is performed on these nodes, the *update* function will not be converted.

Update-rule 2: If a *delete*, *insert* or *update* function is performed on nodes converted to a nested table, then the function will not be converted.

Update-rule 3: If a *delete*, *insert* or *update* function is performed on a node converted to a table or an ADT without a sibling, then this indicates that the function will delete, insert or update a row of the table. In this case the function will not be converted.

Update-rule 4: If an *insert* function is performed on a node converted to the primary key of a table, then this *insert* function must be copied to the foreign key of child-tables to maintain parent-child relationships.

Update-rule 5: If a *select*, *where*, *group_by* or *aggregate* function is performed on a node converted to a table or an ADT without a sibling, then the function will be pushed down to the primary key of the table.

Update-rule 6: If a *select* function is performed on an element referenced from its descendant (recursive path), then the function will be pushed down to the field of the descendant linking to the element.

6. In the case that there is more than one *update* function on tables, the graph will be split into sub-graphs. The number of sub-graphs is equal to the number of update operations performed on the tables which have different funcNo.
7. Optimise each (sub-)graph according to optimisation rules.
8. Generate SQL commands or update/delete join commands from each (sub-)graph. The *bindF/bindL* functions will be ignored in generating the commands.
9. If the generated commands are in the form of update/delete join commands, the commands are rewritten according to the SQL rewrite rules.

4.2.3 An example for translating the XML update language

The ORDB schema graph derived from mapping the four linked XML documents from section 3.4 is shown in Figure 4.6.

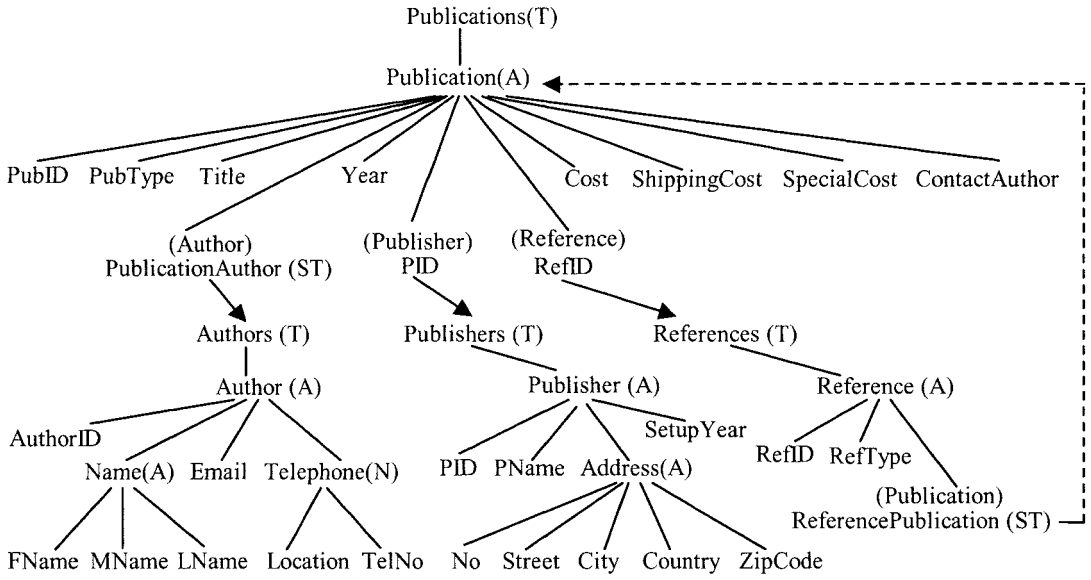

Figure 4.6 ORDB schema graph

From the ORDB schema graph, nodes without a symbol are fields, nodes with (A) are abstract data types, nodes with (N) are nested tables, nodes with (T) are object tables and nodes with (ST) are separate tables keeping the relationship between any two elements. For the table or field derived from the rlink-element, the element name for its source is indicated in the parenthesis. A line with an arrow stands for `rlink` while a dashed-line stands for a recursion.

The command below deletes an rlink-element: *Publication* of the *Reference* (that refers to *PubID* = 'P222') and deletes *Year* of the *Publication* whose *Title* is 'Java'.

```
For $p in doc("Publications.xml")//Publication,
    $r in $p/Reference ~>//Reference
Where $p/Title = "Java"
Delete $p/Year,
Delete $r/Publication
Where $r/rlink(Publication, //Publication[@PubID = "P222"])
```

The steps of translation are as follows:

1. Parse the command and rewrite it as SQL functions. In this step

   Where $r/rlink(Publication, //Publication[@PubID = "P222"]) is rewritten as:

   where ($r/Publication#Publications/Publication, =, 'P222', 2)

   since the predicate of XPath in the rlink-element is based on the PK of the linked element: PubID = "P222".

   The command is rewritten as SQL functions as follows:

```
bindF ('Publications.xml'//Publication, $p, 0)
bindF ($p/Reference ~>//Reference, $r, 0)
where ($p/Title, =, 'Java', 0)
delete($p/Year, 1)
delete($r/Publication, 2)
where($r/Publication#Publications/Publication, = , 'P222', 2)
```

2. Translate the functions into a graph whose paths correspond to paths in the SQL functions. The graph is shown in Figure 4.7(a).
3. Map the graph to the ORDB schema graph and then add key fields (PK and FK) which are used to join tables and finally add the join symbols to indicate which pair of keys is used to join the tables. The result is shown in Figure 4.7(b).

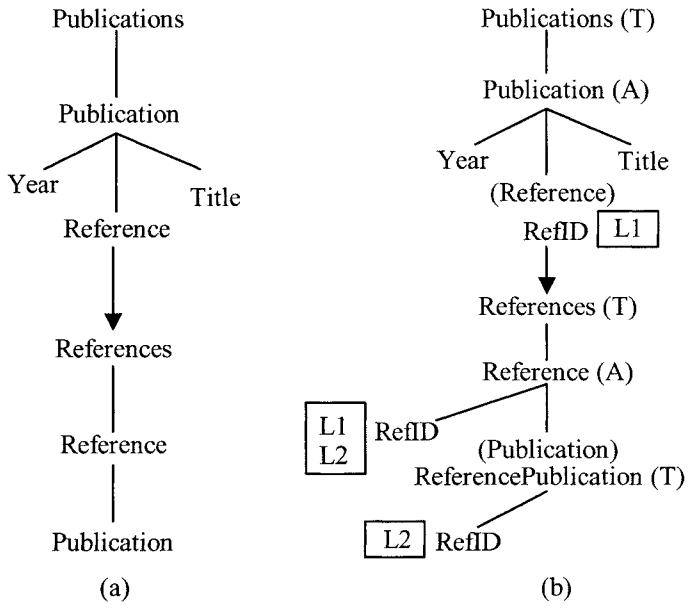

Figure 4.7 (a) Graph whose paths corresponding to paths in SQL functions, (b) Mapping the graph to the ORDB schema graph

4. Map the function to the graph. Since the *delete* function is performed on a node *Year* converted to a field; thus the function is changed to an *update* function. For the function *where*($r/Publication#Publications/Publication, = , "P222", 2), it indicates that the function *where* will be performed on the field *PubID*, the primary

key of *$r/Publication* referring to the linked element *Publications/Publication*; thus the function *where* is pushed down to the *PubID*. The result is shown in Figure 4.8.

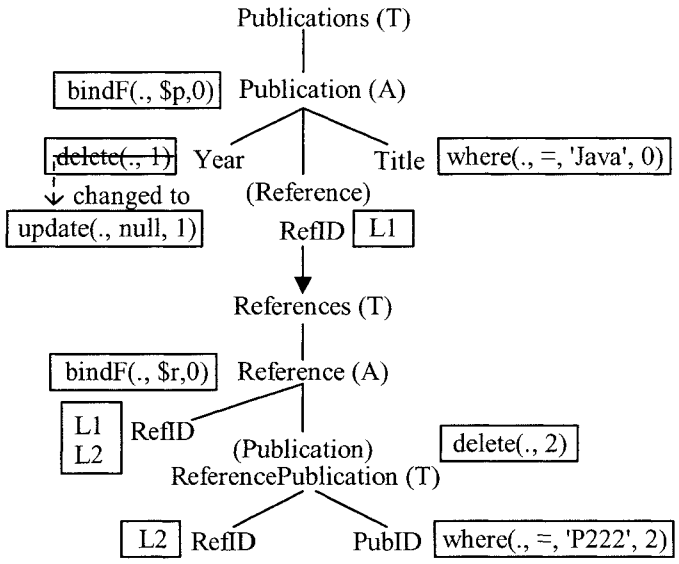

Figure 4.8 Mapping SQL functions to the graph

5. Separate the graph into sub-graphs. Since there are two updated target tables: *Publications* and *ReferencePublication*; thus the graph is separated into two sub-graphs as shown in Figure 4.9.

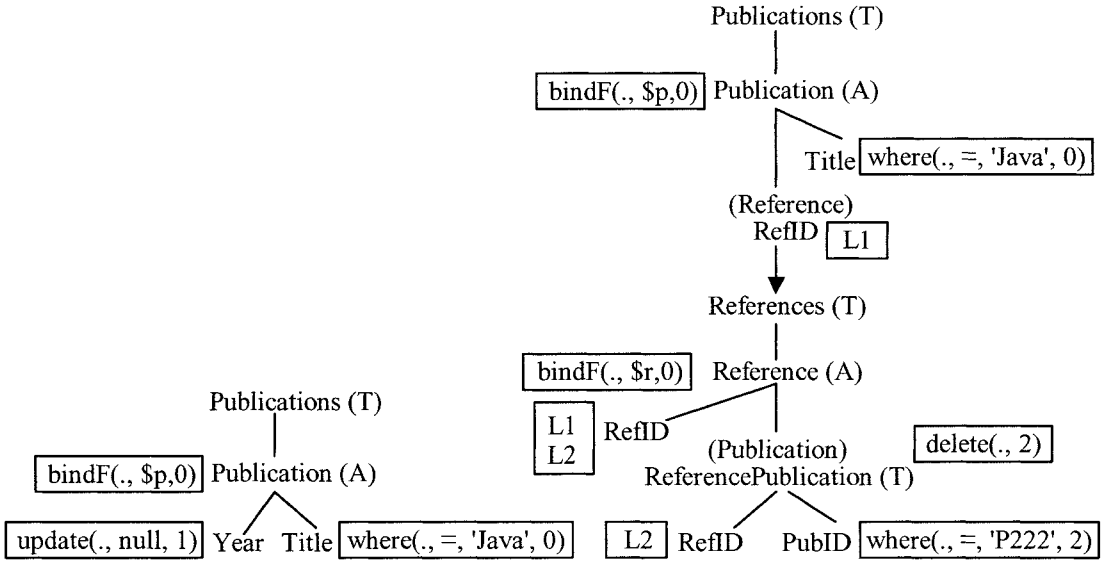

Figure 4.9 Two sub-graphs for two target updated tables

6. Optimise each sub-graph by using optimisation rules. The first sub-graph is involved with only one table; thus optimising is not required. The second sub-graph

is involved with three tables: *Publications*, *References* and *ReferencePublication*. Consider these three contiguous tables. Since there is no *update* functions or conditions on the *References* and the field *RefID* of the *Publications* and the *ReferencePublication* can join directly; then the *References* can be eliminated from the graph. The result is shown in Figure 4.10.

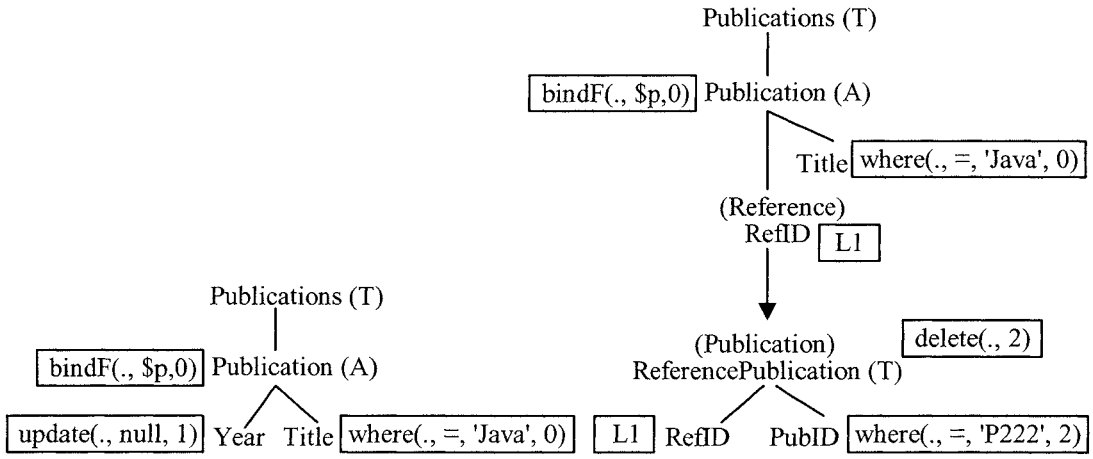

Figure 4.10 Two sub-graphs after optimising

7. Create SQL commands from each sub-graph

   For the first sub-graph, only one table is involved in the update; thus there is no join in the update operation so the SQL command is generated as follows:

```
Update Publications P
Set P.Publication.Year = null
Where P.Publication.Title = 'Java';
```

   For the second sub-graph, a delete join command is generated as follows:

```
Delete ReferencePublication RP
From RP, Publications P
Where RP.PubID = 'P222'
And RP.RefID = P.Publication.RefID
And P.Publication.Title = 'Java';
```

   The delete join command is rewritten as delete with a sub-query as follows:

```
Delete From ReferencePublication RP
Where RP.PubID = 'P222'
And RP.RefID in (select P.Publication.RefID
                 from Publications P
                 where P.Publication.Title = 'Java' ) ;
```

More examples for translating XML update commands into SQL, both in the case of linked XML documents and a single XML document, are shown in Appendix A.

4.3 Translating a recursive function

A recursive function is processed iteratively by passing the value of a variable in each iteration. Moreover an XQuery recursive function can be used along with several features such as the conditional expression and quantifiers whereas a path expression that uses a descendant path (//) to query data whose structure is recursive cannot be used along with these features. Thus unlike path expression queries, the XQuery recursive function cannot be translated into pure SQL commands. Here, we will translate the XQuery recursive function into an SQL procedure, which is a collection of SQL commands equipped with a programming capability. In the SQL standard, it is called a persistent stored module while in Oracle, it is PL/SQL [88]. In this research, PL/SQL is used as an example for translating the recursive function since Oracle is used as the ORDBMS for our prototype.

In the translation, we apply the concept of variables to the concept of tables since only the tables can be directly manipulated by SQL commands.

In this section, firstly, the mechanism for passing a variable's value is described. Secondly, the rewrite rules for translating the recursive function are proposed. Thirdly, steps for translating the recursive function into an SQL procedure are presented and finally an example is given.

4.3.1 The mechanism for passing a variable's value

The mechanism for passing a variable's value is applied to selecting and inserting data from/into tables as follows.

1. The concept of passing a value of a variable to another variable is applied to the concept of selecting data from a table and then inserting this data into another table.
2. The notion of passing a variable's value is that the old value in a variable will be overwritten by the new value passed by another variable. To use tables instead of variables, it means that before inserting data into a table, the old data in the table must be deleted first.

4.3.2 Rewrite rules for translating a recursive function

The clauses of the XML update command in a recursive function will be rewritten as SQL functions or SQL-syntax commands subject to the following rules.

1. In the rewrite rules for FLW(R|I|D) expression, the *Where*, *Replace*, *Insert* and *Delete* clauses are rewritten as SQL functions as mentioned earlier whereas the rule for *For/Let* clause is changed. Instead of binding variables to nodes in XPath, the meaning of an operation in *For/Let* clause is interpreted as the meaning of the operation in SQL because the number for calling the function is dynamic depending on the result derived from the previous loop processing. Thus the number of binding variables cannot be determined in advance. The rule for rewrite the *For/Let* clause is as follows.

   For $var in XPath | Let $var := XPath is rewritten as:

   Insert into $var
   select(XPath, funcNo)

2. The variable which passes a value in the calling function is called 'argument' while the variable which receives the value from the argument is called 'parameter'.

   Thus, in function call,

   Passing $argument to $parameter is rewritten as:

   Insert into $parameter Select * from $argument

3. Replace the variables (parameter and argument) in SQL functions or SQL-syntax commands derived from rules 1 and 2 with their corresponding tables or elements. The variables are categorised into two types: variables which are not a part of XPath and variables which are a part of XPath. Thus the rules for replacing variables are as follows:

   3.1 The variable which is not a part of XPath will be replaced with a table. Here we define that the argument will be superseded by the table *Array* whereas the parameter will be substituted by the table *ProcessingArr*; thus

   ```
   If $var is argument then
           $var    is rewritten as:  table 'Array'
   ElseIf $var is parameter then
           $var    is rewritten as:  table 'ProcessingArr'
   EndIf
   ```

   3.2 The variable which is a part of XPath is the argument/parameter whose value is changed in each loop processing. Since the argument/parameter is converted to

the table *Array/ProcessingArr*, the variable will be converted to its corresponding element whose value is the value kept in the table *Array/ProcessingArr*. In other words, the value of the element converted from the variable in XPath will be derived/selected from the table *Array/ProcessingArr*.

There are two cases for the variable which is a part of XPath: the variable is in the *where* function and the variable is in other SQL functions. In both cases, the variable will be replaced with its corresponding element. In the case of the *where* function, there is no need to select the value from the table *Array/ProcessingArr* since the *where* function has its own condition whereas the value of the variable of other SQL functions is selected from the value kept in the table *Array/ProcessingArr*.

The rules for replacing the variable which is a part of XPath are as follows.

Suppose that E1 is the element corresponding to the variable $var; ergo

where($var/XPath, funcNo) is rewritten as:

where(E1/XPath, funcNo)

Suppose that E1 is the element corresponding to the variable $var and SQLFunct is any SQL function which is not the *where* function; ergo

SQLFunct($var/XPath, funcNo) is rewritten as:

SQLFunct(E1/XPath, funcNo)
where (E1, in, , funcNo)
(select * from Array/ProcessingArr)

4.3.3 Steps for translating a recursive function

The steps for translating a recursive function into an SQL procedure are as follows:

1. Rewrite each clause of the update command until the first calling function in the body of the function is found by using both the rules 1-2 and the rewrite rules in section 4.2.1. The first calling function will not be rewritten.
2. Create a loop structure when the first calling function in the body of the function is found. In the loop, the first calling function and each clause in the body of the function is rewritten until the second calling function is found by using both the rules 1-2 and the rewrite rules in section 4.2.1. The second calling function will not be rewritten.
3. Replace the variables in SQL functions and SQL-syntax commands derived from steps 1-2 by using the rule 3.
4. Follow the concept of passing a variable's value; therefore before inserting data into tables, the old value in such tables must be deleted first. Thus a clause for deletion of old data will be added prior to each clause for insertion of data.
5. Translate SQL functions into SQL commands by using the graph mapping technique and the optimisation rules as described in section 4.2.1.

4.3.4 An example for translating a recursive function

This example translates a recursive function into the SQL procedure. PL/SQL is used as an example of the SQL procedure for translating the recursive function. The recursive function below updates the year of publications which are both direct and indirect references of the publication whose title is 'XQuery'. The function below follows the syntax of XQuery.

```
1  define function allRef($pub as element()*)
2  { For $rp in
        $pub/Reference~>//Reference/Publication~>//Publication
3   Replace $rp/Year with <Year>2004</Year>
4   allRef($rp)
5  }
6  For $p in doc("Publication.xml")/Publications/Publication
7  Where $p/Title = "XQuery"
8  allRef($p)
```

The process starts from the clause in line 6. The steps for translation are as follows.

1. Rewrite each clause until the first calling function in the body of function is found. The clauses and the result of rewriting, by using the rewrite rules 1-2 and the rewrite rules in section 4.2.1, are shown in Figure 4.11.

Clauses outside loop	Result of rewriting
6 For $p in doc("Publication.xml")/Publications/Publication 7 Where $p/Title = "XQuery"	Insert into $p select (/Publications/Publication,0) where ($p/Title, = ,'XQuery', 0) ;
8 allRef($p) 1 define function allRef($pub as element()*)	Insert into $pub Select * from $p;
2 For $rp in $pub/Reference~> //Reference/Publication~>//Publication	Insert into $rp select($pub/Reference~>//Reference/ Publication~>//Publication, 1);
3 Replace $rp/Year with <Year>2004</Year>	update($rp/Year, '2004', 2);

Figure 4.11 Result of rewriting clauses which are outside loop

2. Create a loop structure when the first calling function in the body of the function is found. Rewrite each clause until the second calling function is found. The clauses in the loop and the result of rewriting, by using the rewrite rules 1-2 and the rewrite rules in section 4.2.1, are shown in Figure 4.12.

Clauses inside loop	Result of rewriting
4 allRef($rp) 1 define function allRef($pub as element()*)	Insert into $pub Select * from $rp;
2 For $rp in $pub/Reference~> //Reference/Publication~>//Publication	Insert into $rp select($pub/Reference~> //Reference/Publication~>//Publication,3);
3 Replace $rp/Year with <Year>2004</Year>	update($rp/Year, '2004', 4);

Figure 4.12 Result of rewriting clauses which are inside loop

3. Replace the variables by using the rule 3. The results of replacing variables in clauses which are outside and inside the loop are shown in Figure 4.13 and 4.14 respectively.

Clauses outside loop	Result of replacing variables
Insert into $p select (/Publications/Publication,0) where ($p/Title, = ,'XQuery', 0);	Insert into Array select (/Publications/Publication,0) where (/Publications/Publication/Title,=,'XQuery,0) ;
Insert into $pub Select * from $p;	Insert into ProcessingArr Select * from Array;
Insert into $rp select($pub/Reference~>//Reference/ Publication~>//Publication, 1)	Insert into Array select (/Publications/Publication/Reference~> //Reference/Publication~>//Publication, 1) where (/Publications/Publication, in, , 1) (select * from ProcessingArr) ;
update($rp/Year, '2004', 2);	update(/Publications/Publication/Year,'2004', 2) where(/Publications/Publication, in, , 2) (Select * from Array);

Figure 4.13 Result of replacing variables in clauses which are outside loop

Clauses inside loop	Result of replacing variables
Insert into $pub Select * from $rp;	Insert into ProcessingArr Select * from Array;
Insert into $rp select($pub/Reference~>//Reference/ Publication~>//Publication, 3)	Insert into Array select (/Publications/Publication/Reference~> //Reference/Publication~>//Publication, 3) where (/Publications/Publication, in, , 3) (select * from ProcessingArr) ;
update($rp/Year, '2004', 4);	update(/Publications/Publication/Year,'2004',4) where(/Publications/Publication, in, , 4) (Select * from Array);

Figure 4.14 Result of replacing variables in clauses which are inside loop

4. Follow the concept of passing a variable's value. Thus a clause for deletion of old data is added prior to each clause for insertion of data. The result is shown in Figure 4.15

```
Begin
     Delete from Array;
     Insert into Array
     select(/Publications/Publication,0)
     where(/Publications/Publication/Title, = ,'XQuery', 0) ;

     Delete from ProcessingArr;
     Insert into ProcessingArr
     Select * from Array;

     Delete from Array;
     Insert into Array
     select(/Publications/Publication/Reference~>
             //Reference/Publication~>//Publication, 1)
     where(/Publications/Publication, in, , 1)
     (select * from ProcessingArr) ;

     update(/Publications/Publication/Year, '2004', 2)
     where(/Publications/Publication, in, ,2)
     (select * from Array);

   Loop
      If SQL%RowCount > 0 then
            Delete from ProcessingArr;
            Insert into ProcessingArr
            Select * from Array;

            Delete from Array;
            Insert into Array
            select(/Publications/Publication/Reference~>
                 //Reference/Publication~>//Publication, 3)
            where(/Publications/Publication, in, , 3)
            (select * from ProcessingArr) ;

            update(/Publications/Publication/Year, '2004', 4)
            where(/Publications/Publication, in, ,4)
            (select * from Array);
      Else
         Exit;
      End If;
    EndLoop;
End;
```

Note: Begin..End; block is added to make PL/SQL command completed.

Figure 4.15 Result of adding a delete clause before each insertion of data into tables

5. Translate each group of SQL functions independently into SQL commands as follows:
   5.1 Each graph for each group of SQL functions is created. The paths of each graph are created according to paths in SQL functions.
   5.2 The graphs are mapped to a database schema graph. Then join keys and join symbols are added to the graphs. Then the functions are mapped to the graphs.
   5.3 The *select* and *where* functions are performed on the *Publication* which is converted to an abstract data type without a sibling; thus these functions are pushed down to the proper primary key as shown in Figures 4.16 (a), 4.16 (b) and 4.16 (c) which are graphs of SQL functions in groups 0, 2 and 1 respectively.
   5.4 In group 1, the *select* function performed on the *Publication* which is the element referenced from its descendant (recursive path) will be pushed down to the field of the descendant linking to the *Publication*. The result is shown in Figure 4.16 (c).
   5.5 The graph in Figure 4.16 (c) is optimised according to the optimisation rule 3. Three contiguous tables: *Publications*, *References* and *ReferencePublication* are involved in the graph. Consider these three tables. Since there are no *update* functions or conditions on the *References* and the field *RefID* of the *Publications* and the *ReferencePublication* can join directly; then *References* can be eliminated from the graph. The result is shown in Figure 4.16 (d).
   5.6 SQL commands are generated from the graphs. The SQL functions in groups 3 and 4 are translated in the same way as the SQL functions in groups 1 and 2 respectively; thus the translation for the functions in groups 3 and 4 is not shown here. The SQL commands derived from the graphs are shown in Figure 4.17.
   5.7 The PL/SQL procedure after replacing SQL functions with SQL commands generated from the graphs is shown in Figure 4.18.

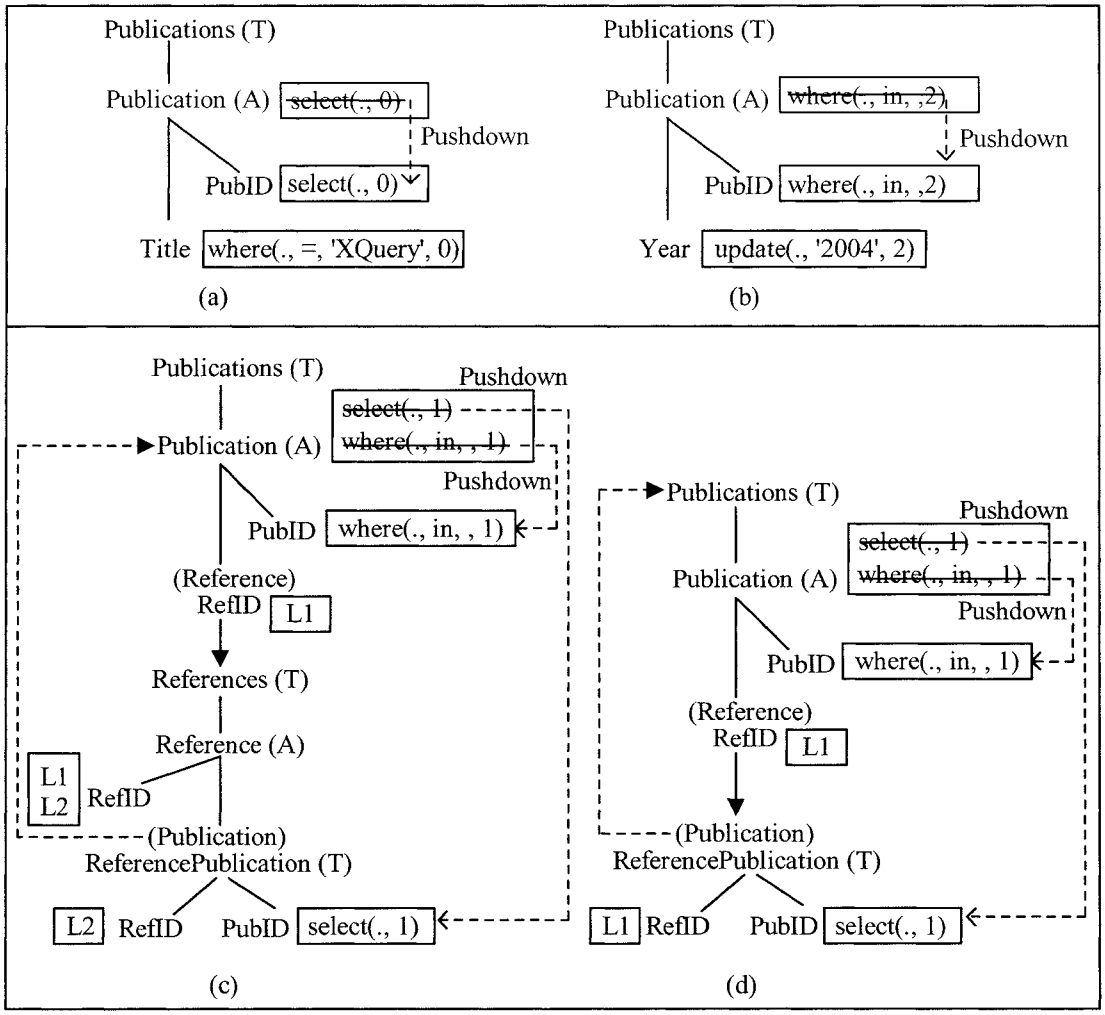

Figure 4.16 Graphs after pushing (a) *select* in group 0, (b) *where* in group 2, (c) *select* and *where* in group 1, (d) graph of SQL functions in group 1 after optimisation

SQLfunctions	SQL commands
select (/Publications/Publication,0) where (/Publications/Publication/Title,=,'XQuery', 0) ;	select P.Publication.PubID from Publications P where P.Publication.Title = 'XQuery' ;
select (/Publications/Publication/Reference~> //Reference/Publication~>//Publication, 1) where (/Publications/Publication, in, , 1)	select RP.PubID from Publications P, ReferencePublication RP where RP.RefID = P.Pulbication.RefID and P.Publication.PubID in
update(Publications/Publication/Year, '2004', 2) where (Publication, in, ,2)	update Publications P set P.Publication.Year = '2004' where P.Publication.PubID in
select (/Publications/Publication/Reference~> //Reference/Publication~>//Publication, 3) where (/Publications/Publication, in, , 3)	select RP.PubID from Publications P, ReferencePublication RP where RP.RefID = P.Pulbication.RefID and P.Publication.PubID in
update(Publications/Publication/Year, '2004', 4) where (Publication, in, ,4)	update Publications P set P.Publication.Year = '2004' where P.Publication.PubID in

Figure 4.17 SQL commands generated from the graphs

```
Begin
   Delete from Array;
   Insert into Array
   select P.Publication.PubID
   from Publications P
   where P.Publication.Title = 'XQuery' ;

   Delete from ProcessingArr;
   Insert into ProcessingArr
   Select * from Array;

   Delete from Array;
   Insert into Array
   select RP.PubID
   from Publications P, ReferencePublication RP
   where RP.RefID = P.Pulbication.RefID
   and P.Publication.PubID in  (select * from PricessingArr);

   update Publications P
   set P.Publication.Year  = '2004'
   where P.Publication.PubID in  (select * from Array);

   Loop
      If SQL%RowCount > 0 then
         Delete from ProcessingArr;
         Insert into ProcessingArr
         Select * from Array;

         Delete from Array;
         Insert into Array
         select RP.PubID
         from Publications P, ReferencePublication RP
         where RP.RefID = P.Pulbication.RefID
         and P.Publication.PubID in (select * from ProcessingArr);

         update Publications P
         set P.Publication.Year  = '2004'
         where P.Publication.PubID in  (select * from Array);
      Else
         Exit;
      End If;
      EndLoop;
End;
```

Figure 4.18 PL/SQL procedure after replacing SQL functions

An example of translating a recursive function into PL/SQL in the case of a single XML document is shown in Appendix A.

4.4 Summary

This chapter started by designing the syntax and semantics of the XML update language, an extension to XQuery. It then concentrated on translating the XML update language into SQL to update XML data stored in an ORDB. The six important features inherited from XQuery are translated into SQL: FLW(R|I|D) expression, conditional expression, quantifier, aggregate functions, (non-recursive) user-defined function and recursive function. Four main techniques: update/delete join commands, rewrite rules, graph mapping and optimisation rules, are utilised for translating the XML update language into SQL. The recursive function is translated into an SQL procedure: a series of SQL commands equipped with a programming capability. Here, PL/SQL is used as an example of the SQL procedure in translating a recursive function to illustrate our technique. In translating the recursive function, we apply the concept of variables to the concept of tables since only the tables can be directly manipulated by SQL commands. Then we use the graph mapping technique and the optimisation rules to generate SQL commands from SQL functions.

In Chapter 5, the next chapter, the change in an ORDB executed by SQL, which has been translated from the XML update language, will be propagated to the original XML documents. Logical statements for updating XML documents will also be presented.

Chapter 5: Reflecting the changes to XML

Chapter 4 proposed an approach for updating XML data stored in an ORDB by translating the XML update language into SQL, which was then executed on the ORDB. In this chapter, a reflection of the changes from the ORDB to XML documents will be presented. In addition, logical statements for updating XML documents by our approach described in Chapters 3, 4 and this chapter will be expressed to show that the values of data in ORDB are equivalent to the values of data in XML documents after updating.

Reflecting the changes from a database to XML documents can be performed by the naïve method: reconstructing the whole XML document. This method yields very low performance since the document is created from scratch [3] and it will be worse if several XML documents are updated since several documents must be reconstructed while usually updating will simply affect only some small parts of the documents. Thus in order to improve performance, reflecting the changes will be performed on only the affected parts.

The remainder of this chapter is organised as follows. Section 5.1 presents how the changes in an ORDB are propagated to XML documents. Section 5.2 describes locating the target or reference position in XML documents for updating. Section 5.3 presents the steps for creating the XPath used to locate the target or reference position for updating and section 5.4 provides an example for creating the XPath. Section 5.5 presents logical statements expressing the approach of updating XML documents. Section 5.6 provides a summary of the chapter.

5.1 Propagating the changes from an ORDB to XML documents

Propagation of the changes from an ORDB to XML documents is performed on only the parts affected by updating. The parts or elements which should be changed in XML documents can be derived from the original conditions in an XML update command by

using the XPath expression to indicate the positions in XML document for updating. However, the original conditions may be complex; thus searching these parts or elements may consume unnecessary time whereas the IDs of elements which should be updated can actually be derived from the step of updating data in the ORDB. Hence, rather than using the original conditions of the XML update command, we use values of a primary key or a foreign key of updated data in the database to indicate the elements in XML documents which should be updated.

The primary keys originate from attributes whose type is ID in each element of XML documents. In the case of some elements converted to tables but when the elements do not provide ID attributes, the value of RowIDs automatically generated by the ORDB system will be recorded for such elements as ID attributes. Hence the values of ID attributes of elements can be indicated by the values of RowIDs kept in the ORDB or by the values of a primary key or a foreign key stored in the ORDB.

When data in the ORDB is updated, the values of a primary key or a foreign key of the updated data will be returned. Subsequently paths in the XML update command are used to create XPath expressions whose conditions are based on these returned values to indicate the target or reference positions in XML documents for updating.

XPath is used to find nodes which are the target or reference positions for updating; however, the XPath has no capability for updating. Hence we propose five propagate functions which serve as operators for updating XML documents. Figure 5.1 presents the list of the propagate functions.

Propagate Functions	**Description**
Insert (nodeLst, targetLst)	Inserting nodeLst into nodes in targetLst
InsertBefore(nodeLst, targetLst)	Inserting nodeLst before nodes in targetLst
InsertAfter(nodeLst, targetLst)	Inserting nodeLst after nodes in targetLst
Delete (targetLst)	Deleting nodes in targetLst
Update (nodeLst, targetLst)	Replacing values of nodes in targetLst with values of nodes in nodeLst

Figure 5.1 Propagate functions

From the functions in Figure 5.1, the parameter 'nodeLst' can be derived directly from the XML update command. The parameter 'targetLst' is an XPath expression whose conditions are based on values for keys returned from the ORDB and sometimes some

predicates in the XML update command. The path in the XPath expression is derived from the path in the XML update command. The type of node in the 'nodeLst' and the 'targetLst' depends on the type of propagate functions as described as follows:

1. Insert (nodeLst, targetLst)

   The parameter 'nodeLst' contains a single pair, attribute-value, whose type is not ID or contains a list of node-value pairs whose nodes can be simple elements or attributes. In the 'nodeLst', a complex element is broken down to a list of simple elements. The value of the parameter 'targetLst' is an XPath expression, retrieving elements which are target containers of the 'nodeLst'. This function performs the insertion of nodes in the 'nodeLst' into elements retrieved by the 'targetLst'.

2. InsertBefore(nodeLst, targetLst)

   The parameter 'nodeLst' contains a single pair, attribute-value, whose type is IDREFs or contains a list of node-value pairs whose nodes can be simple elements or attributes. The value of the parameter 'targetLst' is an XPath expression, retrieving elements or attributes whose type is IDREFs which are reference positions for inserting the 'nodeLst'. This function performs the insertion of nodes in the 'nodeLst' before elements/attributes retrieved by the 'targetLst'.

3. InsertAfter(nodeLst, targetLst)

   The parameters are the same as the parameters of the InsertBefore function but this function performs the insertion of nodes in the 'nodeLst' after elements/attributes retrieved by the 'targetLst'.

4. Delete (targetLst)

   The value of the parameter 'targetLst' is an XPath expression, retrieving elements or attributes whose type is not ID which are targets of the deletion. In the case of deleting nodes which are complex elements, all children of these elements will be automatically cascade deleted. This function performs the deletion of elements/attributes retrieved by the 'targetLst'.

5. Update (nodeLst, targetLst)

   The parameter 'nodeLst' contains a single pair, attribute-value, whose type is not ID or contains a list of node-value pairs whose nodes can be simple elements or attributes whose type is not ID. The value of the parameter 'targetLst' is an XPath expression, retrieving elements or attributes whose type is not ID as the targets of the updating. This function performs the replacement of values of

elements/attributes retrieved by the 'targetLst' with values of nodes in the 'nodeLst'.

5.2 Locating target/reference position for updating

To propagate the changes to XML documents by using the propagate functions, the target nodes or reference nodes in XML documents for updating must be identified. These nodes are identified by an XPath expression whose condition is based on the values for keys returned from the ORDB and some predicates derived directly from the XML update command. Such predicates are the predicate of *before* and *after* clauses and the predicates on IDREFs, rlink-elements and elements/attributes converted to fields of nested tables. To determine which values for a key should be returned, the types of ORDB structure and types of update operations performed on such an ORDB structure are examined. A summary of operations performed on the ORDB and values for the key returned from the ORDB, when each type of XML update command is performed on each type of XML structure, is shown in Table 5.1.

The details of the eight columns in Table 5.1 are as follows.

- Column 1 shows the type of an update operation performed on XML documents.
- Column 2 shows the type of an XML structure on which the update operation in column 1 is performed.
- Column 3 shows the type of an ORDB structure converted from the XML structure in column 2.
- Column 4 shows the type of an update operation performed on an ORDB. The type of an update operation on the ORDB may be different from the type of an update operation on XML documents. This depends on the type of the ORDB structure as defined by the update-rules in section 4.2.
- Column 5 shows the keys for the values which are returned from an ORDB.
- Column 6 shows which ID in XML documents is the receiver for the returned values for the keys in column 5. The receiver and the returned values are used to compose a key-condition for an XPath expression.
- Column 7 shows the other conditions for an XPath expression. These conditions are derived directly from an XML update command. Such conditions are the

predicate of *before* and *after* clauses and the predicates on IDREFs, rlink-elements and elements/attributes converted to fields of nested tables.

- Column 8 shows the element or attribute which is the target or reference position for updating. This position is retrieved by an XPath expression.

Rules for returning values for the keys can be summarised from Table 5.1 as follows:

Return-rule 1: If fields or (fields of) ADTs are updated, the values for the primary key of updated rows are returned.

Return-rule 2: If fields of nested tables are updated or nested tables are updated/deleted/inserted, the values for the primary key of rows containing the nested tables are returned.

Return-rule 3: If ADTs without a sibling or tables are updated/deleted, the values for the primary key of updated/deleted rows are returned.

Return-rule 4: If new rows are inserted into an ADT without a sibling or a table when the table of the ADT or the table itself is not the root element, then the values for the foreign key of the inserted rows are returned.

Return-rule 5: If new rows are inserted into an ADT without a sibling when the table of the ADT is the root element, then nothing is returned.

Return-rule 6: If a separate table that keeps the relationship between two elements is updated/deleted/inserted, the values of the first primary key (primary key derived from the referencing element) of updated/deleted/inserted rows are returned.

The rest of this section will explain the details of the replace operation on the XML side summarised in Table 5.1 and show how to define the parameters of the propagate functions for each case of the replace operation. The details for insert and delete operations on the XML side will be omitted since their explanations are similar to that of the replace operation.

The related notations for defining the parameters are as follows:

Define: @PK is ID of E_i

PK is the key-field of updated table converted from @PK

$V_{pk_1} \ldots V_{pk_n}$ are values of PK indicating which rows are updated

Table 5.1 Summary of update operations on XML and ORDB sides, keys for returned values, receivers, conditions and target of updating

On XML side		On ORDB side		Key-condition to locate target/reference position		Other conditions used along with key-condition	Target or reference position
Operation	XML structure	ORDB structure	Operation	Keys for values returned from ORDB	receiver for returned value		
replace	E or A	field, ADTF, ADT with sibling (E)*	update	PK of updated row	ID of nearest ancestor		E or A
	E or A	NT (E)* or NTF	update	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT	
	E	ADT no sibling, table	update	PK of updated row	ID of E or nearest desc.		
	IDREFs	Separate table	update	1st PK of updated row	ID of nearest ancestor	condition on IDREFs	
	rlinkE	Separate table	del, ins	1st PK of deleted row	ID of nearest ancestor	condition on E@rlink-href	E@rlink-href
	rlinkE	FK	update	PK of updated row	ID of nearest ancestor		
delete	E or A	field, ADTF, ADT with sibling (E)*	update	PK of updated row	ID of nearest ancestor		E or A
	E	NT	delete	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT	
	E or A	NTF	update	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT	
	E	ADT no sibling, table	delete	PK of deleted row	ID of E or nearest desc.		
	IDREFs	Separate table	delete	1st PK of deleted row	ID of nearest ancestor	condition on IDREFs	
	rlinkE	Separate table	delete	1st PK of deleted row	ID of nearest ancestor	condition on E@rlink-href	
	rlinkE	FK	update	PK of updated row	ID of nearest ancestor		
Insert...into	E or A	field, ADTF, ADT with sibling (E)*	update	PK of updated row	ID of nearest ancestor		Parent of E/A
	E	NT	insert	PK of row containing NT	ID of nearest ancestor		
	E or A	NTF	update	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT	
	E	ADT no sibling, table (not root)	insert	FK of inserted row	ID of nearest ancestor		
	E	ADT no sibling (root)	insert	none	none		
	IDREFs, rlinkE	Separate table	insert	1st PK of inserted row	ID of nearest ancestor		
	rlinkE	FK	update	PK of updated row	ID of nearest ancestor		
insert...into before/after	E	field, ADTF, ADT with sibling (E)*	update	PK of updated row	ID of nearest ancestor	condition of before/after	Sibling of E/A specified in before/after condition
	E	NT	insert	PK of row containing NT	ID of nearest ancestor	condition of before/after	
	E	NTF	update	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT condition of before/after	
	E	ADT no sibling, table (not root)	insert	FK of inserted row	ID of nearest ancestor	condition of before/after	
	E	ADT no sibling (root)	insert	none	none	condition of before/after	
	IDREFs, rlinkE	Separate table	insert	1st PK of inserted row	ID of nearest ancestor	condition of before/after	
	rlinkE	FK	update	PK of updated row	ID of nearest ancestor	condition of before/after	

E: element, A: attribute, NT: nested table, NTF: nested table field, ADT: Abstract data type, ADTF: field of ADT, rlinkE: element containing rlink mechanism, PK: primary key, FK: foreign key

(E)* : The ORDB structure can be converted from E only (not from A), (root): table of ADT is root element, (not root): table (of ADT) is not root element

5.2.1 An element/attribute converted to a field (of ADT) or an ADT having a sibling

If an XML replace command is performed on the element/attribute converted to a field (of ADT) or an ADT having a sibling, an update will be performed on the field (of ADT) or the ADT having a sibling in the ORDB and the values for the primary key of the updated row containing the field/ADT are returned. On the XML side, the ID of the nearest ancestor of the element/attribute is the receiver for the returned values used to compose a key-condition. The key-condition is used to indicate the element/attribute in XML documents that will be updated. For each case, the parameters are as follows:

Case 1: @A is converted to a field or a field of an ADT

Replace $/E_1/E_2/\ldots/E_i/\ldots/@A$ with @A = 'newValue'

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK = V_{pk_1}$ or ... or $@PK = V_{pk_n}]/\ldots/@A$

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/@A$, newValue) }

Case 2: E_n is converted to a field or a field of an ADT

Replace $/E_1/E_2/\ldots/E_i/\ldots/E_n$ with $<E_n>$newVal$</E_n>$

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK = V_{pk_1}$ or ...or $@PK = V_{pk_n}]/\ldots/E_n$

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/E_n$, newVal) }

Case 3: E_{n-1} is converted to an ADT having a sibling and E_{n_1},..., E_{n_m} are converted to fields of the ADT

Replace $/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}$ with $<E_{n-1}>$
$<E_{n_1}>val_1</E_{n_1}>$
...
$<E_{n_m}>val_m</E_{n_m}>$
$</E_{n-1}>$

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK = V_{pk_1}$ or ...or $@PK = V_{pk_n}]/\ldots/E_{n-1}$

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/E_{n-1},)$, $(/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/E_{n_1}, val_1)$, ...,
$(/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/E_{n_m}, val_m)$ }

5.2.2 An element/attribute converted to a field of NT or an NT

If an XML replace command is performed on the element/attribute converted to a nested table or a field of a nested table, an update will be performed on the field or the nested table in the ORDB and the values for the primary key of the row containing the nested table are returned. On the XML side, two types of conditions are used to indicate the element/attribute in XML documents that will be updated: a key-condition and conditions on fields of a nested table. The ID of the nearest ancestor of the element/attribute is the receiver for the returned values used to compose the key-condition while the conditions on fields of the nested table are derived directly from the XML replace command if they are provided. For each case, the parameters are as follows:

Case 1: E_{n-1} is converted to a nested table and E_{n_1} and E_{n_2} are converted to fields of the nested table

Replace $/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/E_{n_1}$ with $<E_{n_1}>$newVal$</E_{n_1}>$
Where $/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/E_{n_2}$ = 'val'

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK = V_{pk_1}$ or ...or $@PK = V_{pk_n}]/\ldots/E_{n-1}[E_{n_2} = val]/E_{n_1}$

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/E_{n_1}$, newVal) }

Case 2: E_{n-1} is converted to a nested table and @A and E_n are converted to fields of the nested table

Replace $/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/@A$ with @A = 'newValue'
Where $/E_1/E_2/\ldots/E_i/\ldots/ E_{n-1}/E_n$ = 'val'

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK = V_{pk_1}$ or ... or $@PK = V_{pk_n}]/\ldots/ E_{n-1}[E_n = val]/@A$

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/@A$, newValue) }

Case 3: E_{n-1} is converted to a nested table and E_{n_1} ... E_{n_m} are converted to fields of the nested table

Replace $/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}$ with $<E_{n-1}>$
$<E_{n_1}>val_1</E_{n_1}>$
...
$<E_{n_m}>val_m</E_{n_m}>$
$</E_{n-1}>$

Where $/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/E_{n_1}$ = 'val'

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK = V_{pk_1}$ or ...or $@PK = V_{pk_n}]/\ldots/E_{n-1}[E_{n_1} = val]$

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/E_{n-1},)$, $(/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/E_{n_1}, val_1)$, ...,
$(/E_1/E_2/\ldots/E_i/\ldots/E_{n-1}/E_{n_m}, val_m)$ }

5.2.3 An element converted to an ADT without a sibling or a table

If an XML replace command is performed on the element converted to an ADT without a sibling or a table, an update will be performed on the ADT or the table of the ORDB and the values for the primary key of the updated row are returned. On the XML side, the ID of the element or ID of the nearest descendent is the receiver for the returned values to compose a key-condition. The key-condition is used to indicate the element in XML documents that will be updated. For each case, the parameters are as follows:

Case 1: E_i is converted to an ADT without a sibling or a table and E_{i+1_1} ,..., E_{i+1_m} are converted to fields of the ADT or fields of the table

Replace $/E_1/E_2/\ldots/E_i$ with $<E_i>$
$<E_{i+1_1}>val_1</E_{i+1_1}>$
.....
$<E_{i+1_m}>val_m</E_{i+1_m}>$
$</E_i>$

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK = V_{pk_1}$ or ...or $@PK = V_{pk_n}]$

nodeLst = { $(/E_1/E_2/\ldots/E_i,)$, $(/E_1/E_2/\ldots/E_i/E_{i+1_1}, val_1)$,..., $(/E_1/E_2/\ldots/E_i/E_{i+1_m}, val_m)$ }

5.2.4 An IDREFs converted to a table

An IDREFs of XML is converted to a separate table to keep the relationship between two elements: the ID of the referencing element (first primary key) and the ID of the referenced element (second primary key). Thus the XML replace command performed on the IDREFs is updating on the second primary key and the values for the first primary key of the updated rows will be returned. On the XML side, two conditions are used to indicate the IDREFs' value that will be updated: a key-condition and a condition on the IDREFs. The ID of the nearest ancestor (ID of referencing element) is the receiver for the returned values used to compose the key-condition while the condition on the IDREFs is derived directly from the XML replace command if it is provided. For this case, the parameters are as follows:

Case1: @A is an IDREFs converted to a table

Replace $/E_1/E_2/\ldots/E_i/\ldots/@A$ with @A = 'newValue'
Where $/E_1/E_2/\ldots/E_i/\ldots/@A$ = 'oldVal'

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK = V_{pk_1}$ or ... or $@PK = V_{pk_n}]/\ldots/@A$ [.=oldVal]

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/@A$, newValue) }

5.2.5 An rlink-element converted to a table

In an XML replace command, an rlink-element is converted to a separate table to keep the relationship between two elements: the ID of referencing element (first primary key) and the ID of referenced element (second primary key). Then a sequence of delete and insert operations is performed on this table to delete the old relationship and insert a new relationship. The values for the first primary key of the deleted rows are returned. On the XML side, two conditions are used to indicate the element that will be updated: a key-condition and a condition on the @rlink-href of the rlink-element. The ID of the nearest ancestor (ID of the referencing element) is the receiver for the returned values used to compose the key-condition while the condition on the @rlink-href is derived directly from the XML replace command if it is provided. For this case, the parameters are as follows:

Case 1: E_n is a rlink-element converted a table

Replace $/E_1/E_2/\ldots/E_i/\ldots/E_n$ with rlink(E_n, XPathExp1)
Where $/E_1/E_2/\ldots/E_i/\ldots$/rlink($E_n$, XPathExp2)

Parameters are:

targetLst =
$/E_1/E_2/\ldots/E_i[@PK=V_{pk_1}$ or...or $@PK=V_{pk_n}]/\ldots/E_n$/@rlink-href[.=XPathExp2]

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/E_n$@rlink-href, XPathExp1) }

5.2.6 An rlink-element converted to an FK

If an XML replace command is performed on an rlink-element which is converted to a foreign key, an update will be performed on the foreign key and the value for the primary key of the updated row will be returned. On the XML side, the ID of the nearest ancestor (ID of the referencing element) is the receiver for the returned values used to compose a key-condition. The key-condition is used to indicate the element in the XML documents that will be updated. For this case, the parameters are as follows:

Case 1: E_n is an rlink-element converted to a foreign key

Replace $/E_1/E_2/\ldots/E_i/\ldots/E_n$ with rlink(E_n, XPathExp)

Parameters are:

targetLst = $/E_1/E_2/\ldots/E_i[@PK=V_{pk_1}$ or...or $@PK=V_{pk_n}]/\ldots/E_n$/@rlink-href

nodeLst = { $(/E_1/E_2/\ldots/E_i/\ldots/E_n$@rlink-href, XPathExp) }

5.3 Steps for creating the XPath

As was mentioned earlier in section 5.1, the propagate functions have two parameters: 'nodeLst' and 'targetLst'. The value of the parameter 'nodeLst' can be derived directly from the XML update command. The value of parameter 'targetLst' is an XPath expression whose condition is based on the values for keys returned from the ORDB and some predicates in an XML update command. The steps for creating the XPath expression as the value of the 'targetLst' are as follows:

1. Create each tree whose paths correspond to paths in each delete/replace/insert clause of the XML update command

2. Add the *delete*, *replace*, *insert...into...*, or *insert...before/after...* operation to nodes of the corresponding trees. These nodes will be the returned nodes from XPath expressions and will be the target or reference positions for updating.
3. Add the condition of the *before/after* clause and conditions on the IDREFs, rlink-element and elements/attributes converted to fields of a nested table to the corresponding trees, if these conditions are provided.
4. Map the trees to the ORDB schema graph.
5. By using the summary in Table 5.1 to indicate the receiver of the values returned from the ORDB, add nodes which are receivers of the values returned from the ORDB to the trees.
6. Add conditions based on the values returned from the ORDB to the trees.
7. Generate XPath expressions by starting from the root elements of the documents.

5.4 An Example for creating the XPath

The command below deletes the *Year* of the *Publication* whose *Title* is 'Java' and deletes the *Publication* which is the rlink-element of the *Reference* (of the *Publication* whose *Title* is 'Java') where this rlink-element links back to the *Publication* whose *PubID* is 'P222'.

```
For $p in doc("Publications.xml")//Publication,
    $r in $p/Reference ~>//Reference
Where $p/Title = "Java"
Delete $p/Year,
Delete $r/Publication
Where $r/rlink(Publication, //Publication[@PubID = "P222"])
```

1. There are two delete clauses in the XML update command; thus two trees are created for each delete clause.
2. The delete clauses are added to the corresponding trees.
3. From the second delete clause, there is a condition on the rlink-element; hence this condition is added to the tree of the second delete clause.
4. Both trees are mapped to the ORDB schema graph to indicate the database structure on the trees. The results are shown in Figures 5.2(a) and 5.2(b).

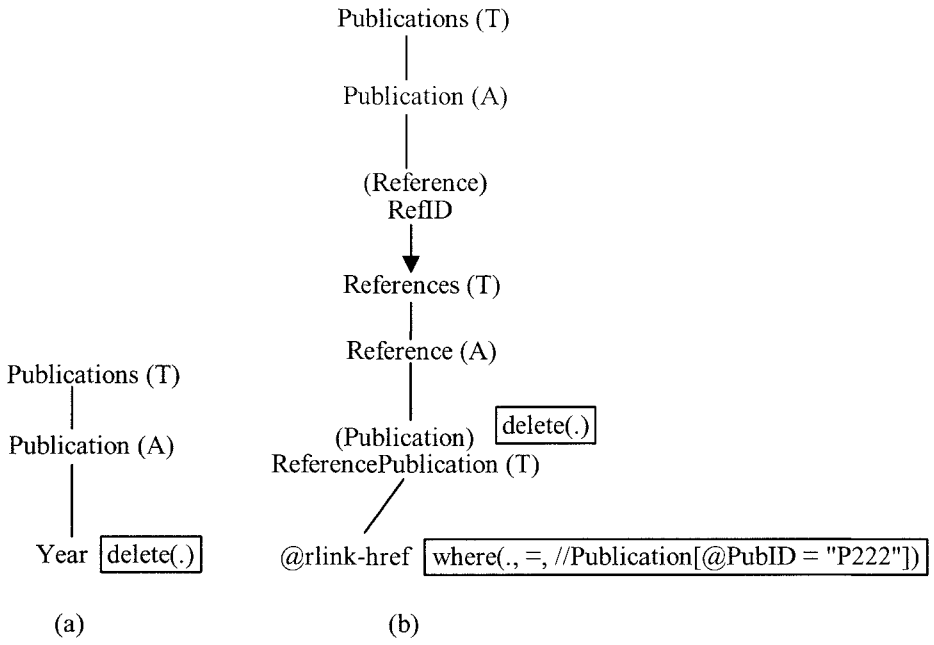

Figure 5.2 Trees of the (a) first and (b) second delete clauses

5. From Figure 5.2(a), deletion is performed on *Year*, a simple field; thus ID of its nearest ancestor is a receiver for the values of the primary key of updated rows returned from the ORDB. The nearest ancestor having an ID is *Publication*; thus its ID (*PubID*) is added to the tree. From Figure 5.2(b), deletion is performed on the rlink-element (*Publication*), a table keeping the ID of the referencing element (1st PK) and the ID of the referenced element (2nd PK); thus the ID of its nearest ancestor is a receiver for the values of the 1st PK of deleted rows returned from the ORDB. The nearest ancestor having an ID is *Reference*; thus its ID (*RefID*) is added to the tree.
6. The conditions based on the returned values are added to the trees as shown in Figure 5.3(a) and 5.3(b). Suppose that the *Publication* whose *Title* = 'Java' has *PubID* = 'P111'; then for the first tree, the returned value is 'P111'. For the second tree, suppose that *RefID* of the *Reference* referring to *PubID* = 'P222' is 'R111'; then the returned value is 'R111'.
7. XPath expressions for indicating target nodes which will be deleted are generated from the trees in Figures 5.3(a) and 5.3(b) and sent to the parameter 'targetLst'. Generating an XPath expression begins with the root element of the node which will be deleted. In Figure 5.3(a) the root element is *Publications* while in the Figure 5.3(b), the root element is *References*.

The XPath expression derived from the first tree is:

targetLst = /Publications/Publication[@PubID = "P111"]/Year

The XPath epression derived from the second tree is:

targetLst = /Refereces/Reference[@RefID = "R111"]/Publication
[@rlink-href = "//Publication[@PubID = 'P222']"]

More examples for creating the XPath expression as a parameter of the propagate functions are shown in Appendix B.

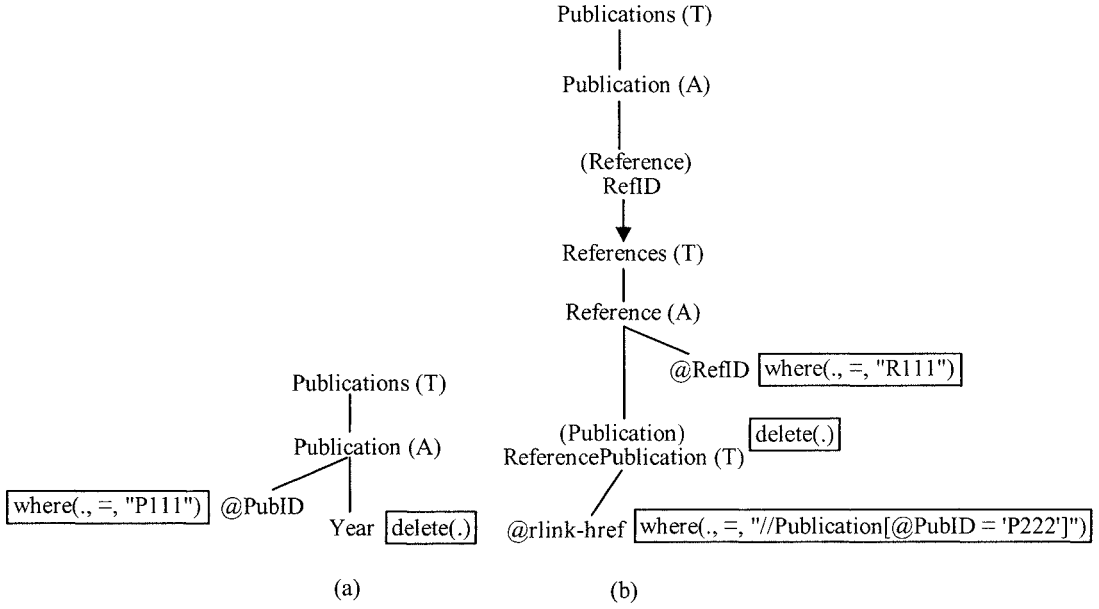

Figure 5.3 Trees of (a) first and (b) second delete clauses after adding conditions

5.5 Logical statements for updating XML documents

This section describes logical statements for our approach in updating XML documents explained in Chapters 3 and 4 and this chapter. Logical statements show that the data values in XML documents and the data values stored in an ORDB will be equivalent after updating.

This section shows only the logical statements in the case of updating an element or attribute converted to a field. Other cases of logical statements are shown in Appendix B.

To show the equivalence of XML data and ORDB data, we start by defining XML data and ORDB data converted from the XML data to show the initial state of the data. Then the logical statement for each type of an update operation (insert/replace/delete) performed on XML is shown as follows:

- First, an update operation performed on the XML side is translated into an update operation performed on the ORDB side by using an update-rule defined in section 4.2.2 in Chapter 4. Then the result after updating the ORDB is shown.
- Then, by using a return-rule defined in section 5.2 in this chapter, the condition of the update operation performed on the XML side is replaced with the key-condition derived from the return-rule. The result after updating the XML is shown.
- Finally, the results from the ORDB side and the XML side are shown to determine that their values are equivalent.

The symbols used in the logical statements are shown in Figure 5.4. Notations used in the logical statements are defined as follows.

Define:

$X_{schema} = \{rt, E_1,...E_n, A_1,...,A_n\}$, E: element, A: attribute, rt is root element

$D_{schema} = \{R_1,..., R_n\}$, R = Relation or Table

R = {C+}, C is column which can be field, nested table (NT) or abstract data type (ADT).

Symbol	Meaning
$\rightarrow$	is converted to
$\Rightarrow$	produces the result as follows
A ≡ B	data value of A is equivalent to data value of B
{exp}	Set of exp
exps	exp | {exp}
`Ins`[Obj, exps1, exps2?]	insert exps1 into Obj where exps2 The exps2 is a predicate and ? has the usual meaning of being optional
`Upd`[Obj, exps1, exps2?]	In XML context, update Obj with the value of exps1 where exps2
`Upd`[Obj, exps1, exps2?]	In database context, update fields of Obj specified in exps1 with the value of exps1 where exps2
`Del`[Obj, exps?]	delete the Obj where exps
`Sel`[Obj1(Obj2)exps?]	select Obj1 from Obj2 where exps

Figure 5.4 Symbols used in Logical Statements

Logical statements for updating an element/attribute converted to a field

After updating an element or attribute converted to a field, the data values in XML documents are equivalent to the data values in the ORDB. Logical statements to express this are as follows.

Definition:

$E = \{o_k, o_1, o_2?\}$ where o is a simple element/attribute, o_k is ID of E.

If E is converted to R Then $R = \{o_k, o_1, o_2\}$

If Instance of $E = \{e_1, e_2\}$ and

$e_1 = E\{(o_k, v_{k1}), (o_1, v_{11})\}$, $e_2 = E\{(o_k, v_{k2}), (o_1, v_{12}), (o_2, v_{22})\}$ where v is value and

If instance of $R = \{r_1, r_2\}$

Then $r_1 = \{v_{k1}, v_{11}, \text{null}\}$, $r_2 = \{v_{k2}, v_{12}, v_{22}\}$

Thus the initial states of data value on the XML side and the ORDB side are equivalent

XML side		**ORDB side**
$e_1 = E\{(o_k, v_{k1}), (o_1, v_{11})\}$	$\equiv$	$r_1 = \{v_{k1}, v_{11}, \text{null}\}$
$e_2 = E\{(o_k, v_{k2}), (o_1, v_{12}), (o_2, v_{22})\}$	$\equiv$	$r_2 = \{v_{k2}, v_{12}, v_{22}\}$

I) Logical statement for an insert operation on XML:

XML: $\texttt{Ins}[E, (o_2, v_{21}), o_1 = v_{11}]$

From update-rule 1: if an insertion is performed on a node converted to a simple field, the insert operation is converted to an update operation in the database context.

r_1 (converted from e_1), an instance of R where the primary key is v_{k1}, is the row containing $o_1 = v_{11}$; thus o_2 in r_1 is updated as follows:

XML: $\texttt{Ins}[E, (o_2, v_{21}), o_1 = v_{11}] \rightarrow$ **ORDB:** $\texttt{Upd}[R, o_2 = v_{21}, o_1 = v_{11}] \Rightarrow r_1 = \{v_{k1}, v_{11}, v_{21}\}$

From return-rule 1: If fields are updated, the values for the primary key of updated rows are returned.

e_1, an instance of E contains $o_k = v_{k1}$; thus o_2 is inserted into e_1 as follows:

XML: $\texttt{Ins}[E, (o_2, v_{21}), o_1 = v_{11}] \rightarrow$ **XML:** $\texttt{Ins}[E, (o_2, v_{21}), o_k = v_{k1}] \Rightarrow e_1 = E\{(o_k, v_{k1}), (o_1, v_{11}), (o_2, v_{21})\}$

Thus: After insertion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$e_1 = E\{(o_k, v_{k1}), (o_1, v_{11}), (o_2, v_{21})\}$	$\equiv$	$r_1 = \{v_{k1}, v_{11}, v_{21}\}$

II) Logical statement for a replace operation on XML:

XML: $\mathtt{Upd}[o_1, (o_1, v_{13}), o_2 = v_{22}]$

From update-rule 1: if a replacement is performed on a node converted to a simple field, the replace operation is an update operation on the field in the database context.

r_2 (converted from e_2), an instance of R where the primary key is v_{k2}, is the row containing $o_2 = v_{22}$; thus o_1 in r_2 is updated as follows:

XML: $\mathtt{Upd}[o_1, (o_1, v_{13}), o_2 = v_{22}]$ → **ORDB:** $\mathtt{Upd}[R, o_1 = v_{13}, o_2 = v_{22}] \Rightarrow$
$r_2 = \{v_{k2}, v_{13}, v_{22}\}$

From return-rule 1: If fields are updated, the values for the primary key of updated rows are returned.

e_2, an instance of E contains $o_k = v_{k2}$; thus o_1 in e_2 is updated as follows:

XML: $\mathtt{Upd}[o_1, (o_1, v_{13}), o_2 = v_{22}]$ → **XML:** $\mathtt{Upd}[o_1, (o_1, v_{13}), o_k = v_{k2}] \Rightarrow$
$e_2 = E\{(o_k, v_{k2}), (o_1, v_{13}), (o_2, v_{22})\}$

Thus: After replacement, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$e_2 = E\{(o_k, v_{k2}), (o_1, v_{13}), (o_2, v_{22})\}$	≡	$r_2 = \{v_{k2}, v_{13}, v_{22}\}$

III) Logical statement for a delete operation on XML:

XML: $\mathtt{Del}[o_1, o_2 = v_{22}]$

From update-rule 1: if a deletion is performed on a node converted to a simple field, the delete operation is converted to an update with null in the database context.

r_2 (converted from e_2), an instance of R where the primary key is v_{k2}, is the row containing $o_2 = v_{22}$; thus o_1 in r_2 is updated with null as follows:

XML: $\mathtt{Del}[o_1, o_2 = v_{22}]$ → **ORDB:** $\mathtt{Upd}[R, o_1 = \text{null}, o_2 = v_{22}] \Rightarrow$
$r_2 = \{v_{k2}, \text{null}, v_{22}\}$

From return-rule 1: If fields are updated, the values for the primary key of updated rows are returned.

e_2, an instance of E contains $o_k = v_{k2}$; thus o_1 in e_2 is deleted as follows:

XML: $\mathtt{Del}[o_1, o_2 = v_{22}]$ → **XML:** $\mathtt{Del}[o_1, o_k = v_{k2}] \Rightarrow$
$e_2 = E\{(o_k, v_{k2}), (o_2, v_{22})\}$

Thus: After deletion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$e_2 = E\{(o_k, v_{k2}), (o_2, v_{22})\}$	≡	$r_2 = \{v_{k2}, \text{null}, v_{22}\}$

To conclude, a comparison of the ORDB and XML equivalence show that in all cases, ORDB data and XML data are equivalent. This shows that mapping the schema, translating the language and propagating the change in this work are fully faithful.

5.6 Summary

This chapter described how to reflect the changes from an ORDB to XML documents. Section 5.1 began by describing the propagation of the changes from an ORDB to XML documents by using the values for keys of updated rows in the ORDB to indicate objects which should be updated in XML documents. The propagate functions are presented to provide the process for updating XML documents. Section 5.2 provided rules for returning the values for the key from the ORDB for each update operation. This section summarised the update operations performed on the XML side and the ORDB side as well as the values returned from the ORDB, the receivers on the XML side receiving the returned values from the ORDB and the conditions used to locate the target or reference position for updating. Section 5.3 described the step for creating an XPath expression as the value of a parameter of the propagate functions, to locate the target or reference positions for updating, and section 5.4 gave an example for creating the XPath expression. Finally, section 5.5 expressed the logical statements for updating XML documents to show that after updating by using our approach, the data values in ORDB are equivalent to the data values in XML documents.

In Chapter 6, the next chapter, the tools used in developing the prototype will be discussed. The architecture of the prototype will also be depicted. Finally, the way in which components in the prototype architecture communicate with each other will be clarified.

Chapter 6: Implementation

Chapters 3-5 described a solution for updating XML documents via an ORDB. The aim of this chapter is to explain how to implement the prototype for the solution. The details of the implementation for translating the XML update language into SQL proposed in Chapter 4 and reflecting the change from an ORDB to XML documents described in Chapter 5, will be explained. Mapping XML documents to an ORDB presented in Chapter 3 has not been implemented since it is not involved in evaluating the performance of XML update processing. However the overview of the implementation for mapping XML documents to an ORDB will be outlined.

This chapter is organised as follows. Section 6.1 discusses the reasons for choosing tools used for the implementation. Section 6.2 illustrates a high level architectural design showing the main modules of the prototype including mapping XML documents to an ORDB. Section 6.3 depicts components of the main modules of the prototype and explains how the components work and communicate with each other. Section 6.4 provides a summary of the chapter.

6.1 Tools used for the implementation

Six tools: Java, ANTLR, dom4j, Xerces, Jaxen and Oracle, are involved with the implementation of the prototype as follows. Java, a programming language is used for the basic coding. ANTLR, a parser generator, is used to parse the XML update language. Dom4j is an XML API while Xerces is an XML parser. Both are used to update XML documents via DOM trees. Jaxen, an XPath engine, is plugged into Xerces. Oracle is an ORDBMS used for storing XML data. The reasons for choosing these tools are given below.

6.1.1 Programming languages for implementing the prototype

Although a multitude of programming languages are currently used, Java is chosen to be the language for underpinning the prototype since Java can be run on any platform (platform independent) and several benchmarks [64] show that Java has acceptable performance.

6.1.2 Tools for parsing languages

Parser generators make it easier to create a compiler for a computer language. By using a parser generator, rather than developing the software parsing and checking the grammar of the language, we are involved in exclusively designing the grammar in a formal language definition notation such as BNF (Backus Naur Form) or EBNF (Extended Backus Naur Form) and specifying actions that should be performed when each part of the language is found. Many parser generators have been developed. For example, YACC [71] and PRECC [22] generate parsers written in C. JavaCC [68] and ANTLR [104] are the most popular parser generators creating parsers written in Java; thus both are the candidates for implementing the prototype.

ANTLR is open source while JavaCC is a commercial product; thus the source code of JavaCC is not available; however its run-time system is free for use. Actually there are no major differences between the capabilities of ANTLR and JavaCC. However, ANTLR provides guide documents which are much better than JavaCC and the grammar syntax of ANTLR is closer to EBNF than JavaCC; moreover, ANLTR has more powerful error reporting than JavaCC. ANTLR is therefore chosen for the implementation.

6.1.3 Tools for updating XML documents

API (Application Programming Interface) and XML Parser are tools facilitating the XML updates. Our approach for updating needs an API or an XML parser with the abilities for validating DTD and updating XML's elements, and providing an XPath engine or a mechanism that an XPath engine can be plugged into.

There are many well-known APIs and XML parsers. However only Xerces [10], JDOM [69] and dom4j [41] have the capabilities for updating XML's elements and validating DTDs correctly. In comparing their performance [122], dom4j consumes the least memory and has the best performance for parsing documents. Furthermore, dom4j includes in itself Jaxen, an XPath engine; thus dom4j is chosen for XML updates. Unfortunately dom4j does not support an insert-before/after operation; thus this function becomes the responsibility of Xerces. Xerces is chosen instead of JDOM because the main job of the prototype is modifying XML documents and Xerces has the best performance for this job while JDOM has the worst performance.

Xerces does not incorporate an XPath engine in itself; thus it needs a plug-in XPath engine. There are three well-known XPath engines: Jaxen [38], Xalan [9] and Saxon [72]. Saxon works with JAXP but does not work with other DOM implementations like Xerces or Crimson. Xalan and Jaxen work with Xerces. When their performances are compared [16], Jaxen outperforms Xalan; hence Jaxen is chosen to be a plug in for Xerces.

To conclude, dom4j is used to implement insert, update and delete operations while Xerces is used for an insert-before/after operation and Jaxen is a plug-in for Xerces.

6.1.4 An ORDBMS for storing XML data

Three well known RDBMSs: Oracle, DB2 and Informix, provide object-relational extensions; however, each DBMS provides different object-relational features. Since the XML structure is hierarchical and XML documents are governed by constraints, a DBMS providing the capabilities for keeping this structure and defining constraints on the storage should be suitable. DB2 does not provide a collection type; thus it cannot be used for hierarchical structured data. Informix supports list/set but no constraint can be defined on these constructions. Oracle supports nested tables and every constraint except the referential integrity constraint can be defined on the nested tables. Thus Oracle is chosen as the ORDBMS for storing XML data for the prototype.

6.2 Architectural design of the prototype

At a high level of architectural design, the prototype consists of three modules: XML-ORDB Mapper, Language Translator and Change Reflector, as shown in Figure 6.1.

Although the XML-ORDB Mapper has not been implemented since it is not involved in evaluating the performance of XML update processing, its implementation will be outlined here as well to show its connection to the other modules. The detail of its components will be omitted in the next section.

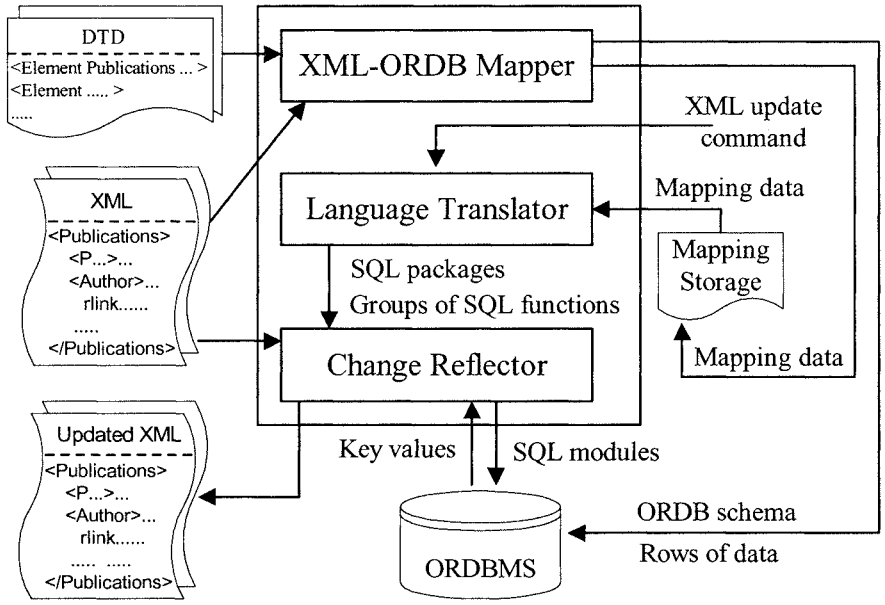

Figure 6.1 Architectural design of the prototype

6.2.1 XML-ORDB Mapper

The XML-ORDB Mapper has two major functions: mapping DTDs to ORDB schemas and loading XML data to the ORDB.

For the first function, the mapper takes XML DTDs as input and produces ORDB schemas and mapping data as output. The mapper parses every complex element in the DTDs into pieces of objects that are convenient for keeping and accessing such as element name, path of element, list of child-element, list of attribute and referenced element. The classes for storing these objects are defined in Figure 6.2.

```
Class DTDs
{  List<Document>       ListOfDoc
}

Class Document
{ String                 Name
  List<ComplexElement> ListOfComplexEle
}

Class ComplexElement
{  String              Name
   String              ID
   Char                Occurence
   List<ChildElement> ListOfChild
   List<Attribute>    ListofAttr
   RefAttribute        ReferencingAtt
   String              Path
   List<RecursiveEle> ListOfRecursive
   List<Rlink>         ListOfRlink

}

Class ChildElement
{  String              Name
   Char                Occurence
}
```

```
Class Attribute
{  String          Name
   String          Constraint
   String          DefValue
   List<String> ListOfOptValue
}

Class RefAttribue
{  String          Name
   String          Type
   String          ReferencedEle
   String          ReferencedDoc
   String          Constraint
}

Class RecursiveEle
{  String          Name
   Char            Occurence
}

Class Rlink
{  String          LinkingEle
   Char            Occurence
   String          LinkedDoc
   String          LinkedEle
}
```

Figure 6.2 Classes for storing objects derived from parsing DTDs

The detail of the **Class ComplexElement** in Figure 6.2 is as follows:

Name: The name of complex elements.

ID: The name of the attribute whose type is ID.

Occurrence: The number of appearances of the element in the parent-element is expressed by occurrence symbols: *, +, ? and 1.

ListOfChild: A list of child-elements consists of child-name and occurrence symbol. This field keeps child-elements which are not recursive-elements or rlink-elements.

ListOfAttr: A list of attributes consists of attribute name, constraint: REQUIRED or IMPLIED, default value and list of optional values. This field keeps attributes whose type is not ID or IDREF(s).

ReferencingAttr: The referencing attribute consists of the name of the referencing-attribute, the name of the referenced element, the name of the referenced document, the reference type: IDREFs or IDREF and constraint: REQUIRED or IMPLIED.

Path: Path is an absolute path starting from the root element to the complex-element.

ListOfRecursive: A list of recursive elements consists of the name of descendent elements that refer back to the complex element and the occurrence symbol of the descendent elements.

ListOfRlink: A list of rlink-elements consists of a linking element with its occurrence symbol, linked document and linked element.

To determine the ORDB structure from the data kept in these classes by using the mapping rules described in Chapter 3, the order of processing the elements kept in the classes is ordered from complex-elements having the longest path to the shortest path. The information about how a particular element or attribute is mapped to the ORDB structure is kept in mapping storage. The mapping storage keeps the mapping data persistent to avoid re-parsing the same DTDs and to use this data when XML data is loaded to the ORDB and when an XML update command is translated into SQL.

For the second function of the XML-ORDB Mapper, the mapper takes XML documents as input and produces rows of data for each table in the ORDB as output. The mapper parses the documents into tree models. During parsing, XML data is validated to assure that it conforms to XML DTDs. The mapper uses data retrieved from the mapping storage to generate appropriate rows of data for the ORDB.

6.2.2 Language Translator

The Language Translator is responsible for translating the XML update language into SQL. It takes an XML update command as input and produces SQL packages and groups of SQL functions as output. Each SQL package consists of SQL commands and a function number derived from the SQL functions. For the steps in the translation, the Language Translator parses an XML update command to create SQL functions for each clause of the update command. It then creates trees of these functions and maps the trees to the ORDB schema graph. It separates the trees into sub-trees according to the number of update operators performed on different tables having different function numbers. Finally, it optimises these sub-trees, generates SQL commands from the sub-trees and packs the SQL commands with the corresponding function numbers together.

6.2.3 Change Reflector

The Change Reflector takes responsibility for reflecting the change from an ORDB to XML documents. It takes the SQL packages and the groups of SQL functions, derived from the Language Translator, and XML documents as input and produces updated XML documents as output. The Change Reflector unpacks SQL packages and then the capability for extracting values for keys of updated rows in the ORDB is incorporated into SQL commands to produce SQL modules. These modules are executed on the ORDB and the values for the keys of the updated rows are returned. The values for the keys of the updated rows and the SQL functions are used to compose the propagate functions used for updating the XML documents.

6.3 Components of the prototype

This section illustrates the details of the components for the Language Translator and Change Reflector. It explains how these components work and communicate with each other.

6.3.1 Components of the Language Translator

Usually programming languages consist of keywords, symbols, operators and strictly defined constructs. To make computers recognise these components, rules defining the meanings of the components of a language must be specified. The rules for the XML update language are defined using EBNF, a formal language definition notation. Two examples from defining rules for keywords and a construct of the XML update language are shown in Figures 6.3(a) and 6.3(b) respectively. (Actually in practice we use ':' instead of ':=' since the tool used for parsing the language accepts ':' rather than ':=').

```
IDENTIFIER      := ('a' .. 'z') ( 'a' .. 'z' | '0' .. '9' | '_' | '.' | '/' | ':')*
OPEN_PAREN      := '('
CLOSE_PAREN     := ')'
DB_QUOTE        := '"'
```

(a)

```
document := "doc" OPEN_PAREN DB_QUOTE IDENTIFIER DB_QUOTE CLOSE_PAREN
```

(b)

Figure 6.3 Rules for (a) keywords and (b) a construct

The Language Translator consists of six components as shown in Figure 6.4. The details of these components are as follows:

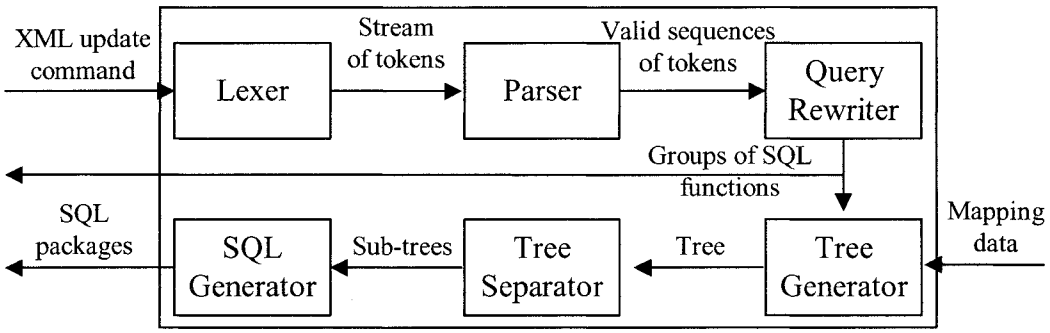

Figure 6.4 Components of Language Translator

Lexer

When an XML update command is processed, it is interpreted as an arbitrary stream of characters. The function of Lexer (or Lexical analyzer) is to quantify this arbitrary stream into discrete groups. Each group is defined by a production rule in EBNF; thus the groups have meaning as they are defined in the rules. Each group identified according to the rule is called a token, a component of a language such as a keyword, symbol or operator. In other words, the Lexer converts a stream of characters to a stream of tokens having individual meaning as dictated by their rules. The Lexer removes white spaces from the XML update command and will generate errors when it finds a stream of characters which it cannot match to a particular token defined by the rules. The stream of tokens will be sent to the Parser.

Parser

The Lexer just groups a stream of characters which it recognises into a stream of tokens having individual meaning for each token but the Lexer does not consider the semantics

of an XML update language as a whole; thus this function is the responsibility of the Parser. As mentioned earlier, a language consists of several components such as keywords and constructs; thus to create a complete clause or a complete command of the language, the clause or the command must consist of a valid sequence of these components. This valid sequence is the grammar of the language. In Figure 6.3(b), the grammar of 'document' is:

"doc" OPEN_PAREN DB_QUOTE IDENTIFIER DB_QUOTE CLOSE_PAREN

Thus if we send a clause as follows:

OPEN_PAREN DB_QUOTE IDENTIFIER DB_QUOTE CLOSE_PAREN "doc"

The Lexer will not send any error since each token is a valid token as defined in the rules already but the Parser will send an error since this clause contains an invalid ordering of valid tokens.

The job of the Parser is to organise a stream of tokens it receives into allowed sequences defined by the grammar rules of the language. During validation of an XML update command, some clauses or tokens in the command will be transformed according to the rules specified in Chapter 4. For example, 'some $a' is transformed to 'count($a) > 0'. Hence only valid sequences of tokens are sent as output to the Query Rewriter. To conclude, the Lexer recognises streams of characters while the Parser recognises streams of tokens.

Query Rewriter

Query Rewriter rewrites the valid sequences of tokens derived from the Parser as SQL functions. Each sequence of tokens is one clause in the XML update command. Each clause that is defined with a particular function number will be called an SQL function. Every *Update* clause and its own condition have the same function numbers starting from 1. The funcNo 0 will be assigned to a *For* clause, *Let* clause and *Where* clause of the *For/Let* clause (*Where* clause does not belong to any specific *Update* clauses). These clauses will be shared for the *Update* clauses. Clauses having the same function number are in the same group. Each SQL function is kept in an *Array* where each cell stores a token of the clause or a function number. Figure 6.5(a) shows an XML update command and Figure 6.5(b) shows how the SQL functions are stored in *Arrays*. Query

Rewriter produces groups of SQL functions held in *Arrays* as output and sends them to Tree Generator and the Change Reflector module.

```
For $p in doc("Library.xml")//Publication,
    $a in $p/Author
Where  $p/Title = "Java"
Delete $p/Year,
Delete $a
Where  $a/Name/LName = "Ian"
```

(a)

'Library.xml'	bindF	/Library/Publication	$p		0
	bindF	/Library/Publication/Author	$a		0
	where	$p/Title	=	'Java'	0
	delete	$p/Year			1
	delete	$a			2
	where	$a/Name/LName	=	'Ian'	2

(b)

Figure 6.5 (a) An XML update command, (b) Six Arrays storing six SQL functions

Tree Generator

Tree Generator generates a tree from the SQL functions derived from the Query Rewriter. Paths in the tree are created according to the paths in the SQL functions. Each node of the tree keeps all the tokens of every SQL function whose paths correspond to the path of the node. The structure variable, which is used to create the tree and a node of the tree, is *class*. The *class* of a node provides attributes for storing all data of SQL functions and provides methods for accessing and modifying the data of these functions. The *class* of a tree provides an attribute 'root' whose type is node-class and provides methods for creating the root, adding a child node and other methods for traversing the tree. The *interfaces* for the tree and the node are shown in Figure 6.6 and the implementations for the *interfaces* are shown in Figure 6.7 (an *interface* is a *class* that is not implemented).

```
public interface Tree
{
        public Tree.Node makeRoot(Object element);
        public Tree.Node root();

        public Tree.Node addChild(Tree.Node node, Object element);
        public void remove(Tree.Node node);

        public Tree.Node parent(Tree.Node node);
        public Tree.Node firstChild(Tree.Node node);
        public Tree.Node nextSib(Tree.Node node);

        public Tree.Node childByName(Tree.Node node, String name);
        .....

        public interface Node
       {
         private Object element;       //keep full path of node
         private LinkedTree.Node firstChild;
         private LinkedTree.Node parent;
         private LinkedTree.Node nextSib;

         private Vector vOper ;  //Operation: insert/update/delete/
                                 //where/and/or
         private Vector vPara;  //Number of para depend on operation
         private Vector vFuncNo;

         /** Mapping database structure**/
         private String dbType;
         private String belongToTable;
         private String fieldOrder;

         private boolean PK;           // true /false
         private boolean FK;           // true /false

         private Vector vReferredFrom; // this PK is referred from
                                        // which node
         private Vector vReferringTo;  // this FK refers to which node

         private boolean remove;  // set in the step of optimisation

         public String getDBType();
         public void   setDBType(String type);
         public String getBelongToTable();
         public void   setBelongToTable(String table);
         ....
       }
}
```

Figure 6.6 Interfaces of Tree and Node

```
public class LinkedTree implements Tree
{
        private LinkedTree.Node root;
        ...

        public LinkedTree()
       {
           root = null;
       }

        public Tree.Node makeRoot(Object element)
       {
           root = new LinkedTree.Node(element);
           return root;
       }

        public Tree.Node addChild(Tree.Node node, Object element)
       {
           LinkedTree.Node lastChild;
           LinkedTree.Node parent = (LinkedTree.Node) node;
           LinkedTree.Node newChild = new LinkedTree.Node(element);
           newChild.parent = parent;

           if (parent.firstChild == null)
           {
                 parent.firstChild = newChild;
                 newChild.nextSib = null;
           }
           else
           {
                 lastChild = parent.firstChild;
                 while (lastChild.nextSib !=null)
                lastChild = lastChild.nextSib;

                 lastChild.nextSib = newChild;
                 newChild.nextSib = null;
           }
           return newChild;
       }

        private static class Node implements Tree.Node
       {
           private Object element;  //keep absolute path of element
           ...
           private Node (Object elem)
           {
                this.element = elem;
                ...
           }
           ...
       }
}
```

Figure 6.7 Implementation of the interface of Tree

Only some main attributes and main methods are shown in the *interfaces* and their implementation.

After creating a tree for the SQL functions, the ORDB schema graph is mapped to the tree. Mapping uses the data from the mapping storage to specify the types of ORDB

structures on the tree. Then primary keys and foreign keys which are used to join tables are added to the tree. The information about the ORDB structures is kept in each node of the tree. From Figure 6.6, the *class* of a node provides attributes for storing the ORDB structures such as primary keys and foreign keys used for joins of tables. Finally the meaning of update operations on the tree will be revised according to the update rules defined in Chapter 4.

Tree Separator

Tree Separator will split the tree derived from the Tree Generator into sub-trees. After splitting the tree, the number of sub-trees is equal to the number of update operations performed on the tables with different function numbers. Thus splitting the tree begins with grouping the tree's nodes according to the types of update operations which have different function numbers. The conditions for each update operation will be in the same group as its update operation. Each group contains all nodes which will be created as a new sub-tree. All the addresses of the nodes in each group are kept in one *Vector* whose behaviour is similar to a *dynamic Array*. Then each sub-tree is created from the nodes' information pointed at by the *Vector*. As for the tree, a tree-class is used to create each sub-tree. Finally, optimisation is performed on the sub-trees according to the optimisation rules specified in Chapter 4. Figure 6.8 shows how the *Vectors* keep addresses of nodes in the tree of the example in Figure 6.5. In Figure 6.8, a delete operation on *Year* is changed to an update operation since *Year* is a field in the ORDB; thus deleting a field means updating that field with null.

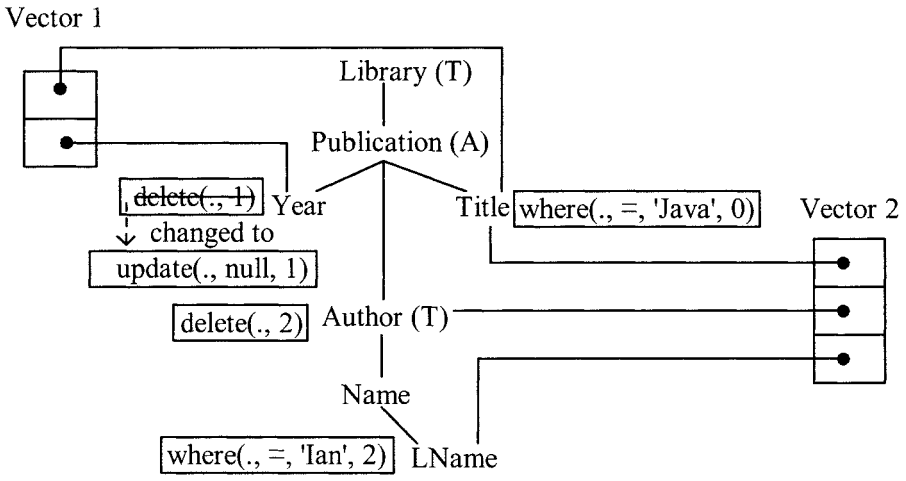

Figure 6.8 Two Vectors storing all addresses of nodes for two new sub-trees

SQL Generator

SQL Generator generates SQL packages from the sub-trees derived from the Tree Separator. The SQL generator starts creating each SQL command by collecting all related tables from each sub-tree. Then the update operation in a sub-tree is searched to generate an *update/delete/insert...select* clause. After that, a *from* clause is generated by using the collected tables and the conditions of the command are generated. Finally, joins between primary keys and foreign keys are generated. During the generation of each clause of the command, if the delete/update is related to more than one table, some clauses of the command will be rewritten as update/delete clauses with sub-queries according to the SQL rewriting rules specified in Chapter 4. Each generated SQL command and the function number of its sub-tree will be packed together and this SQL package will be sent to the SQL Executor in the Change Reflector module.

6.3.2 Components of Change Reflector

The Change Reflector consists of five components as shown in Figure 6.9. The details of these components are as follows.

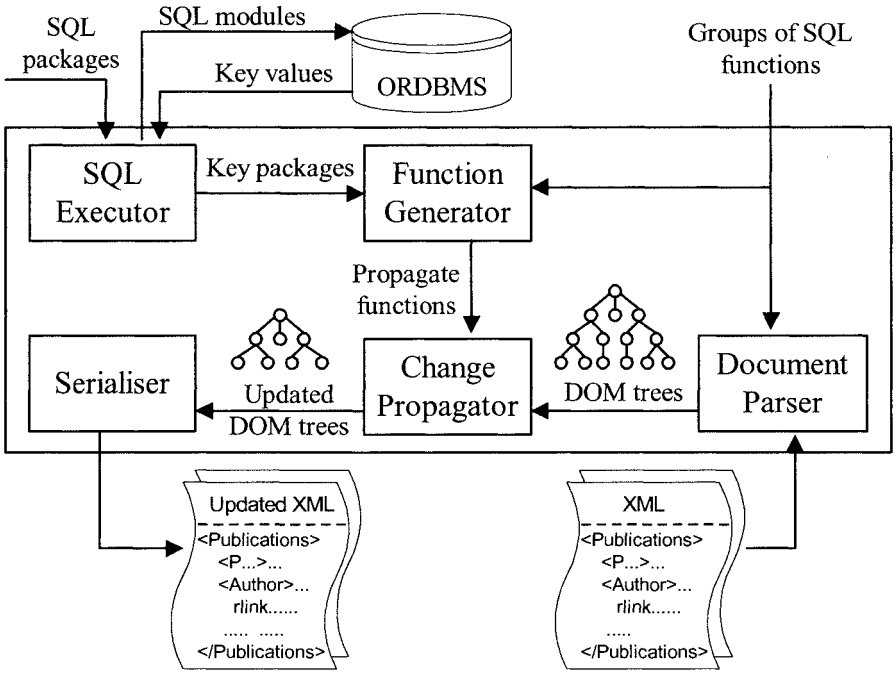

Figure 6.9 Components of Change Reflector

SQL Executor

SQL executor receives the SQL packages sent from the SQL Generator. The SQL Executor unpacks the packages to retrieve the SQL commands and the function numbers of the SQL commands from the packages. The SQL commands are embedded with the capability to extract the values for the keys of the updated rows from the ORDB and, as SQL modules, they are then sent to the ORDBMS. After executing SQL modules on the ORDBMS, the values for the keys are returned to the SQL Executor. The SQL executor packs the returned values of each SQL module with the function number retrieved from the same package as the SQL command in the SQL module. After packing the returned values for the keys with the function numbers, these Key packages are sent to the Function Generator.

Function Generator

Function Generator generates the propagate functions from the Key packages derived from the SQL Executor and the group of SQL functions derived from the Query Rewriter. Two parameters of the propagate functions are data for updating and target/reference positions for updating. The data for updating can be derived directly from the SQL functions while the target/reference positions for updating are specified by XPath expressions.

The Function Generator generates XPath expressions from the values for the keys kept in the Key packages and the paths kept in the SQL functions. To generate XPath expressions, the Key packages are unpackaged to retrieve the values for the keys and the function numbers from the packages. The Function Generator starts by creating each tree from paths in each group of SQL functions and then adding the update operators to the trees to indicate which nodes will be retrieved from the XPath expression as the target/reference positions for updating. The conditions of the *before/after clause* and conditions on fields of a nested table, IDREFs and rlink-element derived from the SQL functions are added to the trees by using the function numbers to indicate which condition should be added to which tree. The ORDB schema is mapped to the trees. The Function Generator uses the function numbers derived from the Key packages to indicate the trees for which the values for the keys are to be added as key-conditions on the trees. The receivers (nodes on the trees) for the values of the

keys are determined by rules in Table 5.1 in Chapter 5. Finally an XPath expression is generated from each tree.

The data for updating XML documents and the update operators derived from the SQL functions and the generated XPaths are composed as the propagation functions. Each part of a propagate function is held in an element of Vectors. Figure 6.10 shows two propagate functions for the example in Figure 6.5 kept in two Vectors. The propagate functions kept in the Vectors are sent to the Change Propagator.

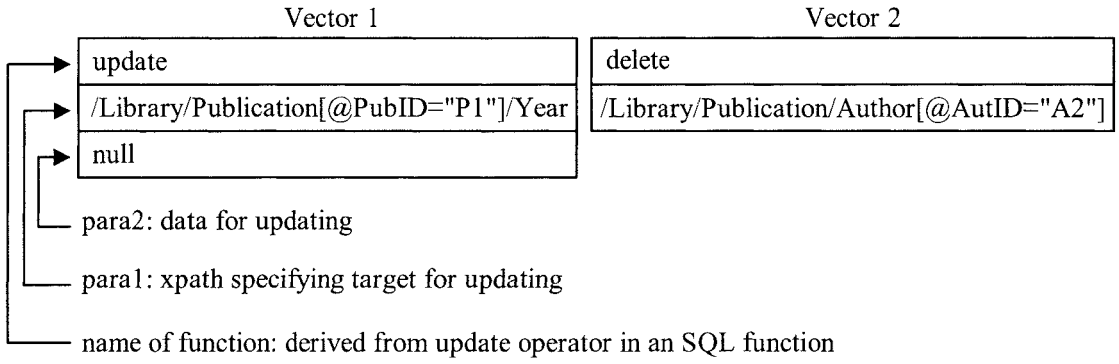

Figure 6.10 Two propagate functions held in Vectors 1 and 2

Document Parser

Document Parser parses XML documents affected by the updating into DOM trees by using information on the SQL functions derived from the Query Rewriter to indicate which documents are affected. The XML documents are parsed by using API of Dom4j or Xerces. Dom4j is used in the case of insert, update and delete operations while Xerces is used in the case of insert-before and insert-after operations. The DOM trees are sent to the Change Propagator.

Change Propagator

Change Propagator updates the DOM trees derived from the Document Parser with the propagate functions derived from the Function Generator. To enable the propagate function to update the DOM trees, the propagate functions are implemented by using APIs of Dom4j and Xerces.

To use these APIs, the data for updating, the second parameter of the propagate functions, must be encapsulated into nodes whose type is an element/attribute of Dom4j or Xerces. Dom4j's APIs can execute XPath expressions directly while Xerces calls Jaxen's APIs for executing XPath expressions. After an XPath expression is executed

on the DOM trees, the target/reference nodes in the DOM trees are returned. The target/reference nodes and the data encapsulated in the element/attribute are sent to the proper APIs determined by the name of the propagate functions to update the DOM trees. The updated DOM trees are sent to the Serialiser.

Serialiser

Serialiser arranges data of the updated DOM trees in serial order and records these data as XML documents. The updated DOM trees are serialised by using APIs of Dom4j or Xerces. If the XML documents are parsed by Dom4j, they are serialised by using Dom4j. If the XML documents are parsed by Xerces, they are serialised by using Xerces.

6.4 Summary

This chapter started by discussing the reasons for choosing six tools: Java, ANTLR, Dom4j, Xerces, Jaxen and Oracle, to implement the prototype in section 6.1. The major criteria for choosing the tools are their respective capabilities and performance while supportive documents are a minor criterion. Section 6.2 illustrated a high level design for the architecture of the prototype. The architecture consists of three main modules: XML-ORDB Mapper, Language Translator and Change Reflector. XML-ORDB Mapper is not implemented since it is not involved in assessing the performance of XML update processing; however the outline approach for implementing it is provided. This does mean though that the initial data in the ORDB for experimental study in Chapter 7 is hand-written. Section 6.3 depicted the main components of the Language Translator and the Change Reflector. In this section, the details for implementing each component and structure variables used for storing data during processing are explained.

In Chapter 7, the next chapter, an experimental study will be conducted to evaluate the performance of the prototype in several aspects such as the effect of caching data or redundant data.

Chapter 7: Experimental Study

The implementation of the prototype has been explained in the previous chapter. The objectives of this chapter are to describe the experimental setup and explain how the performance of the prototype is evaluated including a discussion of the experimental results.

To the best of our knowledge, the evaluation of XML processing on all data caching states, namely cold cache, warm cache and hot cache, have never been conducted before.

This chapter is organised as follows. Firstly, the objectives of the experiments are described in section 7.1. Secondly, how to set-up the experiments as well as the data set and the update features used in the experiments are explained in section 7.2. Thirdly, the details of the experimental results are discussed in section 7.3 and the conclusion of the results is provided in section 7.4. Finally, the summary of this chapter is provided in section 7.5.

7.1 Objectives of experimental study

The hypotheses stated in section 1.4 concentrate on the performance of updating redundant and non-redundant data (the problem of low performance caused by redundant data) and the performance of updating data by using regular path expressions (the problem of updating data whose structure is partially known) including the effect of caching data. The problems which are not related to the performance can be solved by our approach which will be concluded in section 8.1.

Besides testing the hypotheses, our experimental study is to measure the performance within a system in itself to compare the performance of update features and update operations in each system. Thus the experimental study serves five purposes as follows:

The first purpose is to evaluate the performance for updating a native XML database storing redundant data, an ORDB storing redundant XML data and an ORDB keeping

non-redundant XML data. In this experiment, several sizes of the databases containing multiple sizes of redundant data are compared.

The second purpose of the study is to determine whether or not caching data has an effect on the performance of a native XML database and an ORDB containing XML data. If it does, in which cache state the native XML database outperforms the ORDB and in which cache state the ORDB outperforms the native XML database are of interest. For this, the experiment is conducted in cold cache, warm cache and hot cache states.

The third purpose of our experimental study is to compare the performance of three update operations: replace, delete and insert within each system. This is to indicate which update operation has the best or worst performance when they are performed on the same data and the same state of caching, not only in a native XML database but in an ORDB containing XML data as well.

The fourth purpose is to compare the performance of update features within each system and to also compare the performance of update commands based on regular path expressions between the systems. In this case, we design update commands covering seventeen features that the XML update language should have. These seventeen features are based upon a list of "must have" requirements for XML query languages published by W3C [134].

The last purpose of the experimental study is arguably the most important, that is to check the correctness of the work described here in translating the XML update language into SQL. The correctness is checked by comparing the results of translating the XML update language obtained via the prototype and our expected results. For this, additional XML update commands are designed to update data in different structures of an ORDB. Thus these additional update commands are designed according to the ORDB structure types instead of update features. The testing is conducted with these additional commands plus the original commands derived from evaluating the first four purposes.

7.2 Experiment platform and methodology

In these experiments, three types of databases are used. The first is X-Hive, a trial version of a commercial native XML database, used to keep redundant data of a single XML document. The second is an Oracle ORDB employed to keep redundant data of a single XML document (single XML document database). The third is an Oracle ORDB utilised to keep non-redundant data of linked XML documents (linked XML document database). The XML update language for X-Hive is XUpdate [140]; therefore our XML update language is translated into XUpdate to test X-Hive. However reference traversal and recursive features are not supported by X-Hive. All experiments are conducted on 1.3 GHz Pentium M machine with 768 MB main memory and 20 GB disk running Windows XP.

While the measurement of XML query processing is related to the design of data sets and sets of queries, in a similar way, the measurement of XML update processing is related to designing a data set and a set of appropriate update commands because for an internal process to update the target data, the data must first be searched by the query engine. We design a data set and a set of update commands that adhere to the criteria of a benchmark design [53] namely relevance, scalability and simplicity. For portability, a criterion of benchmark design, we will omit this aspect since we do not develop a benchmark but a method to evaluate our XML updating approach. The criteria of the benchmark design are interrelated and can conflict with each other. For example, if the test data is too simple (simplicity), it may not be able to capture the ability of XML to represent complex structures (relevance) whereas if the structure of the XML data is too complex (relevance), it may be difficult to understand the structure (simplicity) and to change the size of the data (scalability).

7.2.1 Data set design

The DTDs of the test data are designed to capture most essential features of XML data representation such as references, nesting, implicit order and recursive structure. The DTDs do not capture *mixed content* and *any* features since these constructions are quite rare in DTDs [32]. DTDs still provide semantics and a structure that are simple enough to understand. The DTD of a single XML document is shown in Figure 7.1.

```
<!ELEMENT Library (Publication*)>
<!ELEMENT Publication (Title, Author+, Year, Publisher, Reference?, Cost,
                      SpecialCost?, ShippingCost?)>
<!ATTLIST Publication PubID ID #REQUIRED>
<!ATTLIST Publication PubType (book | article | journal) #IMPLIED>
<!ATTLIST Publication ContactAuthor IDREF #REQUIRED>
<!ELEMENT Title(#PCDATA)>
<!ELEMENT Cost (#PCDATA)>
<!ELEMENT SpecialCost (#PCDATA)>
<!ELEMENT ShippingCost(#PCDATA)>

<!ELEMENT Author (Name, Email?,  Telephone*)) >
<!ATTLIST Author AuthorID ID #REQUIRED>
<!ELEMENT Name (FName, MName?, LName)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT MName (#PCDATA)>
<!ELEMENT LName (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Telephone (Location, TelNo)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT TelNo (#PCDATA)>
<!ELEMENT Year (#PCDATA)>

<!ELEMENT Publisher (PName, Address?, SetupYear)
<!ATTLIST Publisher PID ID #REQUIRED>
<!ELEMENT PName (#PCDATA)>
<!ELEMENT Address (No, Street, City, Country, Zipcode)>
<!ELEMENT No (#PCDATA)>
<!ELEMENT Stree(#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Zipcode (#PCDATA)>

<!ELEMENT Reference EMPTY>
<!ATTLIST Reference RefID ID #REQUIRED>
<!ATTLIST Reference RefPub IDREFs #REQUIRED>
<!ATTLIST Reference RefType (References | Bibliography | Miscelleneuos)
                            "References">
```

Figure 7.1. DTD of a single XML document

The scalability of the evaluation is specified by using data sets of varying sizes. The data model of XML is represented as a tree; thus the size of data can be scaled by several options: changing depth, fanout or number of nodes of the tree. The depth of a tree is defined by levels of tree and/or the number of repetitions of recursive elements. The fanout of a tree is specified by the number of child nodes for each internal node whereas the number of nodes is specified by the number of sub-elements (sub-trees) at the root level. The number of nodes denotes the length of an XML document. The depth, fanout and the sub-elements at the root level are shown in Figure 7.2.

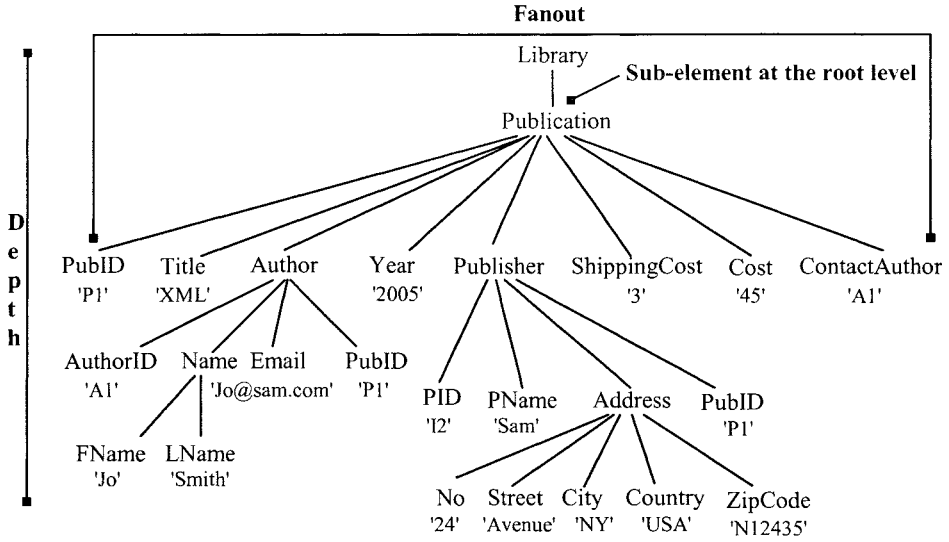

Figure 7.2. Depth, fanout and the sub-element at the root level

To vary the size of data, other researchers [21, 109] increase the depth and/or fanout of the tree to make the data size bigger; however with this method, the large number of data sets will make the system harder to understand because of structural change. Hence in our approach, the size of the data is changed by increasing the number of nodes which are sub-elements at the root level of the document. From Figure 7.2, the sub-element at the root level (Library) is Publication. Changing the size of data by increasing the number of sub-elements at the root level is applied as in other work [5] that increases the size of the object-database by increasing the number of *modules* (the top-level class).

By changing the number of nodes, there is no effect due to structural changes since tree fanout and tree depth are kept constant for all scaled versions of the data set. Therefore the impact of the number of nodes of the data tree can be identified. Another aspect of the data set upon which we focus in addition to data size is data redundancy. Thus in each scaled version of the data set, the number of redundant records in the target-records that will be updated can be changed so that the performance of updating one XML document containing redundant data can be compared to the performance of updating linked XML documents without redundant data. The linked XML documents keeping non-redundant data have a size that is smaller than one XML document containing redundant data because there is no redundant data in the linked XML documents.

7.2.2 Update command design

The performance of an XML query language depends upon the functionalities (expressive power) it provides. According to a list of "must have" requirements for XML query languages published by W3C [134], the capabilities of an XML query language can be categorised into three types: data-centric, document-centric and navigational. Similar to an XML query language, the performance of an XML update language should be evaluated from its expressive power as well.

We will evaluate most of the functionalities specified in the requirements of W3C except for supporting environment information which we do not implement and except for very specific functionalities of query commands such as "able to generate sorted data". In our work, the functionalities which will be evaluated are classified into two types: data-centric and document-centric. The navigational capabilities are included in the document-centric functionalities. The features of these two functionalities are summarized in Figure 7.3. The detail of each feature is described in Appendix C.

Data-centric functionalities		Document-centric functionalities	
C1	Exact Match	C10	Join documents in update
C2	Update without join of documents	C11	Navigation by reference traversal
C3	Change selectivity	C12	Handling missing elements
C4	Allow condition on text	C13	Element ordering
C5	Support aggregation	C14	Using regular path expression
C6	Support quantifiers	C15	Mix between data-centric and document-centric
C7	Joins based on values		
C8	Joins based on pointer	C16	Hierarchical and sequence update
C9	Casting	C17	Recursion and reference traversal

Figure 7.3 The 17 update-features for the experimental study

7.2.3 Experiment Design

The experiments are designed according to the objectives of the experimental study, which has been mentioned in section 7.1, as follows.

1) Varying size of data and varying degrees of redundant data are used in the experiment. We create a program to generate the test data. In this generation, the values of two parameters: size of data and the number of records of redundant data, can be changed. Before conducting the experiment, we performed testing on several data size and it is found that the data whose size starts from 5 MB can show obvious performance difference between the systems. The data size is doubled in each scaled version of the data since we want to assess the relationship between the elapsed time and the data size. However the maximum size is 40 MB because of resource limitations. Thus in the experiments, testing is conducted on data with size 5, 10, 20 and 40 MB that is 8000, 16000, 32000 and 64000 records respectively. These sizes are based on a single XML document. The number of redundant records for 5 MB data size varies from 10, 20, 40 to 80 records while the number of redundant records for 10, 20 and 40 MB data size is two times, four times and eight times respectively the number of redundant records for 5 MB data size. The size of linked XML documents is smaller than a single XML document since such documents do not contain redundant records.

   In updating the linked XML documents, each update command affects 10 records. Thus the number of records in a single document affected by a command varies according to the proportion of redundant records. For example, for the 5 MB data size, 100, 200, 400 and 800 records of a single XML document are affected. This design serves the first purpose of the experimental study.

2) Caching data in computer memory may affect execution time. There are three states of caching data: cold cache, warm cache and hot cache. A cold cache is when nothing is in memory, and all data is read from the disk; a warm cache is when the system has been running and most data are loaded in memory; and a hot cache is when the entire data set required is in memory [93]. Most of the benchmark programs are run in warm cache. Unrelated queries were run first, and then each of the tested queries was run once in turn [49, 121] to collect timings. However our experimental study will be conducted on all three states of caching data.

In a cold cache, the database is restarted for each individual update command. In the case of a warm cache, the database is restarted for each individual update command as well; however before running the update command, five unrelated commands will be run first. In some work [143] researchers use up to 10 query-commands to warm the cache. In a hot cache, the same command is run twice in succession and the performance measured for the second run. This supports the second purpose of the experiment.

3) We designed two sets of 44 update commands for two ORDBs storing XML data. Each set of commands consists of 14 replace commands, 14 delete commands and 16 insert commands for covering 17 features as mentioned in section 7.2.2. For the native XML database, 41 update commands, a subset of the 44 commands of the ORDB, are used. The number of commands for the native XML database is less than for the ORDB storage of XML data since XUpdate does not support recursion and navigation by reference traversal; thus there is no command for these features. In the XML update commands, some commands contain two features since some features are simple or always appear along with other features. In the experimental study, two features appearing along with other features are 'navigation by reference traversal' (C11) and 'update a particular document without join' (C2). The feature C11 appears along with the feature C17 (recursion) while the feature C2 appears in the various commands updating a particular document without a join to other documents. This design supports the third, the fourth and the last purposes.
4) An additional two sets of 15 update commands for two ORDBs storing XML data are designed for updating data in different ORDB structures to serve the last purpose in checking the correctness of translating our XML update language into SQL. These additional commands are based on ORDB structure types rather than update features. This means if an update operation is performed on a particular structure type, the predicate will be performed on that structure type as well.

In each experiment, the update command is performed on a particular database with a specific data size and specific records of data redundancy in each caching state. Each experiment is executed five times and the longest and the shortest elapsed times are discarded; thus only an average of three elapsed times is reported [109].

7.3 Discussion of the experimental results

The graphs and tables, to evaluate the performance of XML update processing and to check the correctness of XML update translation according to the purposes of the experimental study, are produced as follows.

7.3.1 Performance with different data redundancy and data caching

In this section, we produce twelve graphs for the first and the second purposes of the experimental study. The graphs show the effect of updating XML databases storing redundant data and non-redundant data including the effect of data caching in updating XML databases. The twelve graphs are presented as four graphs in Figure 7.4 for the replace operation, four graphs in Figure 7.5 for the delete operation and four graphs in Figure 7.6 for the insert operation. Each graph presents a high-level view of performance in term of average elapsed time of all update commands updating three types of XML databases for a particular data size ranging from 5, 10, 20 to 40 MB in four different sizes of redundant data and in three data caching states: cold, warm and hot. In the graphs, each update always affects 10 records of the linked XML document database (lxd); thus the number of redundancy records of the single XML document database (sxd) and the native XML database (nxd) are multiplied by 10.

The three types of the XML database shown in the graphs are nxd, sxd and lxd, where nxd stores redundant data, sxd is an ORDB storing redundant data while lxd is an ORDB keeping non-redundant data.

The commands C14 (regular path expression) and C17 (recursion) are excluded from the average time of nxd since X-Hive does not support command C17 while the elapsed time of C14 run on X-Hive is so long that it can affect the overall performance of the systems. The command C3 (change selectivity) is also not included in the average time of sxd because it takes quite a long elapsed time. The performance of each command will be discussed in section 7.3.3.

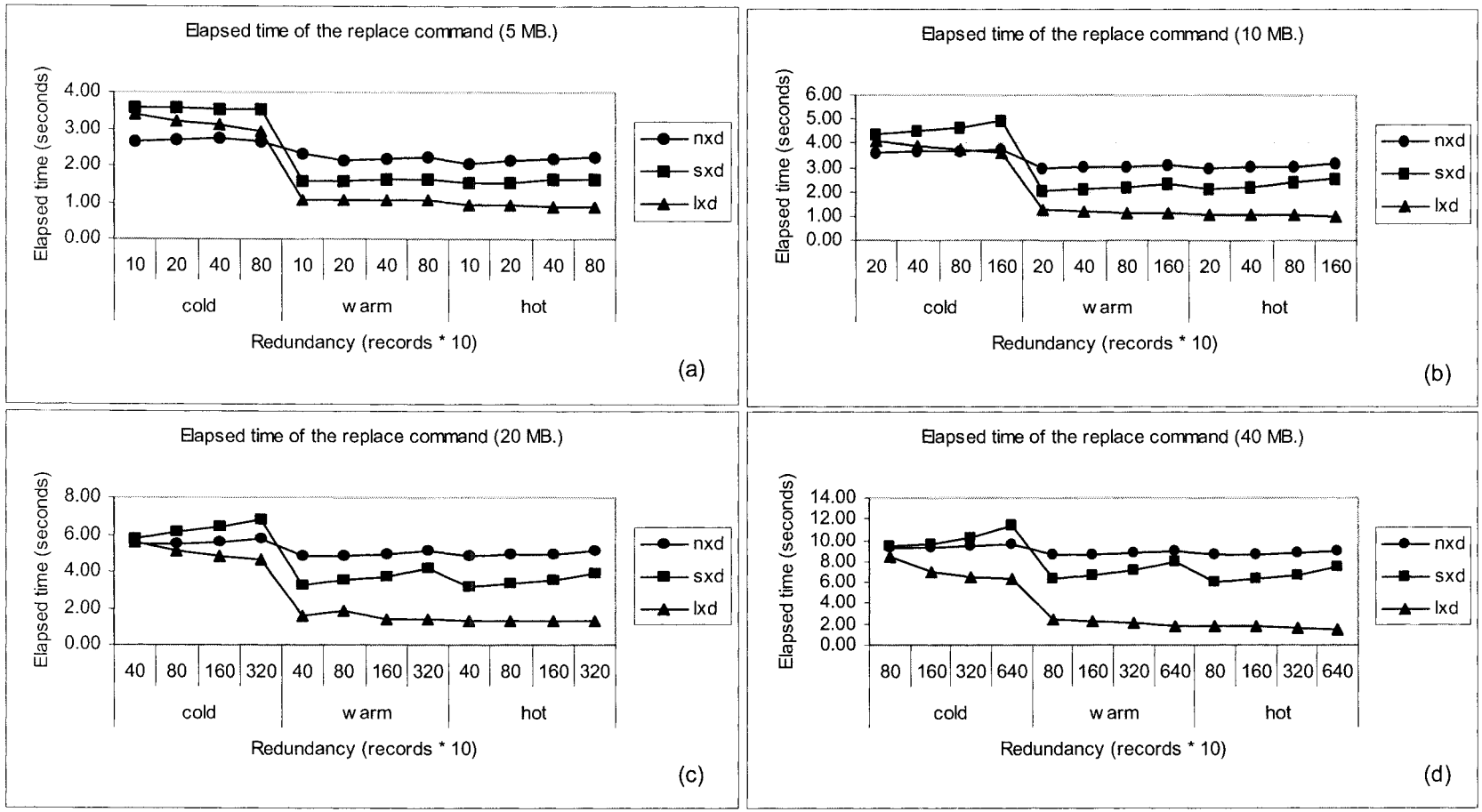

Figure 7.4 Replace time of nxd, sxd and lxd in cold, warm and hot caches

In a cold cache, sxd has the worst performance in every data size. nxd has the best performance when the data size is 5 and 10 MB whereas lxd has the best performance when data size is 20 and 40 MB. In warm and hot caches, nxd has the worst performance for every data size while lxd has the best performance for every data size.

From the graphs in Figure 7.4 in a cold cache for replace commands, sxd has the worst performance for every data size. nxd has the best performance when the data size is 5 and 10 MB whereas lxd has the best performance when data size is 20 and 40 MB. For nxd, XUpdate does not support replacing a complex element; therefore to replace one complex element, several update commands on each simple-element are used instead of one update command on the complex-element. Thus this makes the performance of nxd worse when the data size is bigger. On the other hand, in warm and hot caches, nxd has the worst performance for every data size while lxd has the best performance for every data size.

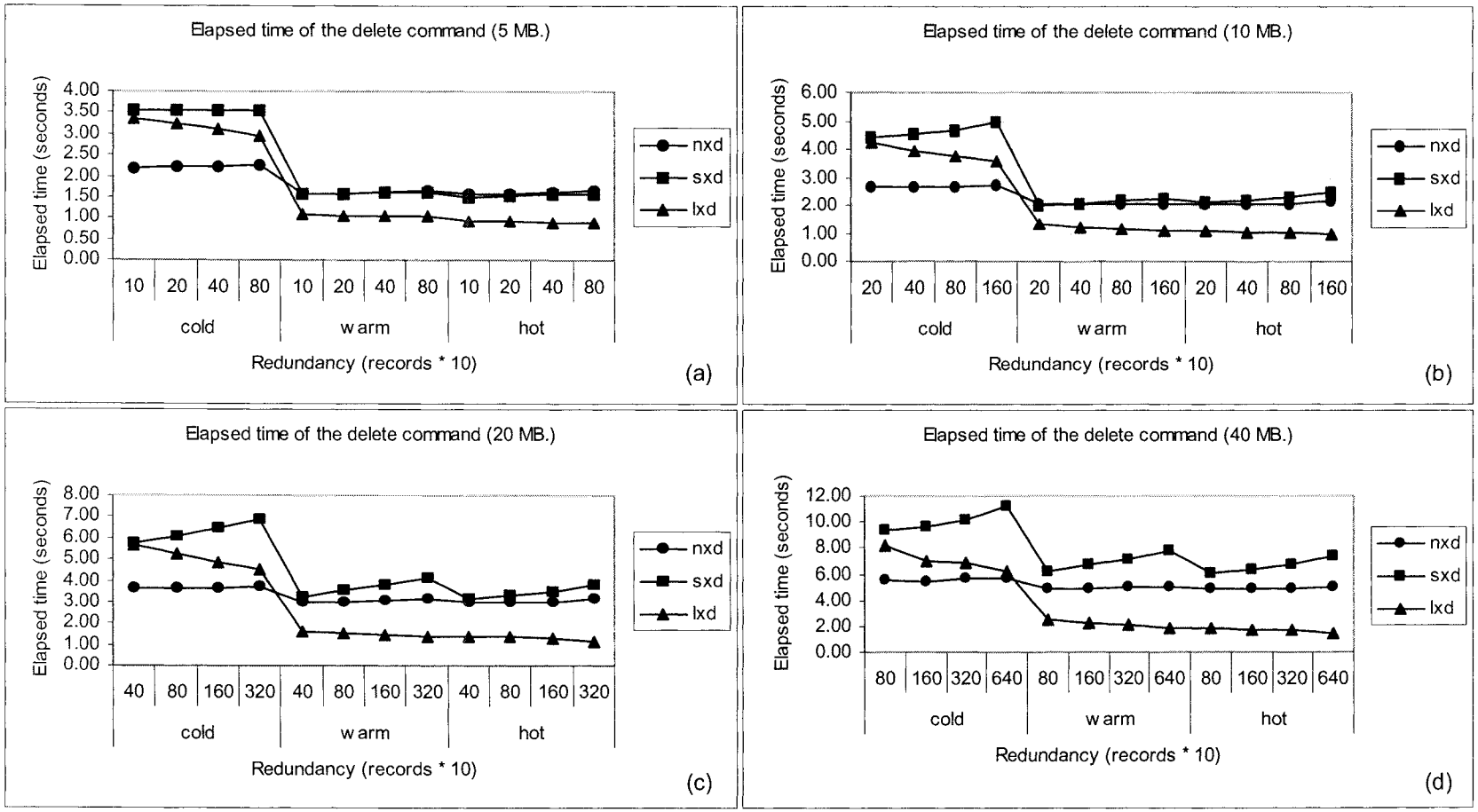

Figure 7.5 Delete time of nxd, sxd and lxd in cold, warm and hot caches.

In a cold cache, nxd has the best performance whereas sxd has the worst performance for every size of data. For warm cache and hot caches, lxd has the best performance whereas sxd still has the worst performance.

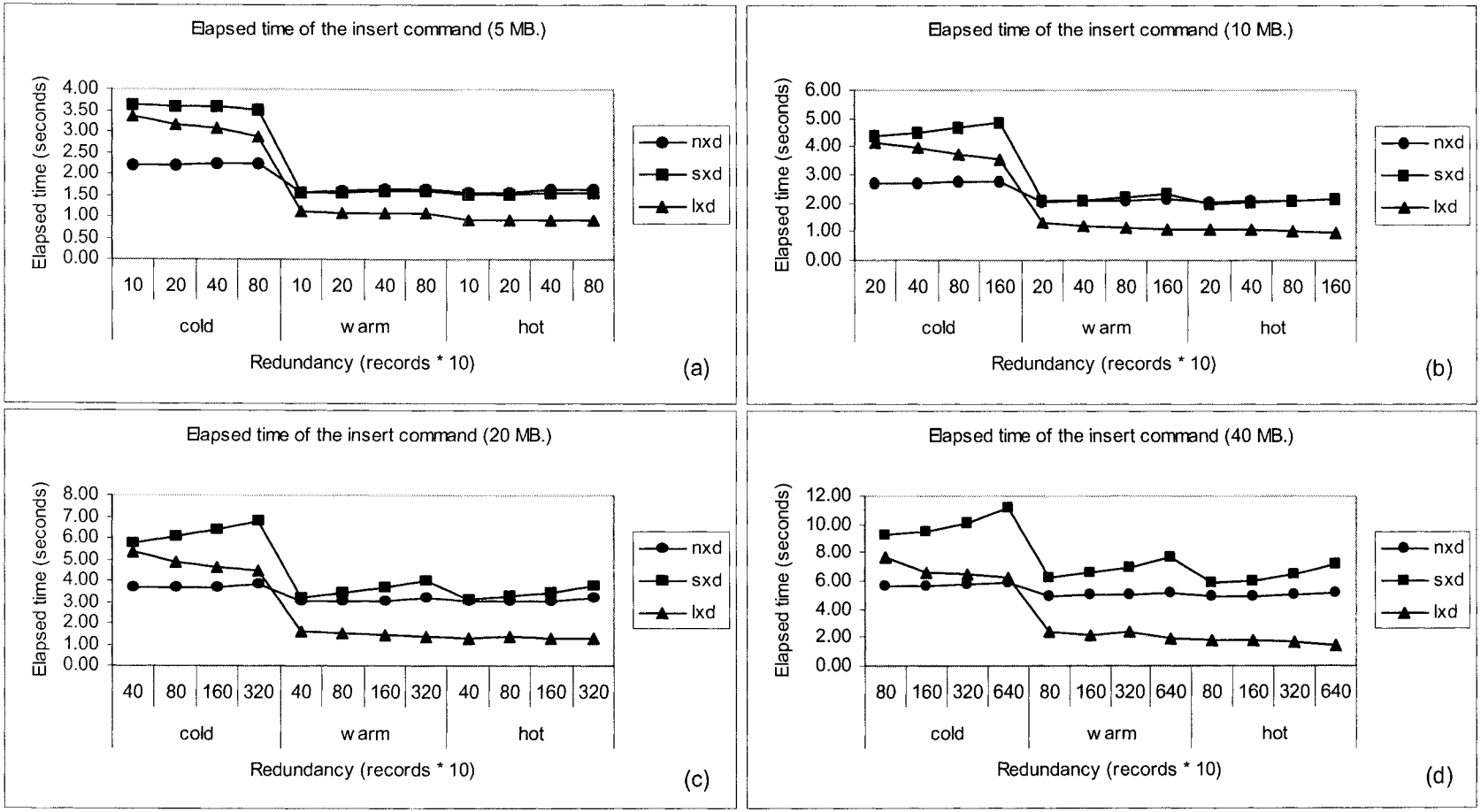

Figure 7.6 Insert time of nxd, sxd and lxd in cold, warm and hot caches.

In a cold cache, nxd has the best performance whereas sxd has the worst performance for every size of data. For warm cache and hot caches, lxd has the best performance whereas sxd still has the worst performance.

From the graphs in Figures 7.5 and 7.6 in a cold cache for delete and insert commands, nxd has the best performance whereas sxd has the worst performance for every size of data. For warm cache and hot caches, lxd outperformed nxd in all cases; however sxd has the worst performance in the case of insert and delete operations but in the case of a replace operation, sxd outperformed nxd. This is because XUpdate does not support the replacement of complex elements but it supports the insertion and deletion of complex elements; therefore inserting or deleting complex elements can be performed in one command; thereby nxd outperformed sxd in this case.

From the graphs in Figures 7.4-7.6, the time in a cold cache for lxd is about four or five times of that in a warm cache, and the time in a cold cache of sxd is double that in a warm cache. By contrast, the time in a cold cache and a warm cache for nxd is not much different. This shows that lxd and sxd benefit from caching data more than nxd. The difference in time between cold and warm caches of lxd is more than that for sxd. Although the majority of the records may be loaded already in a warm cache, sxd has more records that need to be updated.

Both sxd and lxd can exploit the advantage from caching data more than nxd when the ratio of elapsed time in three cache states is calculated. Thus it can be implied that a caching system for a conventional database such as ORDB is better than the caching system of a native XML database. Hence caching data has a greater effect on the performance of the traditional database than on the native XML database. In the case of data redundancy, the performance of sxd is affected by redundant data more than that of nxd as can be seen clearly when the data size is bigger than 5 MB. The lxd does not contain redundant data; thus the number of records that are updated will be constant in every size of redundancy. When the size of redundant data in nxd and sxd is bigger, the data size of lxd is smaller; therefore the performance of lxd is better.

With the limitation of resources, the biggest data size in the experiments is 40 MB. However, to show the difference of performance when the data size is quite large, the extrapolation of data size ranging from 160, 320, 640 to 1280 MB is calculated and shown in the graphs in Figure 7.7-7.9.

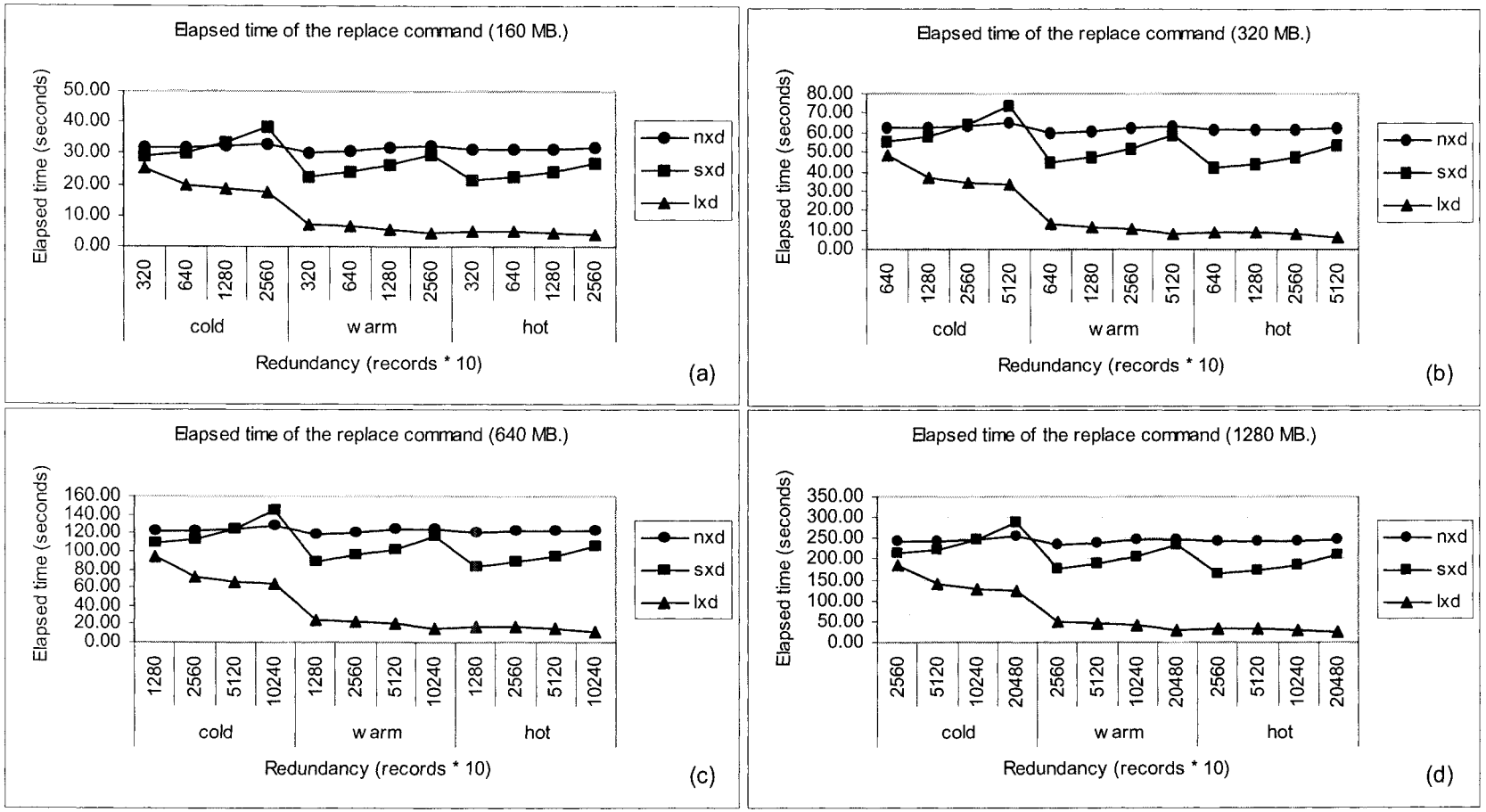

Figure 7.7 Extrapolated replace time of nxd, sxd and lxd in cold, warm and hot caches

When data size is large and there is not much data redundancy, sxd outperformed nxd in a cold cache. The other conclusions about these graphs are the same as the ones for the graphs in Figure 7.4.

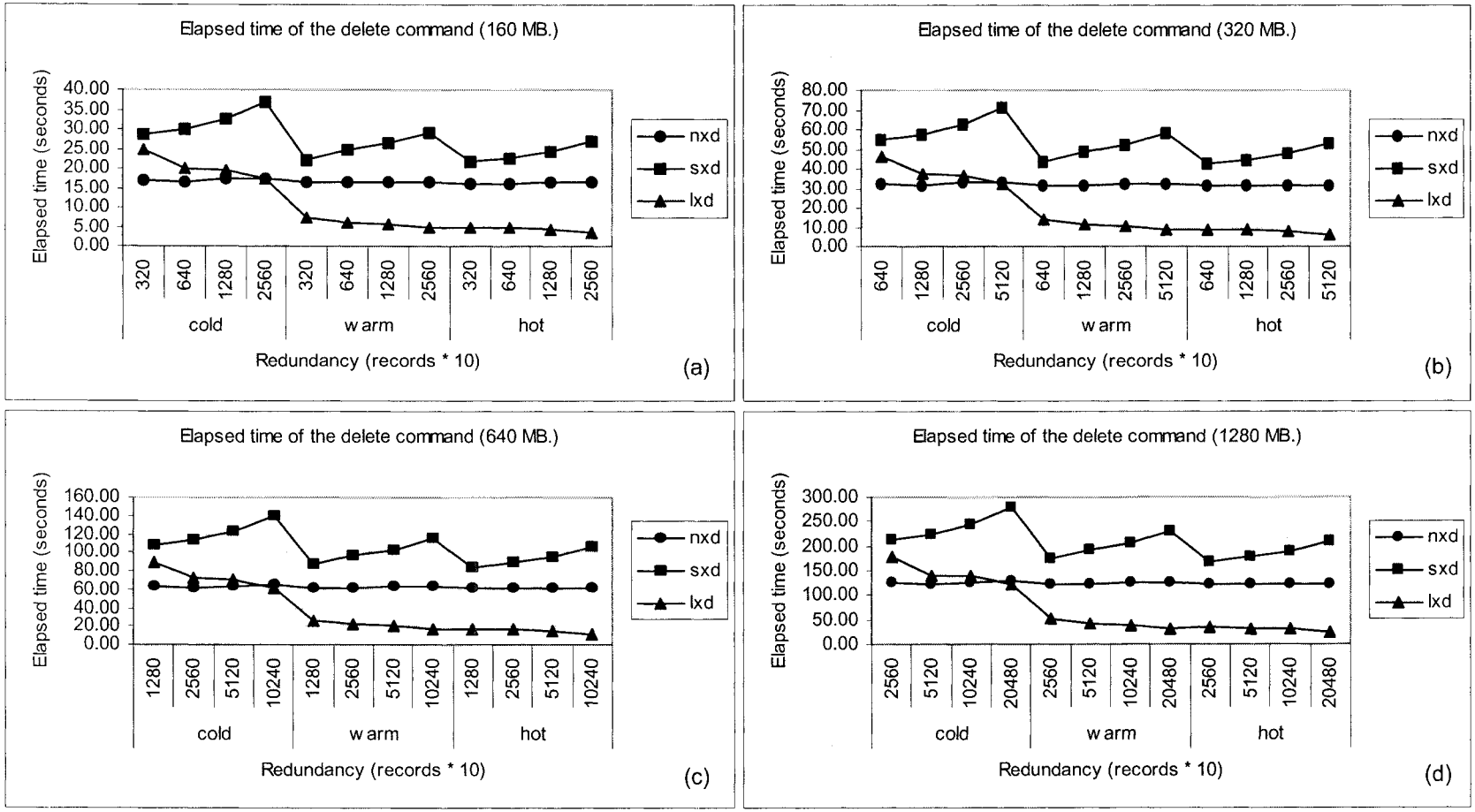

Figure 7.8 Extrapolated delete time of nxd, sxd and lxd in cold, warm and hot caches

When data size is large and there is much data redundancy, lxd outperformed nxd in cold cache. The other conclusions for these graphs are the same as the ones for graphs in Figures 7.5.

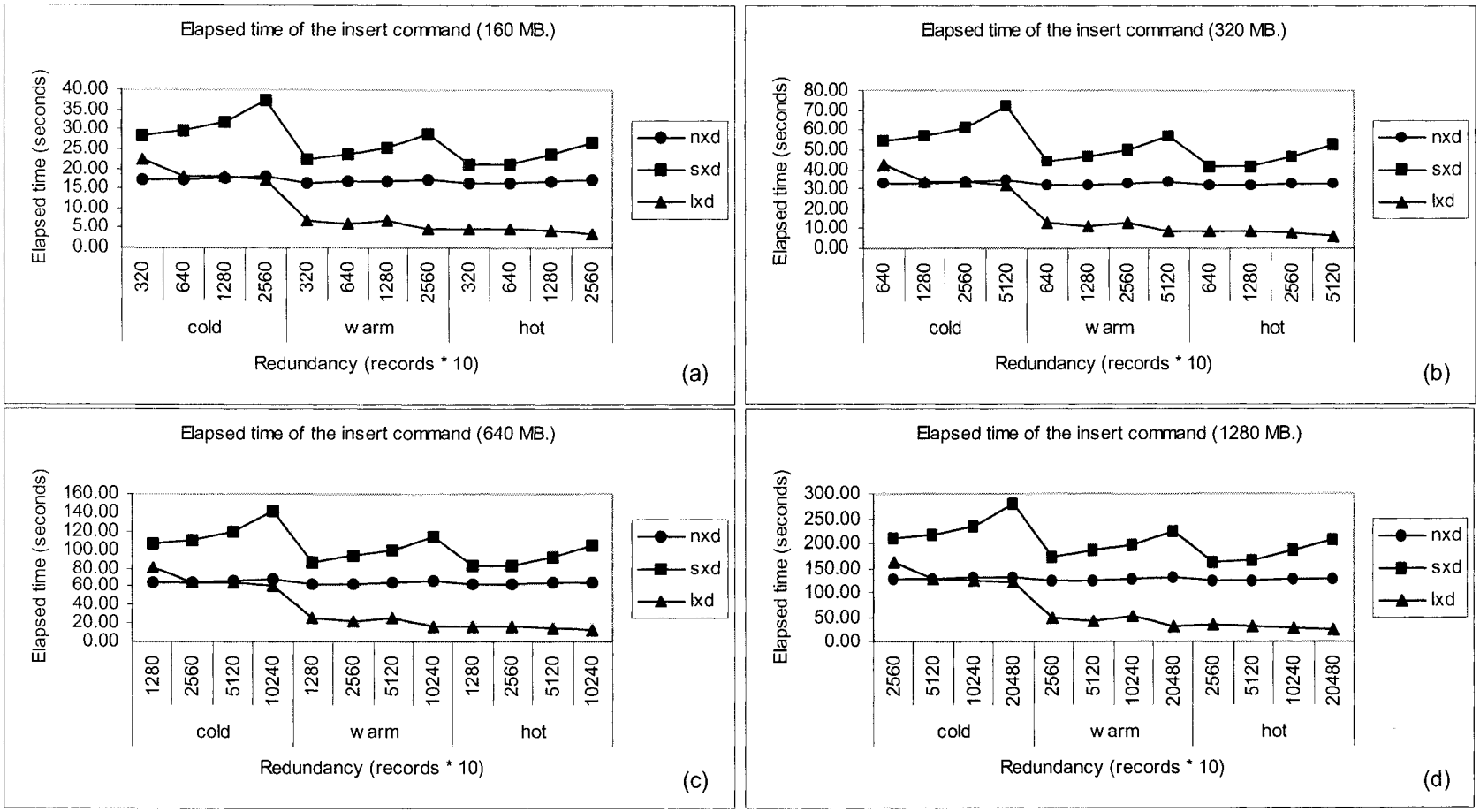

Figure 7.9 Extrapolated insert time of nxd, sxd and lxd in cold, warm and hot caches

When data size is large and there is much data redundancy, lxd outperformed nxd in cold cache. The other conclusions for these graphs are the same as the ones for graphs in Figures 7.6

From the graphs in Figure 7.7 for the replace operation, it can be seen that when the data size is large and there is not much data redundancy (the first three orders of data redundancy), sxd outperformed nxd in a cold cache. The other conclusions about these graphs are the same as the ones for the graphs in Figure 7.4.

From the graphs in Figures 7.8-7.9 for delete and insert operations, it can be seen that when data size is large and there is much data redundancy, lxd outperformed nxd in cold cache. The other conclusions for these graphs are the same as the ones for graphs in Figures 7.5-7.6.

To conclude, in a cold cache, nxd has the best performance in most cases whereas lxd has the best performance in every case for warm and hot caches. However for large data volumes and quite a lot of data redundancy, lxd outperformed nxd in a cold cache. In the real world users work in warm or hot cache states more than in a cold cache state since the cold cache state occurs only once when the first query or update command is run. After that the cache will get warmer until it is hot.

7.3.2 Performance comparison of three update operations

For the third purpose of our experimental study, another twelve graphs are produced to compare the performance of three update operations, namely replace, delete and insert operations for each XML database. The twelve graphs are presented as four graphs in Figure 7.10 for a cold cache, four graphs in Figure 7.11 for a warm cache and four graphs in Figure 7.12 for a hot cache. Each graph compares a high-level view of performance in terms of the average elapsed time of all update commands for each type of update operations performed on three XML databases for a particular data size ranging from 5, 10, 20 to 40 MB in four different sizes of redundant data in a particular cache state. As in section 7.3.1, the commands C14 and C17 are excluded from the average time of nxd. The command C3 is not included in the average time of sxd.

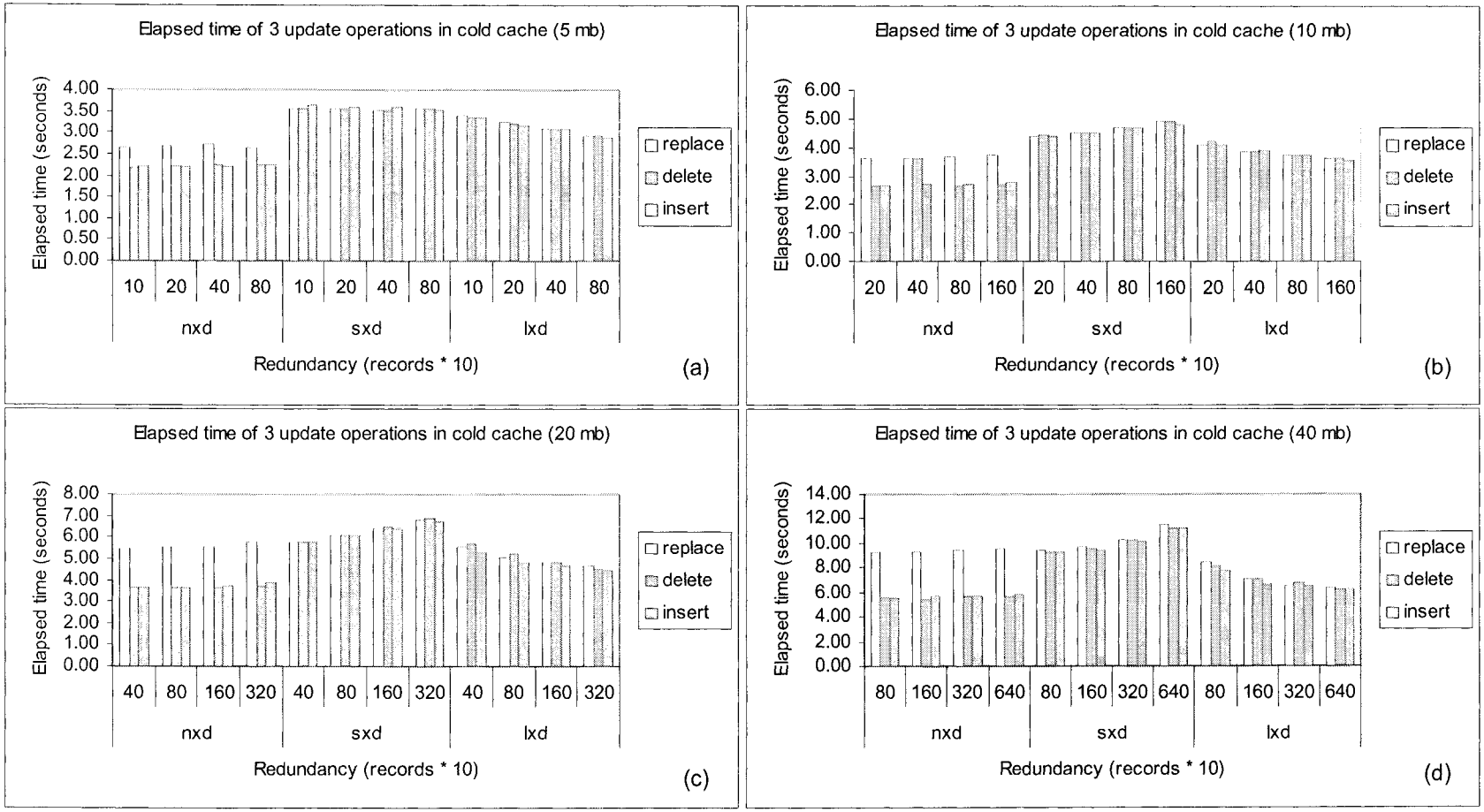

Figure 7.10 Elapsed time of three update operators on nxd, sxd and lxd in a cold cache

For nxd, the delete operation takes the shortest elapsed time while the replace operation takes the longest elapsed time. For sxd and lxd, the insert operation takes the shortest elapsed time whereas it is uncertain whether the replace operation or delete operation takes the longer elapsed time.

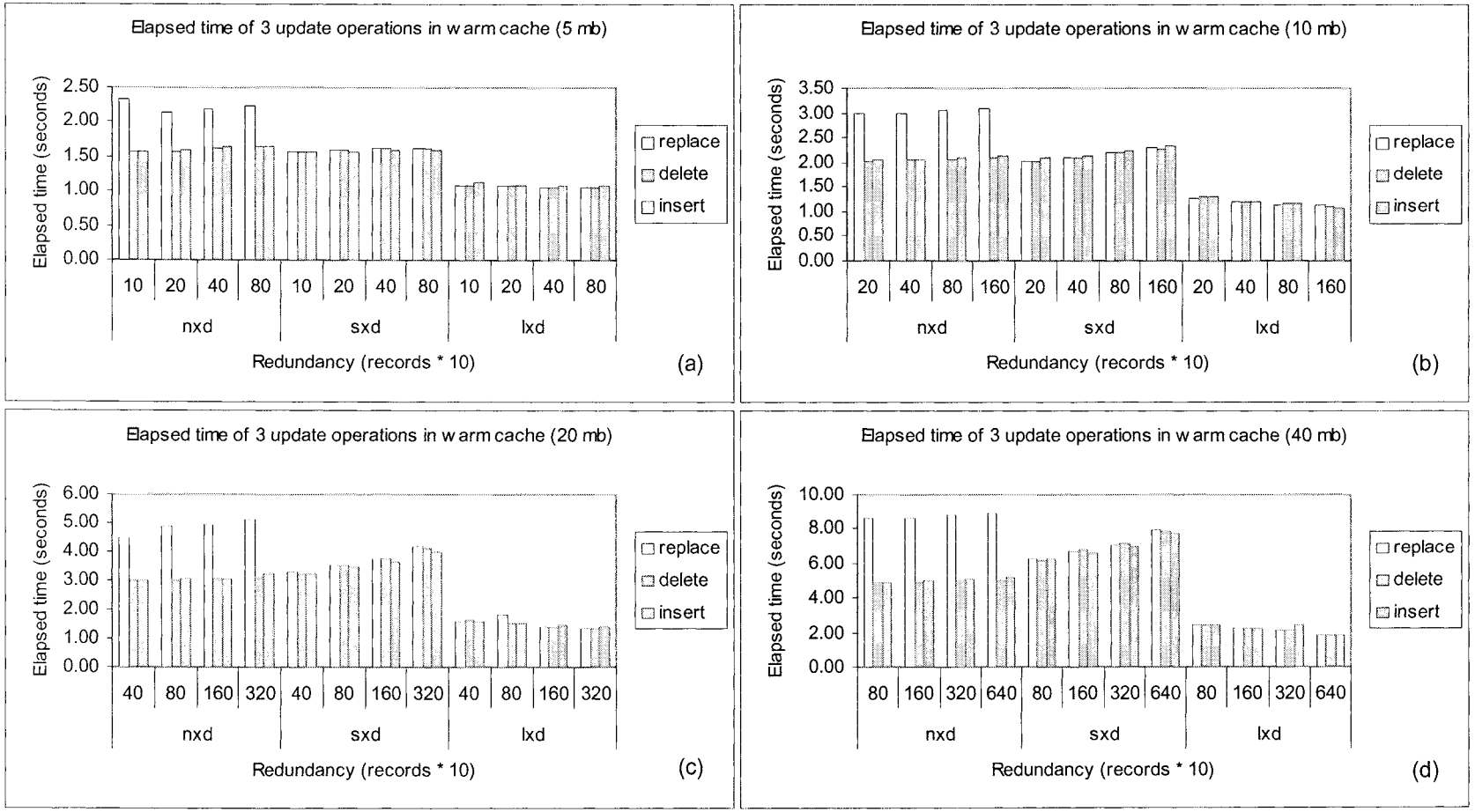

Figure 7.11 Elapsed time of three update operators on nxd, sxd and lxd in a warm cache

For nxd, the delete operation takes the shortest elapsed time while the replace operation takes the longest elapsed time. For sxd and lxd, the insert operation takes the shortest elapsed time whereas it is uncertain whether the replace operation or delete operation takes the longer elapsed time.

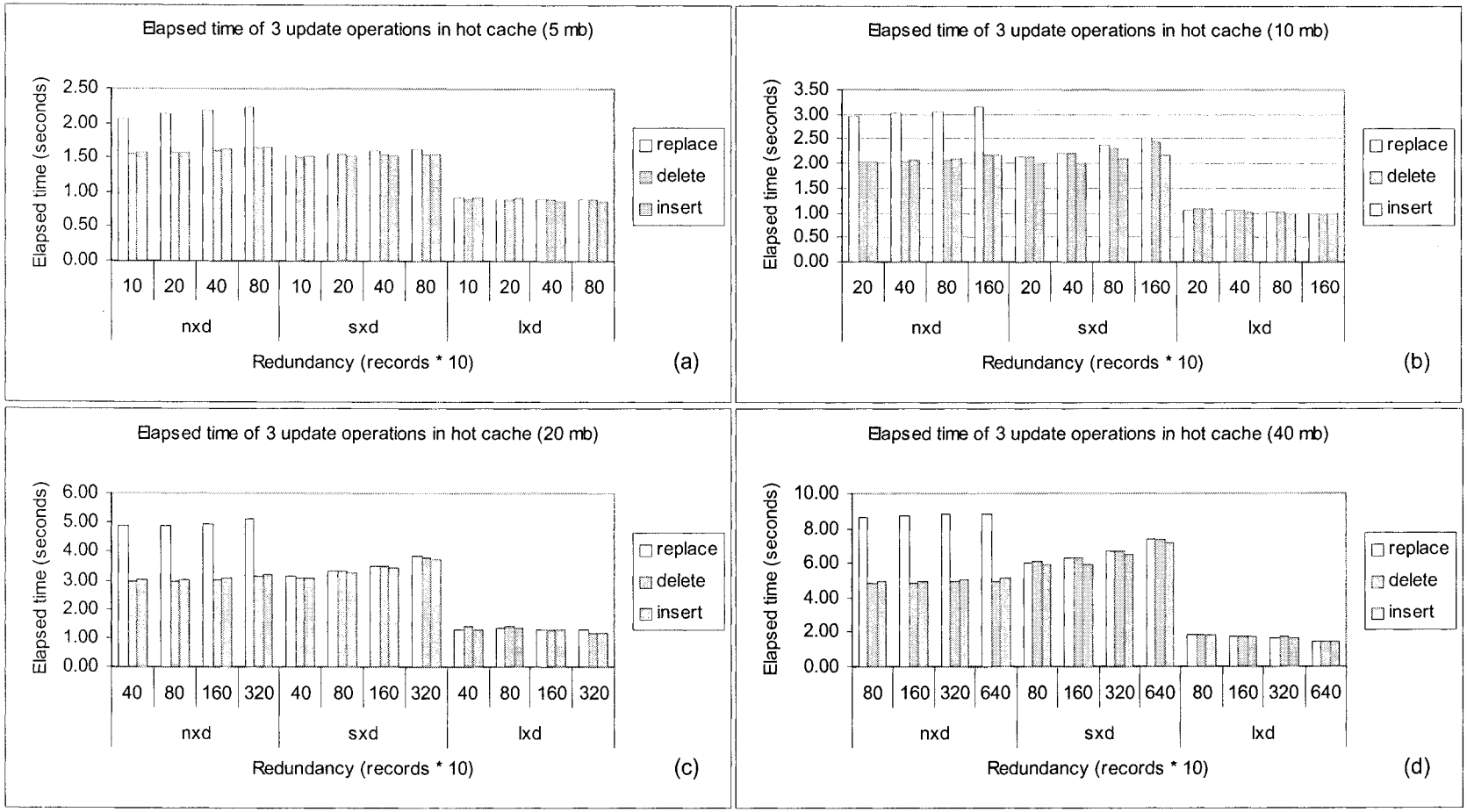

Figure 7.12 Elapsed time of three update operators on nxd, sxd and lxd in a hot cache

For nxd, the delete operation takes the shortest elapsed time while the replace operation takes the longest elapsed time. For sxd and lxd, the insert operation takes the shortest elapsed time whereas it is uncertain whether the replace operation or delete operation takes the longer elapsed time.

From the twelve graphs in Figure 7.10-7.12, in the case of nxd, it is obvious that the replace operation takes the longest elapsed time for every data size and every cache state when compared with the insert and delete operations. This is because the model of the native XML database is a tree model; thus the replace operation in the tree model is a sequence of delete and insert operations; hence its elapsed time is the time for deletion plus the time for insertion. For the case of insert and delete operations in nxd, it is not clear which operation takes the longer elapsed time when the data has a small size (5 MB). For the data size bigger than 5 MB, the insert operation takes a slightly longer elapsed time than that of the delete operation. This is because inserting data to the tree model must add new nodes to the tree while deleting data from the tree model just detaches the existing nodes and it is not necessary to actually delete data immediately when performing a delete operation.

In contrast to nxd, with most cases of sxd and lxd the insert operation has the shortest elapsed time when compared to the replace and delete operations. This is because inserting data in an ORDB just locates the position of the insertion and inserts the data to a continuous area. On the other hand, when replacing or deleting data in an ORDB, not only the target position is located but the existing data is modified as well. Modifying the existing data requires an ORDBMS to preserve the previous version of the data for rollback purposes; moreover, the existing data that will be replaced or deleted may not be in a continuous area; thus it has to jump to several locations to update the data.

In the case of replace and delete operations, it is difficult to determine which operation takes the longer elapsed time since in some case, the replace operation takes a longer time and in some cases, it takes a shorter time; moreover, there is not much performance difference between replace and delete operations.

To conclude, in the case of nxd, the delete operation takes the shortest elapsed time while the replace operation takes the longest elapsed time. For sxd and lxd, the insert operation takes the shortest elapsed time whereas it is uncertain whether the replace operation or delete operation takes the longer elapsed time.

7.3.3 Performance comparison of seventeen update-features

This section serves the fourth purpose of the experimental study. A total of 36 tables are produced. These tables consist of 12 tables for a cold cache, 12 tables for a warm cache and 12 tables for a hot cache with each 12 cache state table in turn comprising 4 tables for the replace operation, 4 tables for the delete operation and 4 tables for the insert operation. Each table shows the elapsed time of the update commands covering 17 features that are performed on three types of XML databases: nxd, sxd and lxd, with a particular data size and four different sizes of redundant data.

However, we present here only some important data which can be used as representative for the performance of the update-features. The representative data consists of insert and delete commands performed in a hot cache with data size 40 MB which is the biggest size of the test data. Only 40 MB data size with 800 redundant records is presented here since the difference of the performance between the update-features is similar for every size of redundant data. In reality both cold and hot caches can be used to capture actual performance since in a cold cache all required data is in the disk whereas in a hot cache all required data is in the memory. However in the real world, applications are run in a warm or hot cache more than in a cold cache; thus we choose the hot cache to show performance of each update feature. The replace commands are not presented here since nxd does not support replacing a complex element; therefore several update commands on each simple-element are executed instead of one update command on the complex-element; thereby this makes it difficult to determine the actual performance difference.

The representative data is summarized into two tables as shown in Tables 7.1 and 7.2. The first table, Table 7.1, shows the performance difference between the update-features of insert commands. The commands are ordered according to the elapsed time in ascending order. The second table, Table 7.2, compares the performance between insert and delete commands. All results for the 36 tables are presented in Appendix D.

Note: The update-time in the graphs and the tables excludes serialisation-time since, usually, serialisation can be performed only once after all updates finish. However, the serialisation-time is shown in the detail-tables in Appendix D. The detail-tables are discussed in section 7.3.5.

The 17 update-features tested in the experimental study are shown in Figure 7.3. As mentioned earlier in section 7.2.3, in the experiments, feature C11 will appear along with feature C17 in the command C17 while the feature C2 is a basic feature; thus C2 will appear in several commands updating a particular document without a join to other documents.

Table 7.1 Insert time of three databases in hot cache (40 MB, 800 redundant records)

nxd	
C17	-
C5	3.75
C15	3.94
C4	4.19
C10	4.32
C3	4.33
C1	4.57
C12	4.65
C8	4.72
C131	4.75
C132	4.75
C7	4.85
C9	4.94
C6	5.62
C16	9.37
C14	85.32
AVG	**4.91**

(a)

sxd	
C12	4.95
C15	5.14
C4	5.15
C9	5.17
C6	5.19
C14	5.21
C132	5.32
C131	5.34
C8	5.42
C7	5.58
C10	5.59
C5	5.72
C3	5.84
C1	5.93
C16	9.26
C17	9.74
AVG	**5.91**

(b)

lxd	
C12	1.38
C17	1.41
C15	1.44
C4	1.47
C14	1.48
C9	1.51
C8	1.76
C1	1.81
C3	1.82
C6	1.82
C132	1.91
C131	1.92
C10	2.03
C7	2.06
C5	2.26
C16	3.34
AVG	**1.84**

(c)

From Table 7.1 (a), the command C14 testing the *regular path expression (//)* produces an unacceptable performance. This indicates that nxd has a weak point in handling the *regular path expression* because all possible paths in XML documents must be navigated. On the other hand from the Tables 7.1 (b) and (c), sxd and lxd can handle this type of command quite well since the XML update language is translated into SQL; thus the real path expression is not genuinely involved in executing the update command. Additionally the path in the regular path expression can be determined by using mapping information; thus fields or tables involved in the expression are resolved in a short time.

Command C16 (*hierarchical and sequence update*) comprises a sequence of update commands; thus its elapsed time is longer than for one single update command. If C14 and C16 are not taken into account, the command C6 testing a quantifier takes the longest elapsed time because not only is the tree searched to match the condition but

also the grouping of data is calculated. The elapsed times of the rest of the commands are close to each other. In this group, the command C9 takes the longest elapsed time since the content of XML is text; thus it takes time to cast the text type to a numeric type to calculate the data.

For sxd from Table 7.1(b), the command C17 testing the *recursion* takes the longest elapsed time because in translating the XML update language, we repeat the deletion and insertion of data from and into the temp table. The elapsed time of the command C16 is the second longest since C16 performs a sequence of update operations; thus more than one subcommand is executed.

The elapsed times of the rest of the commands are close to each other. C5 takes a longer time than C6 since in our language translation, a quantifier is translated into the count() function along with the conditions of the quantifier; thus before grouping and counting the data in each group, the conditions can eliminate unwanted data to decrease the size of data. The command C8 testing the *joins based on pointers* takes a shorter time than C7 since C8 uses a foreign key and a primary key to join data.

For lxd from Table 7.1(c), the order of the commands with respect to elapsed time is different from sxd since the number and the structure of tables in sxd and lxd are not the same. The lxd has more layers of objects inside tables than sxd.

The command C16 takes the longest elapsed time because C16 is a sequence of update operations. The command C5 takes the second longest elapsed time because C5 is related to three tables and it has to both group and count data. The command C7 takes the third longest elapsed time. Although C7 operates on only two tables it does not use a key for join of data. The command C10 takes the fourth longest elapsed time because it deals with four tables. The elapsed times of the rest of the commands are not much different. Similar to sxd, C7 takes a longer time than C8, and C5 takes a longer time than C6.

Table 7.2 Comparison of insert time and delete time

Cmd.	nxd		sxd		lxd	
	Insert	Delete	Insert	Delete	Insert	Delete
C1	4.57	4.53	5.93	5.91	1.81	1.84
C3	4.33	4.25	5.84	12.38	1.82	2.00
C4	4.19	4.07	5.15	5.19	1.47	1.44
C5	3.75	3.69	5.72	5.74	2.26	2.21
C6	5.62	5.53	5.19	5.20	1.82	1.81
C7	4.85	4.77	5.58	5.69	2.06	1.97
C8	4.72	4.68	5.42	5.59	1.76	1.74
C9	4.94	4.86	5.17	5.20	1.51	1.46
C10	4.32	4.26	5.59	5.59	2.03	2.03
C12	4.65	4.63	4.95	5.16	1.38	1.41
C131	4.75	-	5.34	-	1.92	-
C132	4.75	-	5.32	-	1.91	-
C14	85.32	85.44	5.21	5.21	1.48	1.42
C15	3.94	3.83	5.14	5.16	1.44	1.39
C16	9.37	9.28	9.26	9.47	3.34	3.36
C17	-	-	9.74	9.75	1.41	1.41
AVG	**4.91**	**4.86**	**5.91**	**6.07**	**1.84**	**1.82**
STDEV	1.36	1.84	1.43	**1.59**	0.48	**0.53**
%RSD	**27.78**	**30.34**	**24.22**	**26.19**	**26.20**	**29.12**

(a) (b) (c)

From Table 7.2, there is not much difference between the performance of insert and delete commands except for C3 of sxd in Table 7.2(b). For sxd, the delete command C3 testing the *change selectivity* takes much more time than the insert command C3. This is because two conditions: '>=' and '<=' are performed by scanning a full-table. The deletion of the data requires the DBMS to preserve the previous version of the data for rollback purposes; thus it is possible that this operation is performed every time when the scan-operation finds each target record. For lxd, there is no redundant record; thus the number of the operations is much fewer than for sxd.

From the value of the standard deviation, the performance difference between most update-features for nxd, sxd and lxd is not much different. However lxd can handle every feature of commands better than sxd and nxd, and sxd can handle all features better than nxd even though C3 in the case of deletion of sxd takes quite a long time; however the command C14 of nxd takes much more time than the command C3 of sxd.

To conclude, a weak point of nxd is handling the *regular path expression* while sxd and lxd can manage this well. However the capability for handling the *change selectivity* of

sxd in the case of deletion is not adequate but in the case of updating non-redundant data, lxd is satisfactory in handling this feature.

7.3.4 Checking the correctness of XML update language translation

Besides the commands designed to correspond to the features of the XML update language which have been used in the experiments earlier, an additional two sets of 15 update commands for two ORDBs are designed to serve the last purpose of the experimental study, checking the correctness of the translation. Updating an ORDB containing XML data is related not only to the features of the XML update language but also to the schema of the ORDB since the update language is translated into SQL executed on the ORDB. Thus to check the correctness of translating the XML update language, both update features and types of the ORDB structure should be taken into account in translation. Thus the additional update commands are designed according to the updating of data in different ORDB structures, namely table, nested table and abstract data type field. Here, we show only SQL derived from translating the replace commands according to update features for sxd in Table 7.3. However the translation of delete and insert commands is shown in Appendix D as well as the translation of update commands according to update features for lxd and the translation of the additional update commands according to updating data in the different ORDB structure.

To conclude, the prototype can translate the XML update language correctly since the results from translation are as expected. The results also reinforce the validation of the logical statements expressed in Chapter 5.

Table 7.3: Translating XML update commands into SQL for replacing in sxd

Feature	XML update command	SQL in the form of PL/SQL
C1	For $p in doc("Library.xml")/Library/Publication, $i in $p/Publisher/Address, $n in $p/Author/Name Where $i/Country = "Thailand" Replace $n/MName with <MName>Anantasamakom</MName>	update Author A set A.Name.MName = 'Anantasamakom' where A.AuthorID in (select A.AuthorID from Author A, Publisher P, Library L where P.Address.Country = 'Thailand' and A.PubID = L.Publication.PubID and P.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;
C3	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author Replace $a/Telephone with <Telephone> <Location>Office</Location> <TelNo>9999999</TelNo> </Telephone> where $p/Year >= "2000" and $p/Year <= "2001"	select distinct A.AuthorID bulk collect into l_array from Author A, Library L where L.Publication.Year >= '2000' and L.Publication.Year <= '2001' and A.PubID = L.Publication.PubID; forall i in l_array.first..l_array.last update table(select A.Telephone from Author A where A.AuthorID in l_array(i)) set Location = 'Office', TelNo = '9999999' ;
C4	For $i in doc("Library.xml")/Library/Publication/Publisher where contains($i/PName, "Sams Publishing") and contains($i/PName, "LTD.") Replace $i/Address with <Address> <No>22</No> <Street>Times</Street> <City>Ohio</City> <Country>USA</Country> <ZipCode>Ne5 ST7</ZipCode> </Address>	update Publisher P set P.Address.No = '22', P.Address.Street = 'Times', P.Address.City = 'Ohio', P.Address.Country = 'USA', P.Address.ZipCode = 'Ne5 ST7' where P.PName like '%Sams Publishing%' and P.PName like '%LTD.%' returning P.PID bulk collect into l_array;
C5	For $p in doc("Library.xml")/Library/Publication Let $a := $p/Author Where count($a) >= 2 Replace $p@PubType with @PubType = "book"	update Library L set L.Publication.PubType = 'book' where L.Publication.PubID in (select L.Publication.PubID from Library L, Author A where A.PubID = L.Publication.PubID group by L.Publication.PubID having count(*) >= 2) returning L.Publication.PubID bulk collect into l_array;
C6	For $p in doc("Library.xml")/Library/Publication Where some $a in $p/Author satisfies ($a/Name/LName = "Dobson") Replace $p@PubType with @PubType = "book"	update Library L set L.Publication.PubType = 'book' where L.Publication.PubID in (select L.Publication.PubID from Library L, Author A where A.Name.LName = 'Dobson' and A.PubID = L.Publication.PubID group by L.Publication.PubID having count(*) > 0) returning L.Publication.PubID bulk collect into l_array;

C7	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author, $i in $p/Publisher Where $p/Year = $i/SetupYear Replace $i/Address with <Address> <No>77</No> <Street>Avenue 5</Street> <City>New York</City> <Country>USA</Country> <ZipCode>Nr4 587</ZipCode> </Address>	update Publisher P set P.Address.No = '77', P.Address.Street = 'Avenue 5', P.Address.City = 'New York', P.Address.Country = 'USA', P.Address.ZipCode = 'Nr4 587' where P.PID in (select P.PID from Publisher P, Library L where L.Publication.Year = P.SetupYear and P.PubID = L.Publication.PubID) returning P.PID bulk collect into l_array;
C8	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author Where $p@ContactAuthor = $a@AuthorID Replace $a/Email with <Email>contact@hotmail.com</Email>	update Author A set A.Email = 'contact@hotmail.com' where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.ContactAuthor = A.AuthorID and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;
C9	For $p in doc("Library.xml")/Library/Publication Where $p/Cost = $p/SpecialCost * 2 Replace $p/ShippingCost with <ShippingCost>50</ShippingCost>	update Library L set L.Publication.ShippingCost = 50 where L.Publication.Cost = L.Publication.SpecialCost *2 returning L.Publication.PubID bulk collect into l_array;
C10	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author, $i in $p/Publisher Where $i/SetupYear >= "2004" and $p/Year = "2005" Replace $a/Telephone with <Telephone> <Location>School</Location> <TelNo>01912739145</TelNo> </Telephone>	select distinct A.AuthorID bulk collect into l_array from Author A, Publisher P, Library L where P.SetupYear >= '2004' and L.Publication.Year = '2005' and A.PubID = L.Publication.PubID and P.PubID = L.Publication.PubID; forall i in l_array.first..l_array.last update table(select A.Telephone from Author A where A.AuthorID in l_array(i)) set Location = 'School', TelNo = '01912739145' ;
C12	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author where empty($a/Name/MName) Replace $a/Email with <Email>sams@hotmail.com</Email>	update Author A set A.Email = 'sams@hotmail.com' where A.Name.MName is NULL returning A.AuthorID bulk collect into l_array;
C14	For $n in doc("Library.xml")//Name replace $n/MName with <MName>Anonymous</MName>	update Author A set A.Name.MName = 'Anonymous' returning A.AuthorID bulk collect into l_array;

C15	For $p in doc("Library.xml")/Library/Publication, $i in $p/Publisher where contains($i/PName, "Sams") and $p/Year >= "2005" Replace $i/Address with <Address> <No>22</No> <Street>Times</Street> <City>Ohio</City> <Country>USA</Country> <ZipCode>Ne5 ST7</ZipCode> </Address>	update Publisher P set P.Address.No = '22', P.Address.Street = 'Times', P.Address.City = 'Ohio', P.Address.Country = 'USA', P.Address.ZipCode = 'Ne5 ST7' where P.PID in (select P.PID from Publisher P, Library L where P.PName like '%Sams%' and L.Publication.Year >= '2005' and P.PubID = L.Publication.PubID) returning P.PID bulk collect into l_array;
C16	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author, $n in $a/Name Where $p/Cost >= "1500" replace $a/Email with <Email>sams2@hotmail.com</Email> , replace $n/MName with <MName>Anonymous2</MName>	update Author A set A.Name.MName = 'Anonymous2' where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.Cost >= '1500' and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array; update Author A set A.Email = 'sams2@hotmail.com' where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.Cost >= '1500' and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;

C17 & C11	For $p in doc("Library.xml")/Library/Publication
	Where $p/Title = "XML"
	allRef($p)
	
	define function allRef($pub as element()*)
	{
	For $rp in $pub/Reference/@RefPub→
	Publication
	Replace $rp@PubType with
	@PubType = "journal"
	allRef($rp)
	}
		delete from Array;
		insert into Array
		select L.Publication.PubID from Library L
		where L.Publication.Title = 'XML';
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R,
		Library L, Reference R2
		where R2.PubID = L.Publication.PubID
		and R.RefID = R2.RefID
		and L.Publication.PubID in (select * from ProcessingArr);
		insert into Collector select * from Array;
		update Library L
		set L.Publication.PubType = 'journal'
		where L.Publication.PubID in
		(select * from Array);
		Loop
		If SQL%RowCount > 0 then
		delete from ProcessingArr;
		insert into ProcessingArr
		(select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R,
		Library L, Reference R2
		where R2.PubID = L.Publication.PubID
		and R.RefID = R2.RefID
		and L.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Library L
		set L.Publication.PubType = 'journal'
		where L.Publication.PubID in
		(select * from Array);
		Else Exit;
		End If;
		End Loop;
		delete from Collector
		returning id bulk collect into l_array;

7.3.5 Detail-tables

Each change to an XML document consists of multiple operations. To show more detail, detail-tables have been produced to show the ratios of elapsed time by independent operations in sxd, lxd and nxd. The detail-tables in Appendix D present details of the time used in each operation of each command. The operations for updating a native XML database (nxd) consist of updating data in database and serialising from the database to the XML document; thus the update time for nxd consists of update-time of the database and serialisation-time to XML document. The

operations for updating a single XML document database (sxd) and a linked XML document database (lxd) consist of translating the XML update language into SQL, updating data in an ORDB, updating a document object model (DOM) of XML and serialising from the DOM to the XML documents; hence the update time for sxd and lxd consists of the translate-time of the language, the update-time of an ORDB, the update-time of an DOM and the serialisation-time to XML documents.

The detail-tables show that translation time is very small while most of the update time is used in serialising data to XML documents; nevertheless, serialisation is not usually performed for each individual update command. When comparing serialisation-time among three databases, nxd's time is 2-3 times sxd's time and 6-10 times lxd's time. When the data size is increased, the difference of serialisation-time among these databases is increased as well since serialisation of nxd must read data from the database kept on a disk while serialisation of sxd and lxd reads data from main memory. However the advantage of nxd is that data kept on the disk is permanent; thus serialisation can be performed only when necessary to use XML documents. On the other hand, the disadvantage of sxd and lxd is that serialisation must be performed every time after all the updating has finished; however the data is always up-to-date and the change to XML documents can be reflected immediately.

In the case of SQL (update-time in an ORDB) and update-time of DOM for sxd in warm and hot caches, the size of data affects the update-time of the DOM more than SQL. When the data size is small, the SQL is bigger than the update-time of the DOM but when the data size is big the result is the opposite.

7.4 Summary of the experimental study

The results from the experimental study can be summarised as follows.

- In a cold cache, nxd has the best performance except for the case of the replace operation; lxd is better than nxd when the data size is bigger than 10 MB.
- lxd always seems superior to nxd for warm and hot caches because an ORDB can take greater advantage of data caching than a native XML database. sxd has the worst performance in every case except for the replace operation where sxd is better than nxd. This is because XUpdate used in nxd does not support replacing a

complex element; thus this replacement is translated into the replacing of several simple elements.

- The time in a cold cache for lxd is about four or five times of that in a warm cache while the time in a cold cache of sxd is about double that in a warm cache. However the time in a cold cache and a warm cache for nxd is not much different. This may be because a relational engine has been developed for more than 25 years; thus it provides a caching system that is much more efficient than a native XML database.
- The different performance among the three update operations in nxd is more obvious than that in sxd and lxd.

  In the case of nxd, the delete operation takes the shortest elapsed time while the replace operation takes the longest elapsed time. This is because deletion of data just detaches the existing nodes from the tree while replacement is a sequence of deletion and insertion. For sxd and lxd, the insert operation takes the shortest elapsed time whereas we are unsure whether the replace or delete operation takes a longer time. This is because in inserting data into an ORDB, only the insertion of the data is performed. For replacing or deleting data, it requires the ORDBMS to preserve the previous version of the data for rollback purposes before modifying the data.
- nxd has a weak point in handling the *regular path expression* because all possible paths in XML documents must be navigated. Handling the *change selectivity* of sxd in the case of deletion is not good enough since testing two conditions: '>=' and '<=' is performed by scanning a full-table. It is possible that the preservation of the previous version of the data may be performed every time that the scan-operation finds target records. lxd can handle every feature better than nxd and sxd since no redundant data must be updated.
- When comparing serialisation-time among the three databases, nxd's time is 2-3 times greater than that of sxd's time and 6-10 times greater than that of lxd's time. This is because nxd must read data from the disk while serialisation of sxd and lxd reads data from the main memory.
- The experimental results have shown that the XML update language has been translated correctly. The experiments, besides checking the correctness of our design and implementation, also reinforce the validation of the logical statements expressed in Chapter 5.

- Overall the experimental study show that the feeling favoured in this research for updating linked XML documents with an ORDB (lxd) is generally more efficient than in the other approaches considered (sxd and nxd). The exception is when lxd is used with a cold cache but this situation is not likely to occur often in a production environment.

7.5 Summary

In this chapter, we indicated the purposes of the experimental study and explained our methodology to evaluate a native XML database called X-Hive and the prototype implemented for updating XML documents kept in an ORDB. Then the experimental study is conducted to compare the performance of updating X-Hive keeping redundant data by using XUpdate and updating our two XML databases storing redundant data and non-redundant data with the XML update language designed in Chapter 4.

Several XML benchmarks have been proposed for evaluating XML query processing; thus their methodologies are adapted for use in evaluating XML update processing. The results of our experimental study show that:

- Caching data has a great effect on the update performance of an ORDB.
- In a cold cache, a native XML database outperformed an ORDB storing non-redundant data whereas in warm and hot caches, the result is the opposite.
- Overall updating non-redundant data has a better performance than updating redundant data.
- In a native XML database, the delete operation takes the shortest elapsed time while the replace operation takes the longest elapsed time whereas in an ORDB, the insert operation takes the shortest elapsed time.
- A native XML database has a weak point in handling the *regular path expression* whereas an ORDB storing redundant data in the case of deletion cannot handle the *change selectivity* well.
- The correctness of translating the XML update language has been demonstrated.

Chapter 8 will close this thesis with the conclusion of our research and suggest the trend of the future research.

Chapter 8: Conclusions and further work

This chapter draws conclusions on our research work reported in this thesis by revisiting the problems outlined in Chapter 1. In addition to this, a summary of the major contributions of the research is provided and suggestions for possible areas of further research are presented.

The remainder of this chapter is organised as follows: Section 8.1 provides conclusions of the thesis. In section 8.2, the major contributions for this research are summarised. Section 8.3 suggests several interesting avenues for further work to extend our research and indicates the limitations of our research. Section 8.4 provides the reflection on our methodology. Finally, section 8.5 concludes with the overall achievements of this thesis.

8.1 Conclusions of the thesis

This thesis has addressed some of the problems that remained unsolved due to the immaturity of the facilities for updating semi-structured data.

The thesis presents a solution for updating XML via ORDB to advance techniques in this area. Our research focuses on XML as the standard representation for semi-structured data. Our approach is one of the very rare XML update facilities that exploits conventional database techniques to update XML documents. The main conclusions drawn from this thesis are given below.

- This thesis presents a solution to the following five problems:
  1. The preservation of constraints during updating
  2. Data inconsistency and low performance caused by redundant data
  3. Joins of documents in updates
  4. Updating XML data whose structure is known partially sometimes giving a bad performance
  5. Updating XML data whose structure is recursive

The thesis describes a solution for updating XML documents which can solve these problems as shown below.

Firstly, our approach updates XML documents through ORDB; hence XML structure and constraints are mapped to an ORDB. By mapping XML constraints to ORDB constraints, the constraint problem has been solved since the preservation of XML constraints becomes the responsibility of the ORDB engine. However, the ORDB technology does not support the cardinality constraint; thus an internal mechanism to handle this constraint is developed as shown in section 3.3.4.

Secondly, `rlink` is used to represent a relationship between elements in different XML documents. Thus non-redundant data can be kept in multiple documents; data inconsistency caused by data redundancy can be solved and the performance of updating XML documents is improved since no redundant data must be updated. This evidence is shown in section 7.3.1, which compares the performance for updating an XML document storing redundant data and multiple linked XML documents keeping non-redundant data.

Thirdly, `rlink` is mapped to an ORDB as a reference between a foreign key and a primary key. Thus joins of documents via `rlink` in an XML update command are translated into joins between tables in SQL commands. The standard SQL does not support joins between tables in a delete/update command; therefore this command is transformed to a delete/update command with a sub-query as shown in section 4.2.1, presenting the techniques for translating XML update language into SQL.

Fourthly, the information derived from mapping XML to an ORDB can be used to determine the ORDB structure from the path specified in an XML update command. Thus the path in a regular path expression can be quickly identified to indicate which fields or tables are involved in the regular path expression. This speeds up the update of XML data by using a regular path expression as shown in section 7.3.3, which compares the performance of update features for our approach and a commercial native XML database.

Finally, XML data whose structure is recursive is updated by creating a recursive function whose syntax is derived from XQuery. The recursive

function for updating data with a recursive structure is translated into a series of SQL commands. For translating this recursive function, variables are considered as tables since only the tables can be directly manipulated by SQL commands as shown in section 4.3, which presents the translation of a recursive function.

- In our work, the change into an ORDB is reflected to XML documents with the ORDB being used to preserve constraints during updating and to indicate the elements in XML documents which should be updated. This makes it possible to query data from the XML documents instead of just the ORDB because reconstructing (fragments of) XML documents from the ORDB can be expensive [47, 126].
- The solution for XML updates is implemented as a prototype. Our approach is one of the very rare XML update approaches, in the area of applying conventional databases, to handle XML data, which has actually been implemented. Another XML update approach in this area is updating XML in RDB [124]; however this work [124] has not been implemented. Thus our research shows the possibility to update XML documents through conventional databases and helps to validate our approach.
- The performance of the solution has been assessed by the experimental study based on the prototype. The performance evaluation is using cold, warm and hot caches. To the best of our knowledge, the evaluation of the XML processing on all data caching states (i.e. cold, warm and hot caches) has never been conducted before.
- Our approach is applicable to other semi-structured data such as SGML and any other semi-structured data providing DTDs.

8.2 A summary of contributions

The main goal of this thesis was to devise a solution for updating semi-structured data by using XML as the standard representation for semi-structured data. This goal leads to five major contributions: the review of related literature, the solution for XML updates, the prototype implementation, the experimental study and publications. The summary of these contributions is as follows.

The review of related literature

The existing literature shows that there are two dominant solutions for handling XML data: handling XML data by applying conventional databases such as RDB and by native XML databases. Both solutions expose the immaturity of the XML update area. In the thesis we devised a solution for updating XML documents based on the application of conventional databases, since previous research presenting XML updates based upon this solution are very rare. The works based on this solution are very rare because W3C have not presented the standard for the XML update language; thus researchers are not sure about the syntax of the standard language which will be used for translating into SQL. However our work translates the features of XQuery; thus when the XML update standard is released, our work still can be applied to the standard since the standard will be based on XQuery [136]. The literature identifies several problems as follows:

- Existing XML databases cannot guarantee the preservation of constraints when updates are performed.
- None of the existing XML update languages can join or update multiple linked documents.
- Only one existing approach presents an XML update language which updates data whose structure is recursive but it does not provide any implementation [83].
- Data inconsistency and low performance result from redundant data kept in an XML document.
- Using regular path expressions, especially descendent path expressions ('//'), can slow the process of data query drastically.

- Two specific problems for applying conventional databases to handle XML are:
    - Little previous work handles XML constraints during mapping XML to databases; for those that attend to the definition of constraints, several workers define constraints conflicting with each other or define constraints that are impracticable in the current technologies.
    - Several features of XQuery such as recursive function, aggregate functions and quantifier have never been translated into SQL.

The solution for XML updates

Identifying a solution for updating XML documents is the most important contribution of this research. XML documents are updated via ORDB to exploit the maturity of the underpinning relational engine. Three main steps of the XML update solution are: (a) mapping XML schemas to ORDB schemas, (b) translating the XML update language into SQL, and (c) reflecting the change from the ORDB to the XML documents.

The capabilities of the XML update solution are as follows:

- The order of elements in XML documents is preserved after updating;
- The constraints of XML document are checked during updating;
- Six supporting features are inherited from XQuery: FLW(R|I|D) expression, conditional expression, quantifier, aggregate functions, (non-recursive) user-defined function and recursive function;
- Multiple XML documents can be joined and can be updated in one XML update command; and
- Update commands based on regular path expressions are processed efficiently.

Although our approach is based on exploiting conventional database technologies for XML data management, the objective of using an ORDB in this research is different from that of other related work. Previous work used a conventional database as a DBMS of XML documents but our approach uses an ORDB to preserve constraints during updating and mark the elements in XML documents which should be updated. Thus updates are performed on real XML documents as well. This makes it possible to query XML data from XML documents instead of the ORDB. For example, using Kweelt [111] (an implementation of XQuery for querying XML documents directly), the result from querying is returned in XML format without any conversion. Querying

XML data from XML documents reduces the cost of converting data kept in an ORDB to XML format, since nowadays [138], the major expense of exchanging messages between Web Services comes from converting data held within a service such as between a database and XML format.

The prototype implementation

The prototype of the XML update solution has been implemented to realize the proposed techniques. This prototype helps to validate the approach and shows the possibility to update XML documents through conventional databases as well as providing a fundamental basis for conducting the experimental study.

The prototype has been implemented by using six tools: Java, dom4j, Xerces, Jaxen, ANTLR and Oracle. Java is used as the language for basic coding. Dom4j and Xerces are used for updating XML data modelled as DOM trees. Jaxen is an XPath engine which is plugged in to Xerces. ANTLR is used to parse the XML update language. Oracle is used as an ORDBMS for storing XML data.

The experimental study

The experimental study has been conducted to serve five purposes as follows:

1. To study the effect of data redundancy.
2. To evaluate the performance in different cache states.
3. To compare the performance of three update operations: replace, delete and insert.
4. To compare the performance of seventeen update features of the XML update language.
5. To check the correctness of translating the XML update language into SQL.

The results from the experimental study can be summarised as follows:

- The ORDB storing non-redundant data has the best performance in every case for warm and hot caches. However, in a cold cache, the native XML database keeping redundant data has the best performance in most cases.
- The ORDB can take advantage from caching data more than the native XML database.
- The different performance among three update operations in the native XML database is strongly marked compared to that in the case of the ORDB.

- The ORDB storing non-redundant data can handle every feature of commands better than the databases keeping redundant data.
- The XML update language is translated into SQL correctly

Publications

- Some parts of chapters 1, 3, 4 and 5 are published in:
  Amornsinlaphachai, P., Rossiter, N. and Ali, M. A.: Updating XML using Object-Relational Database. British National Conference, Sunderland, UK., *Lecture Notes in Computer Science* **3567,** p. 155-160, 2005.
- Some parts of chapter 2 and the translating rules in Chapter 4 are published in:
  Amornsinlaphachai, P., Rossiter, N. and Ali, M. A.: Translating XML update language into SQL. *Journal of computing and information technology*, **14**(2), p. 81-100, 2006.
- Some parts of chapter 2 and the mapping rules in chapter 3 are published in:
  Amornsinlaphachai, P., Rossiter, N. and Ali, M. A.: Storing Linked XML documents in Object-Relational DBMS. *Journal of computing and information technology,* **14**(3), p. 225-241, 2006.

8.3 Limitations of this research and Further work

Our thesis has some limitations that should be removed in the future since these limitations are related directly to updating XML in a conventional database. The limitations are updating XML structure, mapping XML based on XML Schema and optimisation.

- It is possible that sometimes we want to change the XML structure. For example, Thai people do not have middle names; thus most database systems in Thailand do not contain this field. If we want to keep the names of other foreign people who have middle names, we need to change the database schema.
- For mapping XML based on XML Schema, in this thesis, only DTDs are mapped to an ORDB. However XML Schema is another standard from W3C and the use of XML Schemas is increasing. Thus a complete system should support both DTD and XML Schema.

- In the case of optimisation, although we have presented some optimisation techniques during the translation of the XML update language into SQL, it should be possible to optimise further SQL commands derived from the translation.

For further work, since XML can be used as a DBMS, it should have the facilities usually found in conventional DBMSs such as transactions, concurrency control and security. The facilities for transactions, concurrency control, security and querying XML are beyond the scope of the thesis. Indeed existing XML DBMSs also lack these facilities except for providing simple XML query facilities. Another further work is updating XML in a native XML database although it is not an extension to our work but is another popular approach for handling XML.

Updating the structure of XML

One of the characteristics of semi-structured data is a changeable schema. Thus it is possible that the schema/structure of XML documents can change. If the XML schema is changed, the database schema must be changed too. In conventional databases, schema updates are very expensive since usually, to update the schema, all data must be unloaded from the database, the database is dropped and recreated and then the data is loaded into the database again. It is a challenging job to invent a solution for updating the database schema while keeping the performance of the update acceptable.

Mapping XML to an ORDB based upon XML Schema

XML Schema is much more complex than DTD since it is rich in features and constraints; therefore designing mapping rules covering all or most of its features and constraints is a non-trivial task. The techniques for updating XML data kept in an ORDB presented in this thesis are still applicable for mapping XML Schema to an ORDB since the rules for translating the language and propagating the change from an ORDB to XML are independent of any XML schema used in mapping.

Optimisation

Since an XML update command is translated into SQL commands, adapting the concept of optimising SQL or other query languages used for conventional databases is possible. For instance, it is possible to adapt the notion of distributed processing [6]: an

execution plan of one OQL command is created and partitioned, then each partition is executed over distributed query evaluators.

In order to apply this concept to optimise the updating of XML documents, instead of distributing each partition of the execution plan of an SQL command, several SQL commands may be processed in parallel since one XML update command can be translated into several SQL commands. Nevertheless, a subtle issue remains with the order of processing of each update command since one XML update command can mix several update clauses which may be related to each other, for example, an XML update command:

```
Replace author/publication/title with <title>XML</title>
Where author/e-mail = 'jo@yahoo.com',
Replace author/e-mail with <e-mail>jo_smith@yahoo.com</e-mail>
Where author/e-mail = 'jo@yahoo.com'
```

If the above command is processed in parallel and the second replace clause is finished before the first replace clause, the title of publication will never be updated. There are therefore some subtle issues which must be considered for optimisation; especially when there are several update clauses in one command that are related to each other.

Transaction processing

One transaction can relate to more than one update command. Every command will be completed, if the transaction is completed, or all commands will be aborted if the transaction is aborted; thus all entities are always in a consistent state.

Designing a manager for transaction processing for updating XML documents via ORDB is related to not only ORDB transactions but XML document transactions as well. The difficulty of designing a transaction manager is to ensure that data both in an ORDB and in XML are always consistent and equivalent. An ORDB provides a mechanism for transaction processing already; however, XML does not have any mechanism for this.

Handling multi-user access (concurrency control)

It is possible while a user is updating XML data, that the other users want to update or query the same data or different data but within the same documents; thus the system should provide the ability to control access to the same data or the same documents

such as locking at a data level or at a document level. Locking at the document level means that it does not let other users access other data in the same document while another data item is being updated. This is contrary to locking at the data level that lets other users access other data in the same document while another data item is being updated.

Actually, locking data in an ORDB can be handled by the facilities in its own engine, provided to manage concurrency. However, it is not easy to determine which data in ORDB should be locked; since one part in an XML document can be dispersed over several tables; thus to lock only one XML fragment, data from several tables can be involved.

Security for accessing data

Usually each user has a different right to access data; thus the system should provide a security service to define access rights for each user to access data. Besides defining access rights to access whole documents, sometimes only some parts of documents are allowed to some users; thus usually this can be defined with views. It is possible to create an XML view and define the access rights on the view. If views are granted with the update privilege, this leads to another challenging issue: updating XML views. Updating XML views is a difficult process since such views are a logical presentation of XML documents defined by queries; thus often it is vague how to trace view updates back to updates of base data: for example a view created by using aggregate functions.

Querying XML data from an ORDB

In the thesis, only the solution for updating XML data stored in an ORDB has been presented. This research can be extended with a capability for querying XML data kept in an ORDB by applying the techniques for translating the XML update language to translate XQuery into SQL. However, there may be concerns about performance since reconstruction of XML documents is expensive.

Updating XML documents kept in a native XML database

Updating XML in a native XML database is an interesting work since currently its capability is quite limited. However, we should bear in mind that many features of

databases such as constraint handling, joins of documents, transaction processing and concurrency control must be developed from scratch.

8.4 Reflection on methodology

There are several research methodologies in computer science such as simulation, mathematical proof and experiment study. Our research uses the experimental study to test the hypotheses related to conceptual design and to system performance. Thus the experimental study appears to be an appropriate methodology as in testing the hypotheses we can also validate our solution by comparing the results from the prototype with the expected results.

This methodology helps to answer the research question which is used to develop the hypothesis verified by the experimental study. The conclusion derived from the experimental results supports the hypotheses as follows.

1. By confirming the conceptual and logical approach to updating XML through implementation and testing, as discussed in Section 8.1.
2. By showing that the performance of updating XML with our method, over a variety of cache states, data size and degrees of redundancy, is relatively good, in particular:
    a) updating non-redundant XML data yields better performance overall than updating redundant XML data
    b) updating data in a conventional database always outperforms updating data in NXD for commands based on regular path expressions.

One limitation of our methodological approach is validating it by comparing the results from the prototype with the expected results since there is a limited number of test cases. To improve our methodological approach, it is possible to prove the correctness of our solution by a mathematical proof. In our work, the semantics of the XML update language are defined in a mathematical format because the meaning of a language should be given without any ambiguity. Also defined in a mathematical format, the logical statements, showing how our solution works, should be able to be implemented by any programming languages. To prove the correctness of our solution by mathematics, the rules for mapping XML, translating the language and propagating the

change need to be revised into a more formal form so that these rules can be used in a formal proof.

8.5 The overall achievement of the thesis

The overall achievement of the thesis can be summarized as follows.

- The first achievement is the presentation of a solution for updating XML as the standard representation of semi-structured data via ORDB. The objective of the solution is different from other work. An ORDB is used to preserve constraints and to indicate the positions of updating in XML documents whereas other works use the conventional database in the form of an XML DBMS.
- The second is the techniques for translating six features of the XML update language inherited from XQuery into SQL since several features of XQuery have never been translated before.
- The third is the translation of rules from mapping structure and constraints of XML to an ORDB since no work has been presented to show the practicable rules for mapping XML to an ORDB in the available ORDB technology.
- The fourth is the experimental study conducted on all data caching states. The performance evaluation of the XML processing on all data caching states has never been conducted before.
- The thesis also demonstrates the practicability and the correctness of the solution through the prototype.

Overall the author believes that the research makes an advance in the research area of updating XML. The author believes that the findings presented in the thesis will draw more attention to the area and stimulate more research in this field.

Appendix A: Translating XML update language into SQL

This appendix is an extension of Chapter 4, showing more examples for translating the update language.

Examples for translating the update language into SQL for a single XML document

A DTD for a single XML document and its schema graph are shown in Figures A.1-2.

```
<!ELEMENT Library (Publication*)>
<!ELEMENT Publication (Title, Author+, Year, Publisher, Reference?, Cost, SpecialCost?, ShippingCost?)>
<!ATTLIST Publication PubID ID #REQUIRED>
<!ATTLIST Publication PubType (book | article | journal) #IMPLIED>
<!ATTLIST Publication ContactAuthor IDREF #REQUIRED>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Cost (#PCDATA)>
<!ELEMENT SpecialCost (#PCDATA)>
<!ELEMENT ShippingCost (#PCDATA)>

<!ELEMENT Author (Name, Email?,  Telephone*)) >
<!ATTLIST Author AuthorID ID #REQUIRED>
<!ELEMENT Name (FName, MName?, LName)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT MName (#PCDATA)>
<!ELEMENT LName (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Telephone (Location, TelNo)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT TelNo (#PCDATA)>
<!ELEMENT Year (#PCDATA)>

<!ELEMENT Publisher (PName, Address?, SetupYear)
<!ATTLIST Publisher PID ID #REQUIRED>
<!ELEMENT PName (#PCDATA)>
<!ELEMENT Address (No, Street, City, Country, Zipcode)>
<!ELEMENT No (#PCDATA)>
<!ELEMENT Stree (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Zipcode (#PCDATA)>

<!ELEMENT Reference EMPTY>
<!ATTLIST Reference RefID ID #REQUIRED>
<!ATTLIST Reference RefPub IDREFs #REQUIRED>
<!ATTLIST Reference RefType (References | Bibliography | Miscelleneuos) "References">
```

Figure A.1 DTD for a single XML document

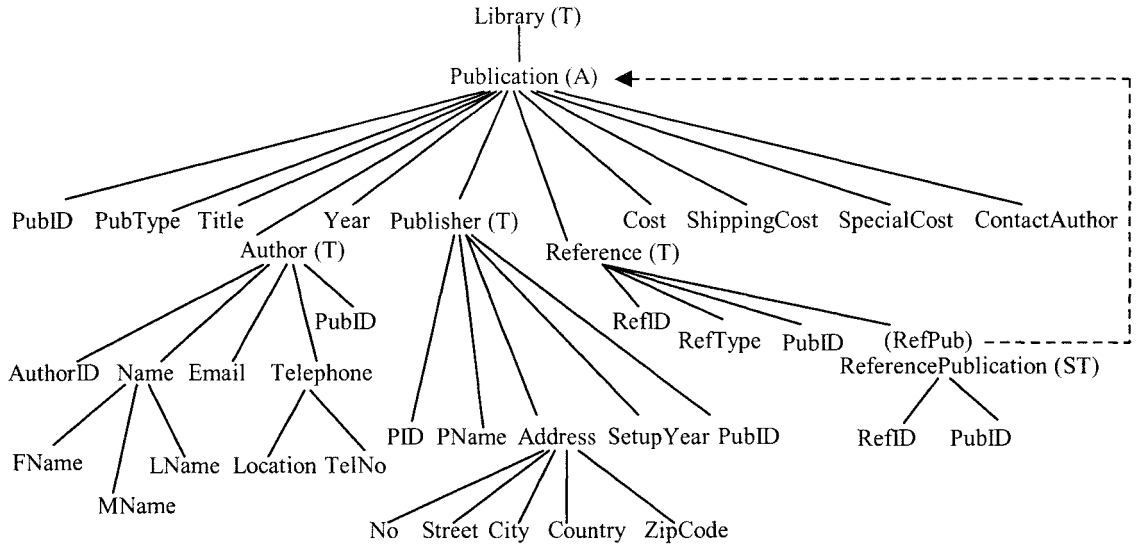

Figure A.2 ORDB Schema graph derived from DTD in Figure A.1

Example1: The command below updates PubType of the publication when there are more than two authors

```
For $p in doc("Library.xml")//Publication
Let $a := $p/Author
Where count($a) > 2
Replace $p@PubType with @PubType = "article"
```

1. count($a) is rewritten as count($a) group_by ($p.) count($a) and is used along with Where clause; thus Where clause is changed to having. The command is parsed as follows:

   ```
   bindF('Library.xml'//Publication, $p, 0)
   bindL($p/Author, $a, 0)
   having(count($a), > , 2, 0)
   group_by($p, 0)
   update($p/PubType, 'article', 1)
   ```

2. A graph is created according to paths, elements and attributes in functions and mapped to a database schema graph. Finally, the functions are mapped to the graph. The graph after mapping is shown in Figure A.3.

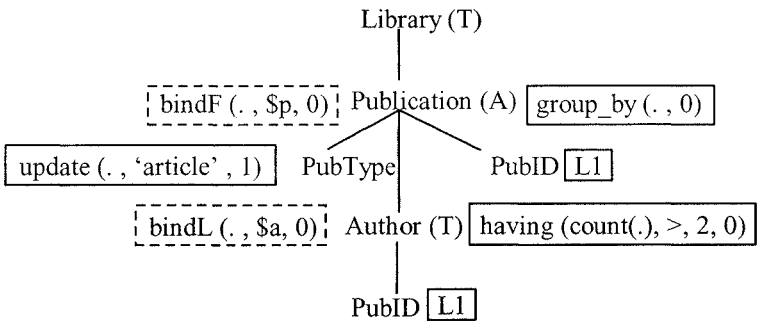

Figure A.3 Graph after mapping database schema graph and SQL functions

3. From Figure A.3, the function group_by is on a node converted to abstract data field and this field has no sibling; thus the function is pushed down to primary key field as shown in Figure A.4.

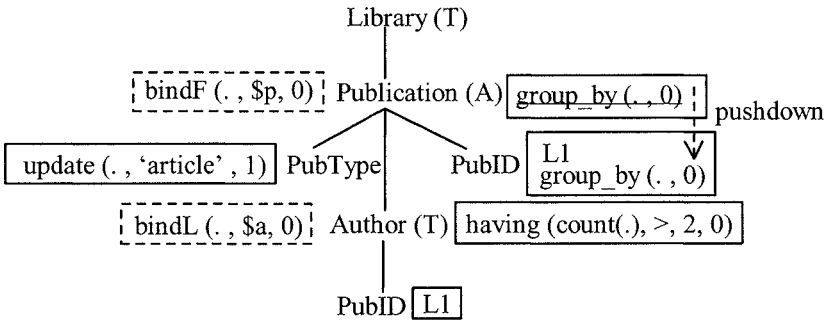

Figure A.4 Graph after pushing group_by

4. From the graph, the update join command is generated as follows:

```
Update Library L
From L, Author A
Set L.Publication.PubType = 'article'
Where L.Publication.PubID = A.PubID
Group by L.Publication.PubID
Having count(*) >2
```

5. The update join command is rewritten into SQL command as follows:

```
Update Library L
Set L.Publication.PubType = 'article'
Where L.Publication.PubID in
                        (select L.Publication.PubID
                         from Library L, Author A
                         where L.Publication.PubID = A.PubID
                         group by L.Publication.PubID
                         having count(*) >2
                        ) ;
```

Example 2: The command below update PubType of the publication which has some author whose last name is 'Smith'

```
For $p in doc("Library.xml")//Publication
Where every $a in $p/Author
      satisfies ($a/Name/LName = "Smith")
Replace $p@PubType with @PubType = "book"
```

1. ```
   For $p in doc("Library.xml")//Publication
   Where every $a in $p/Author
         satisfies ($a/Name/LName = "Smith")
   ```
   is translated into:

   ```
   For $p in doc("Library.xml")//Publication
   Let $a := $p/Author
   Where $a/Name/LName = "Smith"
   And count($a) = (For $pp in doc("Library.xml")//Publication
                    Let $aa := $pp/Author
                    Where $p = $pp
                    Return count($aa))
   ```

   Thus the command becomes:

   ```
   For $p in doc("Library.xml")//Publication
   Let $a := $p/Author
   Where  $a/Name/LName = "Smith"
   And count($a)  = (
                   For $pp in doc("Library.xml")//Publication
                   Let $aa := $pp/Author
                   Where $pp = $p
                   Return count($aa)
                  )
   Replace $p@PubType with @PubType = "book"
   ```

2. Then the command is rewritten in SQL functions as follows:

   ```
   $p = bindF('Library.xml'//Publication)
   $a = bindL($p/Author)
   where ($a/Name/LName , = , 'Smith' , 0)
   group_by($p, 0)
   having(count($a), = , :1, 0)
   $pp = bindF('Library.xml'//Publication, 1)
   $aa = bindL($pp/Author, 1)
   select (count($aa),1)
   where($pp, = , $p, 1)
   group_by($pp, 1)
   update ($p/PubType, "article", 2)
   ```

3. A graph is created according to elements and attributes of the functions and then the graph is mapped to database schema graph as shown in Figure A.5(a). Finally, the functions are mapped to the graph as shown in Figure A.5(b).
   ```

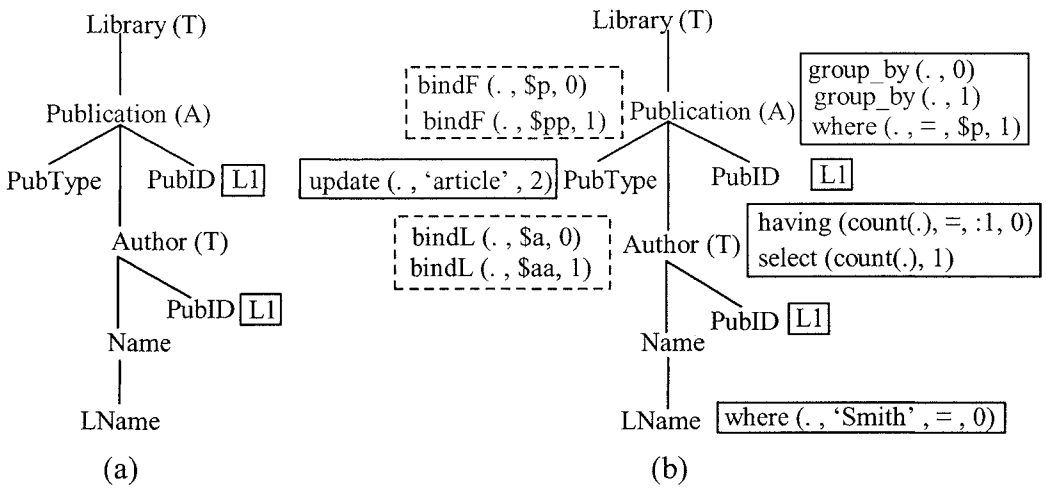

Figure A.5 graph after (a) mapping ORDB schema graph, (b) mapping SQL functions

4. From the Figure A.5(b), the function group_by and where are in a node converted to abstract data field and this field has no sibling; thus the function is pushed down to primary key field as shown in Figure A.6.

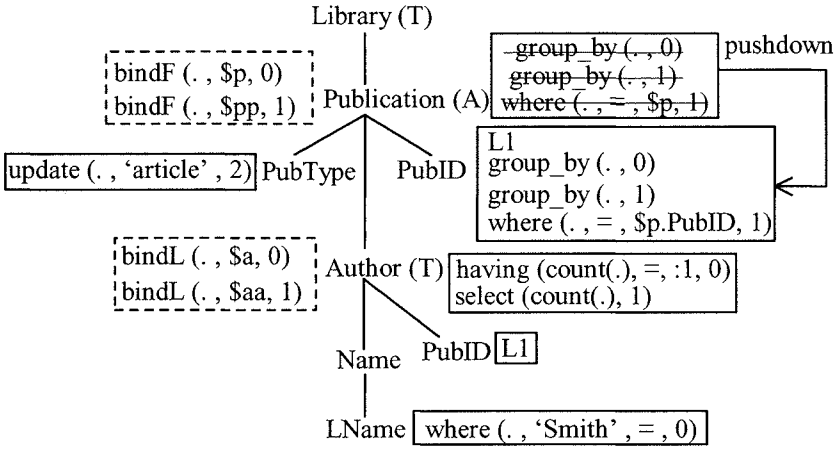

Figure A.6 Graph after pushing-down group_by and where functions

5. The update join command is generated from the graph as follows:

```
Update Library L
From L, Author A
Set L.Publication.PubType = 'article'
Where A.Name.LName = 'Smith'
And L.Publication.PubID = A.PubID
group by L.Publication.PubID
having count(*) =
            (select count(*)
             from  From Library LL, Author AA
             where LL.Publication.PubID = L.Publication.PubID
             and LL.Publication.PubID = AA.PubID
             group by LL.Publication.PubID )
```

6. The update join command is rewritten into SQL command as follows:

```
Update Library L
Set L.Publication.PubType = 'article'
Where L.Publication.PubID in
        (select L.Publication.PubID
         from Library L, Author A
         where A.Name.LName = 'Smith'
         and L.Publication.PubID = A.PubID
         group by L.Publication.PubID
         having count(*) =
             (select count(*)
             from Library LL, Author AA
              where LL.Publication.PubID=L.Publication.PubID
               and LL.Publication.PubID = AA.PubID
               group by L.Publication.PubID
             )) ;
```

Example 3: The recursive function below updates the year of publications which are both direct and indirect references of the publication whose title is 'XQuery'

```
1 define function allRef($pub as element()*)
2{
3     For $rp in $pub/Reference/@RefPub → Publication
4     Replace $rp/Year with <Year>2004</Year>
5     allRef($rp)
6}
7  For $p in doc("Library.xml")/Library/Publication
8  Where $p/Title = "XQuery"
9  allRef($p)
```

The process starts from the clause in line 7. Then it is translated into PL/SQL as follows.

1. Rewrite each clause until the first calling function in body of function is found. The clauses and the result of rewriting by using rewriting rules 1-2 are shown in Figure A.7.

XQuery clauses	Result of rewriting
7 For $p in doc("Library.xml")/Library/Publication 8 Where $p/Title = "XQuery"	Insert into $p select (/Library/Publication, 0) where ($p/Title, = ,'XQuery', 0);
9 allRef($p) 1 define function allRef($pub as element()*)	Insert into $pub Select * from $p;
3 For $rp in $pub/Reference/@RefPub→ Publication	Insert into $rp Select($pub/Reference/@RefPub→ Publication, 1);
4 Replace $rp/Year with <Year>2004</Year>	update($rp/Year, '2004', 2);

Figure A.7 Result of rewriting clauses in the update command

2. Create loop structure when the first calling function is found in body of the function. Clauses in the loop which are rewritten by using rewriting rules 1-2 as

shown in Figure A.8 and rewriting clauses in the loop produces the loop structure in Figure A.9.

Clauses in loop	Result of rewriting
5 allRef($rp) 1 define function allRef($pub as element()*)	Insert into $pub Select * from $rp;
3 For $rp in $pub/Reference/@RefPub→ Publication	Insert into $rp select($pub/Reference/@RefPub→ Publication, 3);
4 Replace $rp/Year with <Year>2004</Year>	update($rp/Year, '2004', 4);

Figure A.8 Result of rewriting clauses in function body which are processed in loop

```
Loop
   If SQL%RowCount > 0 then
          Insert into $pub
          Select * from $rp;
          Insert into $rp
          select ($pub/Reference/@RefPub → Publication, 3);
          update($rp/Year, '2004', 4);
   Else
          Exit;
   End If;
EndLoop;
```

Figure A.9 Loop structure derived from rewriting clauses in function body

3. Replace the variables by using rule 3. The result of replacing variables in clauses which are outside and inside the loop are shown in Figures A.10 and A.11 respectively.

Clauses	Result of replacing variables
Insert into $p select (/Library/Publication, 0) where ($p/Title, = ,'XQuery', 0);	Insert into Array select (/Library/Publication, 0) where (Publication/Title, = ,'XQuery', 0);
Insert into $pub Select * from $p;	Insert into ProcessingArr Select * from Array;
Insert into $rp Select($pub/Reference/@RefPub→ Publication, 1);	Insert into Array select(Publication/Reference/@RefPub→Publication,1) where(Publication, in, , 1) (Select * from ProcessingArr);
update($rp/Year, '2004', 2);	update(Publication/Year, '2004', 2) where(Publication, in, , 2) (Select * from Array);

Figure A.10 Result of replacing variables in clauses which are outside loop

Clauses in loop	Result of replacing variables
Insert into $pub Select * from $rp;	Insert into ProcessingArr Select * from Array;
Insert into $rp select($pub/Reference/@RefPub→ Publication, 3);	Insert into Array select(Publication/Reference/@RefPub→ Publication,3) where(Publication, in, , 3) (Select * from ProcessingArr) ;
update($rp/Year, '2004', 4);	update(Publication/Year, '2004', 4) where(Publication, in, , 4) (Select * from Array);

Figure A.11 Result of replacing variables in clauses which are inside loop

4. Follow the concept of passing a variable's value, thus each clause for insertion of data is preceded by a clause for deleting old data in the table. The result is shown in Figure A.12.

5. Translating each group of SQL functions independently into SQL commands as follows:

   5.1 Each graph is created according to elements and attributes of each group of SQL function embedded in a PL/SQL command.

   5.2 The graph is mapped to a database schema graph. Then join keys and join symbols are added to the graph.

   5.3 The functions are mapped to the graph. The select function performed on element referenced from IDREFs of its descendant by dereferencing symbol (→) will be pushed down to IDREFs (coming before the symbol dereferencing) linking to such element. The result is shown in Figure A.13.

   5.4 There are select and where functions performed on element converted to abstract data type field without sibling; thus these functions are pushed down to the proper primary key as shown in Figure A.14.

   5.5 SQL commands generated from the graphs are shown in Figure A.15. SQL commands of the functions in groups 3 and 4 are the same as those in group 1 and 2 respectively; thus we show only SQL commands generated from the functions in groups 0-2.

   5.6 PL/SQL command after replacing SQL functions with SQL commands and generated from the graph is shown in Figure A.16.

```
Begin
       Delete from Array;
    Insert into Array
    select (/Library/Publication, 0)
    where(Publication/Title, = , 'XQuery' , 0);

    Delete from ProcessingArr;
    Insert into ProcessingArr
    Select * from Array;

    Delete from Array;
    Insert into Array
    select (Publication/Reference/@RefPub → Publication, 1)
    where (Publication, in, ,1)
    (select * from PricessingArr);

    update(Publication/Year, '2004', 2)
    where (Publication, in, ,2)
    (select * from Array);

    Loop
        If SQL%RowCount > 0 then
                  Delete from ProcessingArr;
                  Insert into ProcessingArr
                  Select * from Array;

                   Delete from Array;
                   Insert into Array
                   select (Publication/Reference/@RefPub → Publication, 3)
                   where (Publication, in, ,3)
                  (select * from ProcessingArr);

                   update(Publication/Year, '2004', 4)
                   where (Publication, in, ,4)
                   (select * from Array);
              Else
           Exit;
       End If;
      EndLoop;
End;
```

Figure A.12 Clauses for insertion of data are preceded by clauses for deleting old data

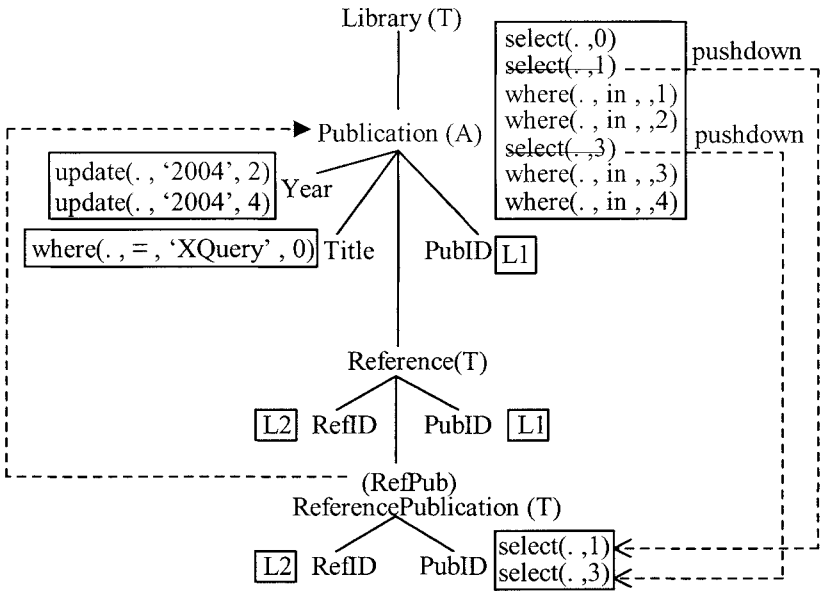

Figure A.13 Graph after mapping database schema graph and pushing select functions performed on element following dereferencing symbol

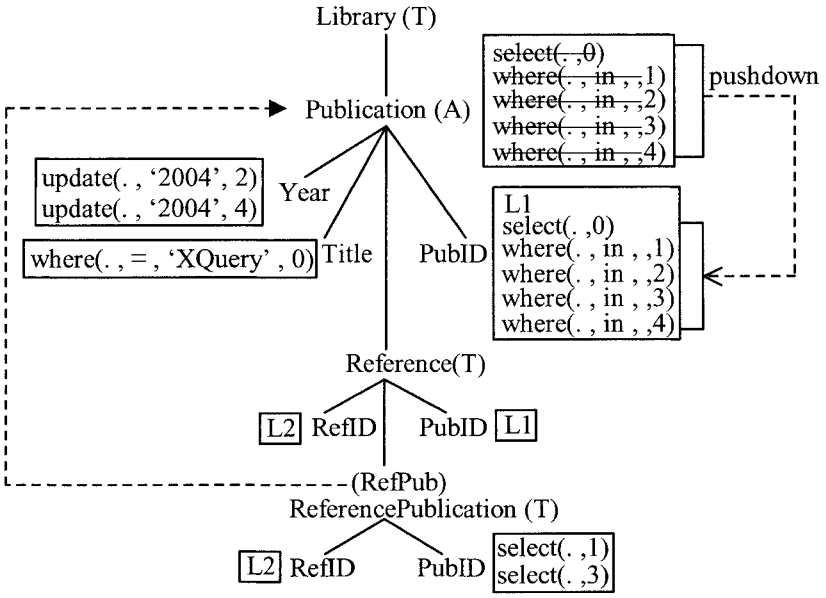

Figure A.14 Graph after pushing functions performed on abstract data type field

SQLfunctions	SQL commands
select (/Library/Publication, 0) where(Publication/Title, = , 'XQuery' , 0);	Select L.Publication.PubID From Library L Where L.Publication.Title = 'XQuery';
select (Publication/Reference/@RefPub → Publication, 1) where (Publication, in, ,1)	Select RP.PubID From Library L, Reference R, ReferencePublication RP Where RP.RefID = R.RefID And R.PubID = L.Pulbication.PubID And L.Publication.PubId in
update(Publication/Year, '2004', 2) where (Publication, in, ,2)	Update Library L Set L.Publication.Year = '2004' Where L.Publication.PubID in

Figure A.15 SQL commands generated from the graphs

```
Begin
    Delete from Array;
    Insert into Array
    Select L.Publication.PubID
    From Library L
    Where L.Publication.Title = 'XQuery';

    Delete from ProcessingArr;
    Insert into ProcessingArr
    Select * from Array;

    Delete from Array;
    Insert into Array
    Select RP.PubID
    From Library L, Reference R, ReferencePublication RP
    Where RP.RefID = R.RefID
    And R.PubID = L.Pulbication.PubID
    And L.Publication.PubId in  (select * from PricessingArr);

    Update Library L
    Set L.Publication.Year = '2004'
    Where L.Publication.PubID in (select * from Array);

    Loop
        If SQL%RowCount > 0 then

            Delete from ProcessingArr;
            Insert into ProcessingArr
            Select * from Array;

            Delete from Array;
            Insert into Array
            Select RP.PubID
            From Library L, Reference R, ReferencePublication RP
            Where RP.RefID = R.RefID
            And R.PubID = L.Pulbication.PubID
            And L.Publication.PubId in (select * from ProcessingArr);

            Update Library L
            Set L.Publication.Year = '2004'
            Where L.Publication.PubID in (select * from Array);
        Else
            Exit;
        End If;
      EndLoop;
End;
```

Figure A.16 PL/SQL command after replacing SQL functions with SQL commands

Examples for translating the update language into SQL for linked XML documents

Example 1: The command below inserts rlink-element linking to reference whose id is 'R999' into publication whose title is 'XQuery' and inserts first name to author (of publication whose title is 'XQuery') whose last name is 'Smith'

```
For $p in doc("Publications.xml")//Publication,
       $a in $p/Author ~>//Author
Where $p/Title = "XQuery"
Insert rlink(Reference, //Reference[@RefID = "R999"]) Into $p,
Insert <FName>Joey</FName> Into $a/Name
Where $a/Name/LName = "Smith"
```

1. Parse a command into update language functions

Insert rlink(Reference, //Reference[@RefID = "R999"]) Into $p is rewritten as:

insert ($p/Reference#References/Reference, 'R999',1)

since predicate of XPathExp in rlink is in the form PK(linkedEle) = value. The command is rewritten as SQL functions as follows:

```
bindF ('Publications.xml'//Publication, $p, 0)
bindF ($p/Author ~>//Author, $a, 0)
where ($p/Title, = , 'XQuery', 0)
insert ($p/Reference#References/Reference, 'R999',1)
insert($a/Name/FName, 'Joey', 2)
where($a/Name/LName, = , 'Smith', 2)
```

2. Translate the SQL functions into a graph having nodes correspond to paths, elements and attributes of the functions. Then we map the graph into ORDBschema graph and finally add key fields which are used to join the tables and add join symbols to the graph. The result of the graph is shown in Figure A.17.

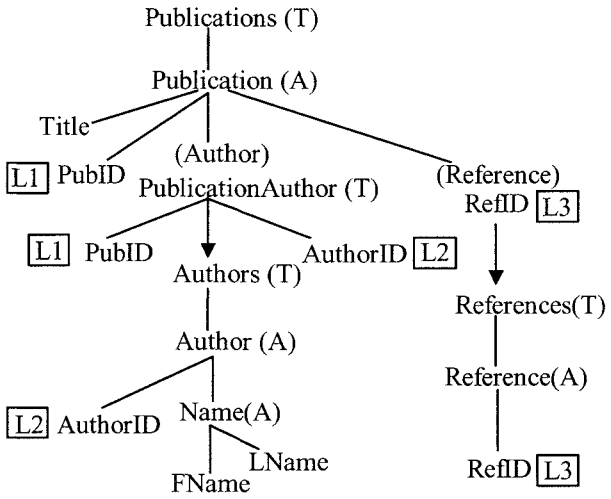

Figure A.17 Graph after mapping ORDB schema graph and adding join keys

3. Map the functions to the graph. Since two insert functions are performed on a node converted to a field, thus the functions are changed to update functions. The result is shown in Figure A.18.

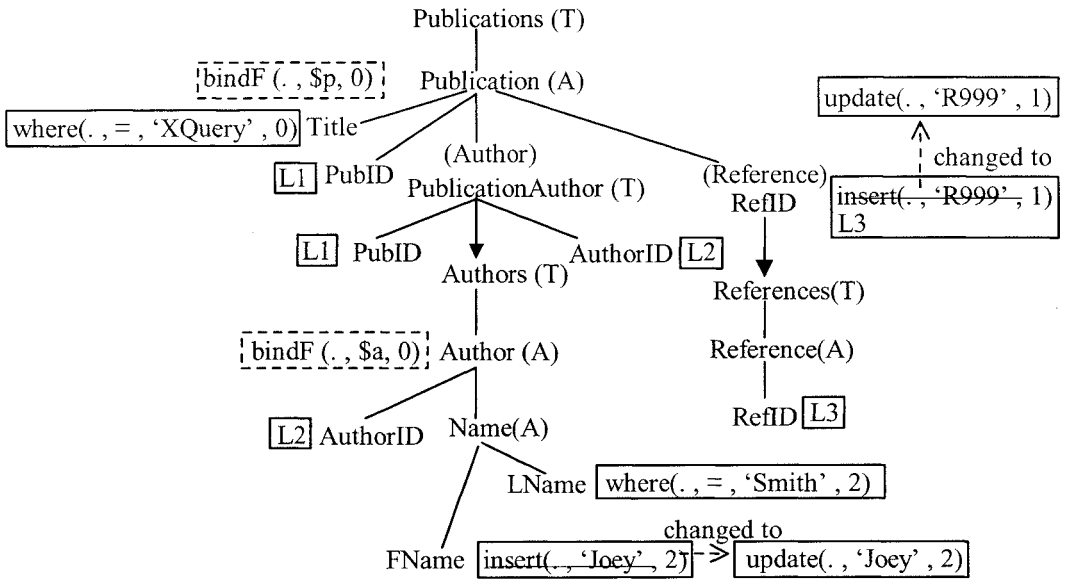

Figure A.18 Graph after mapping SQL functions and changing insert to update

4. Separate the graph into sub-graphs. Since there are two updated target tables: Publications and Authors; thus the graph is separated into two sub-graphs. In the first sub-graph, References is not related to updating Publications; thus L3 can be eliminated. According to optimization rules, there is no need to optimize both sub-graphs. Two sub-graphs are shown in Figure A.19.

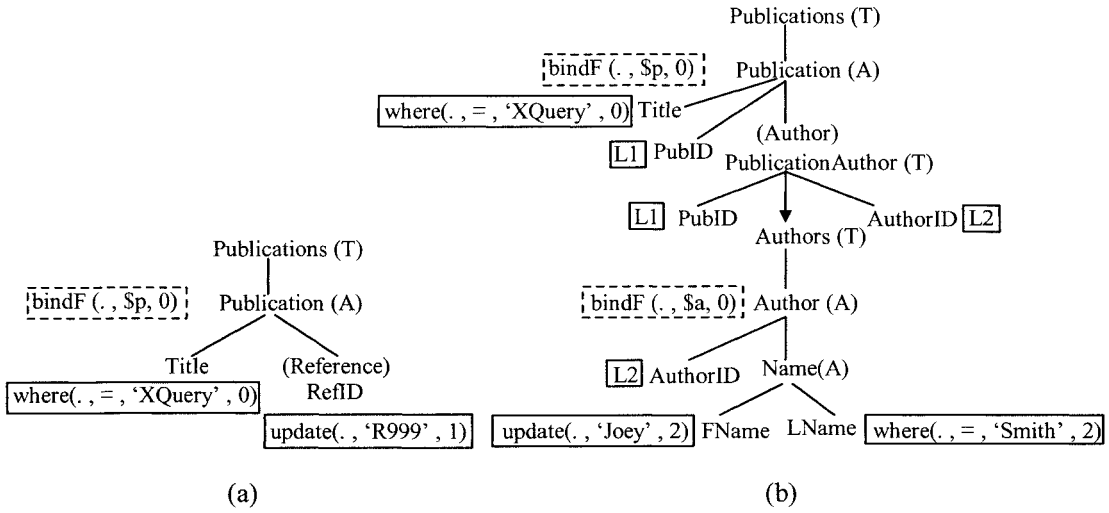

Figure A.19 Two sub-graphs for two update operators

5. Create SQL commands from each sub-graph

   From the first sub-graph, one table is involved in the update; thus the SQL command is generated as follows:

```
Update Publications P
Set P.Publication.RefID = 'R999'
Where P.Publication.Title = 'XQuery' ;
```

For the second sub-graph, there are three tables involved in the update; thus an update join command is generated as follows:

```
Update Authors A
From A, PublicationAuthor PA, Publications P
Set A.Author.Name.FName = 'Joey'
Where A.Author.Name.LName = 'Smith'
And A.Author.AuthorID = PA.AuthorID
And PA.PubID = P.Publication.PubID
And P.Publication.Title = 'XQuery'
```

The update join command is rewritten as an SQL command as follows:

```
Update Authors A
Set A.Author.Name.FName = 'Joey'
Where A.Author.AuthorID in
                    (select PA.AuthorID
                    from Authors A, PublicationAuthor PA,
                         Publications P
                    where A.Author.Name.LName = 'Smith'
                    and A.Author.AuthorID = PA.AuthorID
                    and PA.PubID = P.Publication.PubID
                    and P.Publication.Title = 'XQuery') ;
```

Example 2: The command below inserts a new reference.

```
For   $r in doc("References.xml")/References
Insert      <Reference RefID = "R999" RefType = "bibliography">
            rlink(Publication, //Publication[Title = "XQuery"])
            rlink(Publication, //Publication[Title = "XML"])

      </ Reference>
Into $r
After $r/Reference@RefID = "R555"
```

1. Since the content which will be inserted is of complex content; thus the complex content is shredded into several simple content and then the command is rewritten as follows:

```
For  $r in doc("References.xml")/References
Insert Reference into $r
Insert Reference@RefID = "R999"  into $r,
Insert Reference@RefType = "bibliography" into $r
Insert Reference/
       rlink(Publication,//Publication[Title="XQuery"]) into $r,
Insert Reference/
       rlink(Publication, //Publication[Title = "XML"]) into $r
```

2. The command is parsed into the functions as follows:

```
bindF('References.xml'/References, $r, 0)
insert($r/Reference, , 1),
insert($r/Reference/RefID, 'R999' , 1)
insert($r/Reference/RefType, 'bibliography' , 1)
insert($r/Reference/Publication#Publications/Publication, :1,1)
select(Publications/Publication/PK , :1)
where (Publications/Publication/Title, = 'XQuery' , :1)
insert($r/Reference/Publication#Publications/Publication, :2,1)
select(Publications/Publication/PK , :2)
where(Publications/Publication/Title, = 'XML' , :2)
```

3. The functions are converted to a graph having nodes corresponding to paths, elements and attributes of the functions as shown in Figure A.20(a).

4. The graph is mapped to database schema graph and then join keys and join symbols are added to the graph. Since there is an insert function performed on RefID which is primary key of References; hence the insert function is copied to foreign key (RefID) in child-table (ReferencePublication). The graph is shown in Figure A.20(b).

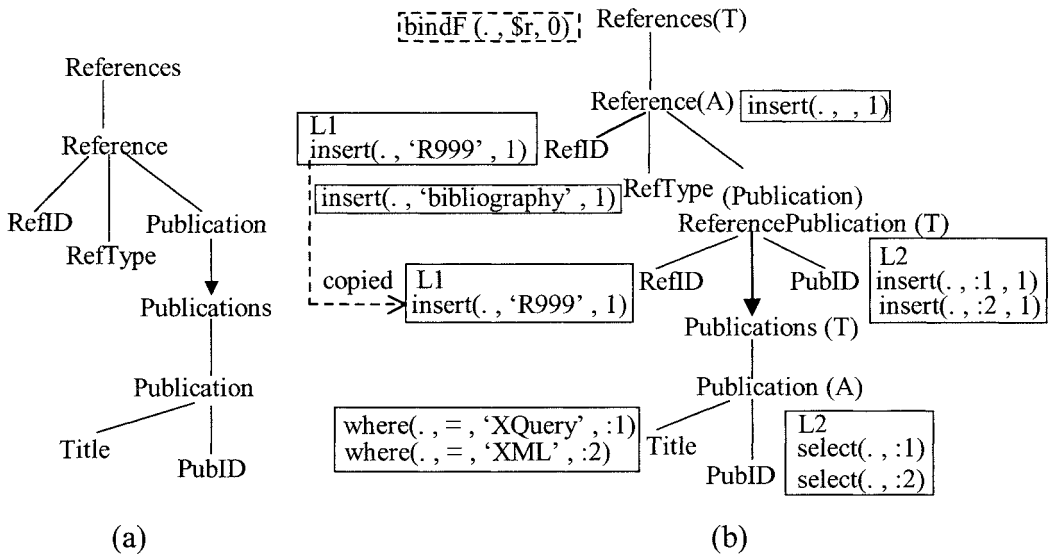

Figure A.20 (a) Graph with nodes corresponding to nodes in SQL functions, (b) Graph after mapping SQL functions and copying insert operator

5. The graph is separated into two sub-graphs as shown in Figure A.21.

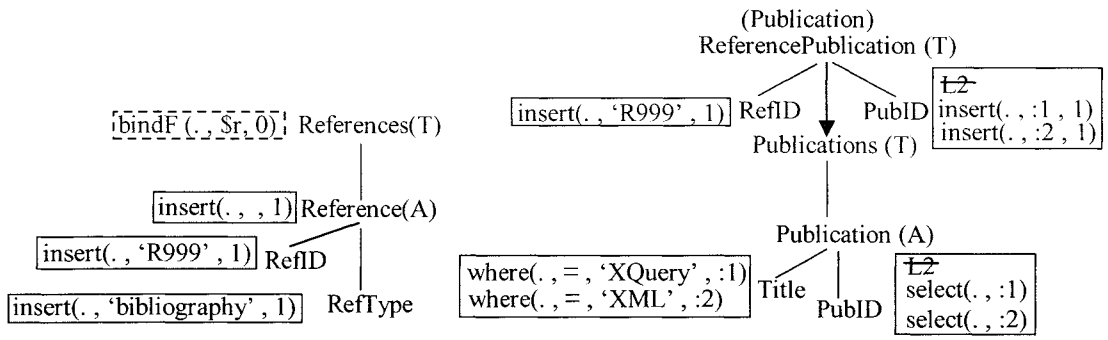

Figure A.21 Two sub-graphs for two insert operators

6. SQL command from the first sub-graph is generated as follows:

```
Insert References
Values (TReference('R999', 'bibliography'));
```

Note: assuming that Reference has type of TReference

For the second sub-graph, two tables are involved in the insert but select and where functions are on the same table; thus the join between tables is not required and therefore L2 can be eliminated. SQL commands are generated as follows:

```
Insert ReferencePublication
Select 'R999', P.Publication.PubID
From Publications P
Where P.Publication.Title = 'XQuery';

Insert ReferencePublication
Select 'R999', P.Publication.PubID
From Publications P
Where P.Publication.Title = 'XML';
```

Note: if there is more one insert function on a field, the number of insert statements will be generated according to the number of the insert functions.

Example 3: The command below updates PubType of the publication when there are more than two authors

```
For $p in doc("Publications.xml")//Publication
Let $a := $p/Author~>//Author
Where count($a) > 2
Replace $p@PubType with @PubType = "article"
```

1. The command is parsed to functions as follows:

```
bindF('Publications.xml'//Publication, $p, 0)
bindL($p/Author~>//Author, $a, 0)
having(count($a), > , 2, 0)
group_by($p, 0)
update ($p/PubType, 'article', 1)
```

2. A graph is created according to elements and attributes in functions and mapped to database schema graph. The graph is shown in Figure A.22(a). Then the functions are mapped to the graph as shown in Figure A.22(b). From Figure A.22(b), the function group_by is in a node converted to abstract data field without sibling; thus the function is pushed down to the primary key field.

3. From the graph, update join command is generated as follows:

```
Update Publications P
From P, PublicationAuthor PA, Authors A
Set P.Publication.PubType = 'article'
Where P.Publication.PubID = PA.PubID
And PA.AuthorID = A.Author.AuthorID
Group by P.Publication.PubID
Having count(*) > 2
```

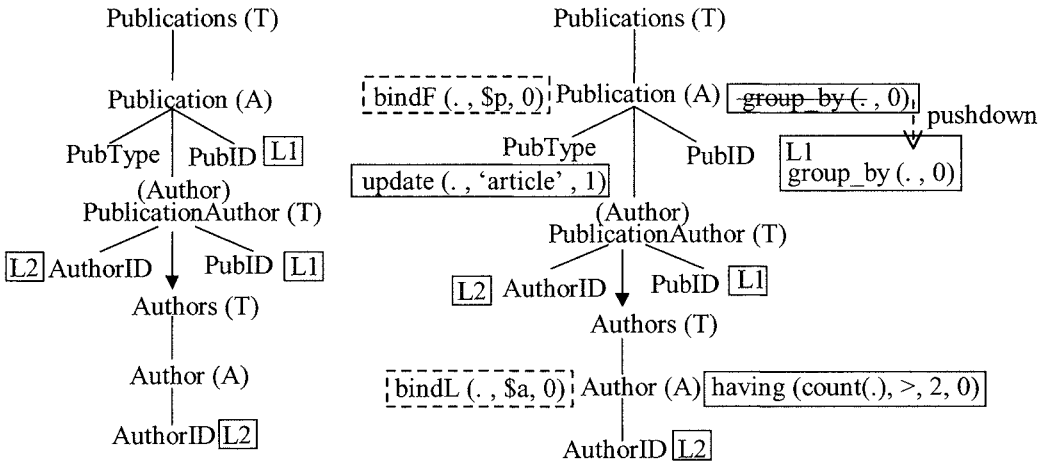

Figure A.22 Graph (a) after mapping ORDB schema graph, (b) after mapping SQL functions and pushing-down group_by

4. The update join command is rewritten into SQL command as follows:

```
Update Publications P
Set P.Publication.PubType = 'article'
Where P.Publication.PubID in
        (select P.Publication.PubID
         from Publications P, PublicationAuthor PA, Authors A
         where P.Publication.PubID = PA.PubID
         and PA.AuthorID = A.Author.AuthorID
         group by P.Publication.PubID
         having count(*) > 2
        );
```

Appendix B: Creating the XPath and logical statements

This appendix shows more examples for creating XPath as a parameter of the propagate functions and also expresses the logical statements for updating XML documents presented in Chapter 5.

Examples for creating XPath

Example 1: The command below inserts rlink-element linking to reference whose id is 'R999' into publication whose title is 'XQuery' and inserts first name to author (of publication whose title is 'XQuery') whose last name is 'Smith'

```
For $p in doc("Publications.xml")//Publication,
       $a in $p/Author ~>//Author
Where $p/Title = "XQuery"
Insert rlink(Reference, //Reference[@RefID = "R999"]) Into $p,
Insert <FName>Joey</FName> Into $a/Name
Where $a/Name/LName = "Smith"
```

There are two insert clauses in XQuery update command; thus two trees are created, a condition of the second clause is added to its tree and both trees are mapped to the database schema. The results are shown in Figures B.1(a) and B.1(b).

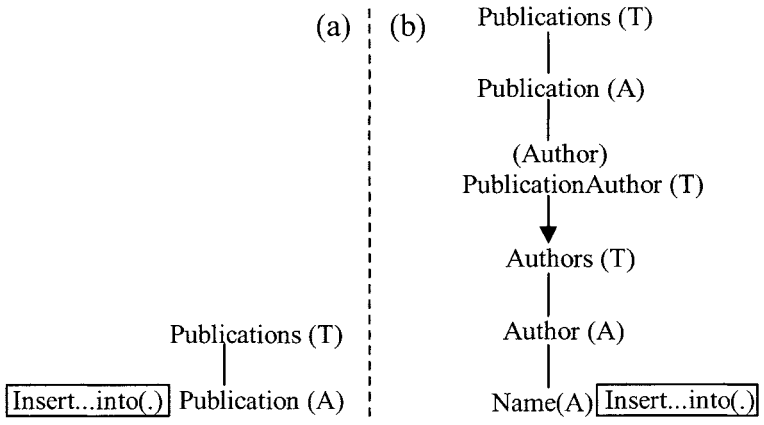

Figure B.1 Trees of (a) first and (b) second insert clauses

From Figure B.1(a), the target for inserting rlink-element: Reference (converted to FK) is Publication; thus ID of nearest ancestor of the rlink-element is a receiver for the

values of the primary key returned from ORDB. The nearest ancestor having ID is Publication; thus PubID is added to the tree. Supposing that publication whose tile = 'XQuery' has PubID = 'P111'; thus the returned value is 'P111'. The condition based on this returned value is added to the first tree. The result is shown in Figure B.2(a).

From Figure B.1(b), the target for inserting FName (simple field) is Name; thus ID of nearest ancestor of FName is a receiver for the values of the primary key returned from ORDB. The nearest ancestor having ID is Author; thus its ID (AuthorID) is added to the tree. Supposing that author whose last name = 'Smith' has AuthorID = 'A222'; thus the returned value is 'A222'. The condition based on this returned value is added to the second tree. The result is shown in Figure B.2(b).

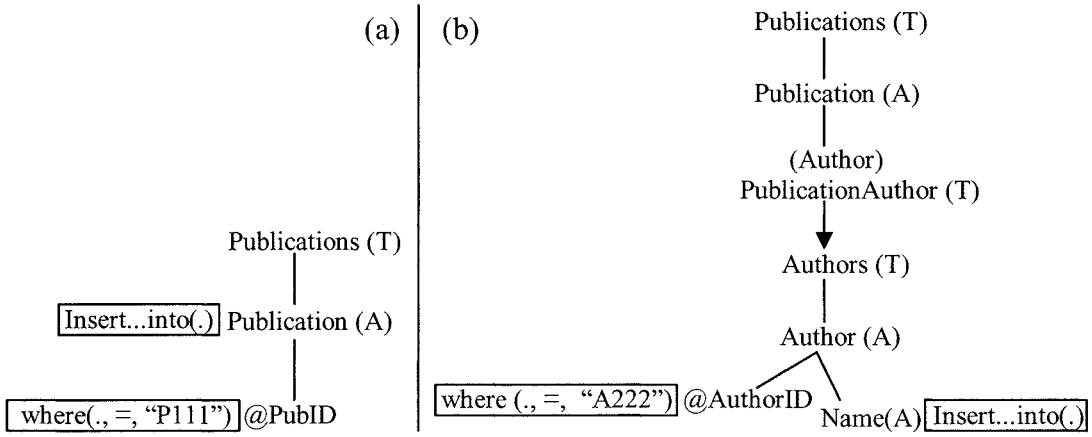

Figure B.2 Trees of (a) first and (b) second insert clauses after adding conditions

XPath expressions for indicating positions which will be inserted are generated from the trees in Figures B.2 and sent to the parameter 'targetLst'. In Figure B.2(a) root element is Publications while in the Figure B.2(b), root element is Authors. The two XPath expressions generated for the first and second insert clauses are as follows.

targetLst = /Publications/Publication/[@PubID = "P111"]

targetLst = /Authors/Author/[@AuthorID = "A222"]/Name

Example 2: The command below inserts a new reference.

```
For   $r in doc("References.xml")/References
Insert <Reference RefID = "R999" RefType = "bibliography">
         rlink(Publication, //Publication[Title = "XQuery"])
         rlink(Publication, //Publication[Title = "XML"])
       </ Reference>
Into $r
```

There is only one insert clause; thus only one tree is created as follows.

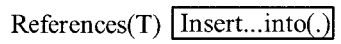

Figure B.3 Tree for the insert clause

From Figure B.3, the target for inserting Reference (converted to ADT without sibling) is References, the root element; thus nothing is returned from ORDB; thereby no receiver. Thus XPath expression for indicating positions which will be inserted is generated from the tree in Figure B.3 and sent to the parameter 'targetLst' as follows: targetLst = /References

Logical statements

This section shows logical statements for four cases of updating ORDB whose structure is converted from an element or attribute of XML documents. Other cases are not shown since the concepts of the logical statements for these cases are the same as or similar to the ones presented here.

I) Element having sibling is converted to ADT

After updating an element converted to an ADT, the data values in XML documents are equivalent to the data values in the database. Logical statements to express this are as follows.

Definition:

$E_2 = \{o_1, o_2\}$ where o is a simple element/attribute.

$E_1 = \{E_2?, p_1, p_k\}$ where p_1 is a simple element/attribute and p_k is ID of E_1.

If $E_1 \rightarrow$ R and $E_2 \rightarrow$ ADT

Then R = {ADT, p_1, p_k}, ADT = $\{o_1, o_2\}$

If Instance of $E_2 = \{e_1\}$ and

$e_1 = E_2\{(o_1, v_{11}), (o_2, v_{21})\}$ where v is value and

If Instance of $E_1 = \{\tilde{e}_1, \tilde{e}_2\}$ and

$\tilde{e}_1 = E_1\{e_1, (p_1, v_{p11}), (p_k, v_{pk1})\}$, $\tilde{e}_2 = E_1\{(p_1, v_{p12}), (p_k, v_{pk2})\}$ and

If instance of ADT = $\{r_1\}$ and instance of R = $\{t_1, t_2\}$

Then $r_1 = ADT(v_{11}, v_{21})$,

$t_1 = \{r_1, v_{p11}, v_{pk1}\} = \{ADT(v_{11}, v_{21}), v_{p11}, v_{pk1}\}$, $t_2 = \{null, v_{p12}, v_{pk2}\}$

Thus the initial states of data value on the XML side and the ORDB side are equivalent

XML side		**ORDB side**
$\tilde{e}_1 = E_1\{e_1, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$t_1 = \{r_1, v_{p11}, v_{pk1}\}$
$e_1 = E_2\{(o_1, v_{11}), (o_2, v_{21})\}$	$\equiv$	$r_1 = ADT(v_{11}, v_{21})$
$\tilde{e}_2 = E_1\{(p_1, v_{p12}), (p_k, v_{pk2})\}$	$\equiv$	$t_2 = \{null, v_{p12}, v_{pk2}\}$

A) Logical statement for insert operation on XML:

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{12}), (o_2, v_{22})\}, E_1\{p_1 = v_{p12}\}]$

Define: $e_2 = E_2\{(o_1, v_{12}), (o_2, v_{22})\}$

From update-rule 1: if an insert is performed on a node converted to an ADT, the insert operation is converted to an update operation in the database context.

t_2 (converted from $\tilde{e}_2$), an instance of R where the primary key is v_{pk2}, contains $p_1 = v_{p12}$; thus ADT in t_2 is updated with the value of e_2 as follows:

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{12}), (o_2, v_{22})\}, E_1\{p_1 = v_{p12}\}] \rightarrow$

ORDB: $\texttt{Upd}[R, ADT = ADT(v_{12}, v_{22}), p_1 = v_{p12}] \Rightarrow$
$t_2 = \{ADT(v_{12}, v_{22}), v_{p12}, v_{pk2}\}$

From return-rule 1: If ADTs are updated, the values for the primary key of updated rows are returned.

$\tilde{e}_2$ (an instance of E_1) contains $p_k = v_{pk2}$; thus e_2 is inserted into $\tilde{e}_2$ as follows:

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{12}), (o_2, v_{22})\}, E_1\{p_1 = v_{p12}\}] \rightarrow$

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{12}), (o_2, v_{22})\}, E_1\{p_k = v_{pk2}\}] = \texttt{Ins}[E_1, e_2, E_1\{p_k = v_{pk2}\}] \Rightarrow$
$\tilde{e}_2 = E_1\{e_2, (p_1, v_{p12}), (p_k, v_{pk2})\}$

Thus: After insertion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$\tilde{e}_2 = E_1\{e_2, (p_1, v_{p12}), (p_k, v_{pk2})\}$	$\equiv$	$t_2 = \{ADT(v_{12}, v_{22}), v_{p12}, v_{pk2}\}$
$e_2 = E_2\{(o_1, v_{12}), (o_2, v_{22})\}$	$\equiv$	$ADT(v_{12}, v_{22})$

B) Logical statement for replace operation on XML:

XML: Upd[E_2, $E_2\{(o_1, v_{12}), (o_2,v_{22})\}$, $E_1\{p_1 = v_{p11}\}$]

From update-rule 1: if a replace is performed on a node converted to an ADT, the replace operation is converted to an update operation in the database context.

t_1 (converted from $\tilde{e}_1$), an instance of R where the primary key is v_{pk1}, contains $p_1 = v_{p11}$; thus r_1 (converted from e_1), an instance of ADT in t_1 is updated as follows:

XML: Upd[E_2, $E_2\{(o_1, v_{12}), (o_2,v_{22})\}$, $E_1\{p_1 = v_{p11}\}$] $\rightarrow$
ORDB: Upd[R, ADT = ADT(v_{12}, v_{22}), $p_1 = v_{p11}$] $\Rightarrow$
r_1 = ADT(v_{12}, v_{22}) and t_1 = {ADT(v_{12}, v_{22}), v_{p11}, v_{pk1}}

From return-rule 1: If ADTs are updated, the values for the primary key of updated rows are returned.

$\tilde{e}_1$ (an instance of E_1) contains $p_k = v_{pk1}$; thus e_1 (an instance of E_2) in $\tilde{e}_1$ is updated as follows:

XML: Upd[E_2, $E_2\{(o_1, v_{12}), (o_2,v_{22})\}$, $E_1\{p_1 = v_{p11}\}$] $\rightarrow$
XML: Upd[E_2, $E_2\{(o_1, v_{12}), (o_2,v_{22})\}$, $E_1\{p_k = v_{pk1}\}$] $\Rightarrow$
$e_1 = E_2\{(o_1, v_{12}), (o_2,v_{22})\}$

Thus: After replacement, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$e_1 = E_2\{(o_1, v_{12}), (o_2,v_{22})\}$	$\equiv$	r_1 = ADT(v_{12}, v_{22})
$\tilde{e}_1 = E_1\{e_1, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	t_1 = {ADT(v_{12}, v_{22}), v_{p11}, v_{pk1}}

C) Logical statement for delete operation on XML:

XML: Del[E_2, $E_1\{p_1 = v_{p11}\}$]

From update-rule 1: if a delete is performed on a node converted to an ADT, the delete operation is converted to an update operation with null in the database context.

t_1 (converted from $\tilde{e}_1$), an instance of R where the primary key is v_{pk1}, contains $p_1 = v_{p11}$; thus r_1 (converted from e_1), an instance of ADT in t_1 is updated with null as follows:

XML: Del[E_2, $E_1\{p_1 = v_{p11}\}$] $\rightarrow$ **ORDB:** Upd[R, ADT = null, $p_1 = v_{p11}$] $\Rightarrow$
t_1 = {null, v_{p11}, v_{pk1}}

From return-rule 1: If ADTs are updated, the values for the primary key of updated rows are returned

$\tilde{e}_1$ (an instance of E_1) contains $p_k = v_{pk1}$; thus e_1 (an instance of E_2) in $\tilde{e}_1$ is deleted as follows:

XML: $\texttt{Del}[E_2, E_1\{p_1 = v_{p11}\}] \rightarrow$ **XML:** $\texttt{Del}[E_2, E_1\{p_k = v_{pk1}\}] \Rightarrow$

$$\tilde{e}_1 = E_1\{(p_1, v_{p11}), (p_k, v_{pk1})\}$$

Thus: After deletion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$\tilde{e}_1 = E_1\{(p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$t_1 = \{\text{null}, v_{p11}, v_{pk1}\}$

II) Element is converted to nested table

After updating an element converted to a nested table (NT), the data values in XML documents are equivalent to the data values in the database. Logical statements to express this are as follows.

Definition:

$E_2 = \{o_1, o_2\}$ where o is a simple element/attribute.

$E_1 = \{E_2^*, p_1, p_k\}$ where p_1 is a simple element/attribute and p_k is ID of E_1.

If $E_1 \rightarrow$ R and $E_2 \rightarrow$ NT

Then R = {NT, p_1, p_k}, NT = {o_1, o_2}

If Instance of E_2= {e_1, e_2, e_3} and

$e_1 = E_2\{(o_1, v_{11}), (o_2, v_{21})\}$, $e_2 = E_2\{(o_1, v_{12}), (o_2, v_{22})\}$ and

$e_3 = E_2\{(o_1, v_{13}), (o_2, v_{23})\}$ where v is value and

If Instance of $E_1 = \{\tilde{e}_1, \tilde{e}_2\}$ and

$\tilde{e}_1 = E_1\{e_1, e_3, (p_1, v_{p11}), (p_k, v_{pk1})\}$, $\tilde{e}_2 = E_1\{e_2, (p_1, v_{p12}), (p_k, v_{pk2})\}$ and

If instance of NT = {r_1, r_2, r_3} and instance of R = {t_1, t_2}

Then $r_1 = \text{NT}\{(v_{11}, v_{21})\}$, $r_2 = \text{NT}\{(v_{12}, v_{22})\}$, $r_3 = \text{NT}\{(v_{13}, v_{23})\}$,

$t_1 = \{r_1, r_3, v_{p11}, v_{pk1}\} = \{\text{NT}\{(v_{11}, v_{21}), (v_{13}, v_{23})\}, v_{p11}, v_{pk1}\}$,

$t_2 = \{r_2, v_{p12}, v_{pk2}\} = \{\text{NT}\{(v_{12}, v_{22})\}, v_{p12}, v_{pk2}\}$

Thus the initial states of data value on the XML side and the ORDB side are equivalent

XML side		ORDB side
$e_1 = E_2\{(o_1, v_{11}), (o_2, v_{21})\}$	≡	$r_1 = NT\{(v_{11}, v_{21})\}$
$e_2 = E_2\{(o_1, v_{12}), (o_2, v_{22})\}$	≡	$r_2 = NT\{(v_{12}, v_{22})\}$
$e_3 = E_2\{(o_1, v_{13}), (o_2, v_{23})\}$	≡	$r_3 = NT\{(v_{13}, v_{23})\}$
$\tilde{e}_1 = E_1\{e_1, e_3, (p_1, v_{p11}), (p_k, v_{pk1})\}$	≡	$t_1 = \{NT\{(v_{11}, v_{21}), (v_{13}, v_{23})\}, v_{p11}, v_{pk1}\}$
$\tilde{e}_2 = E_1\{e_2, (p_1, v_{p12}), (p_k, v_{pk2})\}$	≡	$t_2 = \{NT\{(v_{12}, v_{22})\}, v_{p12}, v_{pk2}\}$

A) Logical statement for insert operation on XML:

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{14}), (o_2, v_{24})\}, E_1\{p_1 = v_{p12}\}]$

Define: $e_4 = E_2\{(o_1, v_{14}), (o_2, v_{24})\}$

From update-rule 2: if an insert is performed on a node converted to an NT, the insert operation is an insert operation on the NT in the database context.

t_2 (converted from $\tilde{e}_2$), an instance of R where the primary key is v_{pk2}, contains $p_1 = v_{p12}$; thus the new data of NT is inserted into t_2 as follows:

XML: $\texttt{Ins}[E1, E_2\{(o_1, v_{14}), (o_2, v_{24})\}, E_1\{p_1 = v_{p12}\}] \rightarrow$

ORDB: $\texttt{Ins}[\texttt{Sel}[NT(R)\ p_1 = v_{p12}], \{v_{14}, v_{24}\}] \Rightarrow$

$t_2 = \{NT\{(v_{12}, v_{22}), (v_{14}, v_{24})\}, v_{p12}, v_{pk2}\}$

From return-rule 2: If nested tables are inserted, the values for the primary key of rows containing the nested tables are returned.

$\tilde{e}_2$, an instance of E_1 contains $p_k = v_{pk2}$; thus the new data is inserted into $\tilde{e}_2$ as follows:

XML: $\texttt{Ins}[E1, E_2\{(o_1, v_{14}), (o_2, v_{24})\}, E_1\{p_1 = v_{p12}\}] \rightarrow$

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{14}), (o_2, v_{24})\}, E_1\{p_k = v_{pk2}\}] = \texttt{Ins}[E1, e_4, E_1\{p_k = v_{pk2}\}] \Rightarrow$

$\tilde{e}_2 = E_1\{e_2, e_4, (p_1, v_{p12}), (p_k, v_{pk2})\}$

Thus: After insertion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		ORDB side
$\tilde{e}_2 = E_1\{e_2, e_4, (p_1, v_{p12}), (p_k, v_{pk2})\}$	≡	$t_2 = \{NT\{(v_{12}, v_{22}), (v_{14}, v_{24})\}, v_{p12}, v_{pk2}\}$
$e_4 = E_2\{(o_1, v_{14}), (o_2, v_{24})\}$	≡	$NT\{(v_{14}, v_{24})\}$

B) Logical statement for replace operation on XML:

XML: Upd[E_2, $E_2\{(o_1, v_{14}), (o_2,v_{24})\}$, $\{E_1\{p_1 = v_{p11}\}, E_2\{o_2 = v_{21}\}\}$]

From update-rule 2: if a replace is performed on a node converted to an NT, the replace operation is an update operation on the NT in the database context.

t_1 (converted from $\tilde{e}_1$) an instance of R where the primary key is v_{pk1}, contains $p_1 = v_{p11}$ and the first instance in NT which is r_1 (converted from e_1, an instance of E_2) of t_1 contains $o_2 = v_{21}$; thus this instance is updated as follows:

XML: Upd[E_2, $E_2\{(o_1, v_{14}), (o_2,v_{24})\}$, $\{E_1\{p_1 = v_{p11}\}, E_2\{o_2 = v_{21}\}\}$] $\rightarrow$
ORDB: Upd[Sel[NT(R)$p_1 = v_{p11}$], $\{NT.o_1 = v_{14}, NT.o_2 = v_{24}\}$, $NT.o_2 = v_{21}$] $\Rightarrow$
$t_1 = \{NT\{(v_{14}, v_{24}), (v_{13}, v_{23})\}, v_{p11}, v_{pk1}\}$

From return-rule 2: If nested tables are updated, the values for the primary key of rows containing the nested tables are returned.

$\tilde{e}_1$ (an instance of E_1) contains $p_k = v_{pk1}$ and e_1 (an instance of E_2) in $\tilde{e}_1$ contains $o_2 = v_{21}$; thus e_1 in $\tilde{e}_1$ is updated as follows:

XML: Upd[E_2, $E_2\{(o_1, v_{14}), (o_2,v_{24})\}$, $\{E_1\{p_1 = v_{p11}\}, E_2\{o_2 = v_{21}\}\}$] $\rightarrow$
XML: Upd[E_2, $E_2\{(o_1, v_{14}), (o_2,v_{24})\}$, $\{E_1\{p_k = v_{pk1}\}, E_2\{o_2 = v_{21}\}\}$] $\Rightarrow$
$e_1 = E_2\{(o_1, v_{14}), (o_2,v_{24})\}$, $\tilde{e}_1 = E_1\{e_1, e_3, (p_1, v_{p11}), (p_k, v_{pk1})\}$

Thus: After replacement, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$\tilde{e}_1 = E_1\{e_1, e_3, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$t_1 = \{NT\{(v_{14}, v_{24}), (v_{13}, v_{23})\}, v_{p11}, v_{pk1}\}$
$e_1 = E_2\{(o_1, v_{14}), (o_2,v_{24})\}$	$\equiv$	$NT\{(v_{14}, v_{24})\}$

C) Logical statement for delete operation on XML:

XML: Del[E_2, $\{E_1\{p_1 = v_{p11}\}, E_2\{o_2 = v_{21}\}\}$]

From update-rule 2: if a delete is performed on a node converted to an NT, the delete operation is a delete operation on the NT in the database context.

t_1 (converted from $\tilde{e}_1$), an instance of R where the primary key is v_{pk1}, contains $p_1 = v_{p11}$ and the first instance in NT which is r_1 (converted from e_1, an instance of E_2) of t_1 contains $o_2 = v_{21}$; thus this instance is deleted as follows:

XML: Del[E_2, $\{E_1\{p_1 = v_{p11}\}, E_2\{o_2 = v_{21}\}\}$] $\rightarrow$
ORDB: Del[Sel[NT(R)$p_1 = v_{p11}$], $NT.o_2 = v_{21}$] $\Rightarrow$
$t_1 = \{\{NT(v_{13}, v_{23})\}, v_{p11}, v_{pk1}\}$

From return-rule 2: If nested tables are deleted, the values for the primary key of rows containing the nested tables are returned.

$\tilde{e}_1$ (an instance of E_1) contains $p_k = v_{pk1}$ and e_1 (an instance of E_2) in $\tilde{e}_1$ contains $o_2 = v_{21}$; thus e_1 in $\tilde{e}_1$ is deleted as follows:

XML: $\texttt{Del}[E_2, \{E_1\{p_1 = v_{p11}\}, E_2\{o_2 = v_{21}\}\}] \rightarrow$
XML: $\texttt{Del}[E_2, \{E_1\{p_k = v_{pk1}\}, E_2\{o_2 = v_{21}\}\}] \Rightarrow$
$\tilde{e}_1 = E_1\{e_3, (p_1, v_{p11}), (p_k, v_{pk1})\}$

Thus: After deletion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$\tilde{e}_1 = E_1\{e_3, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$t_1 = \{NT\{(v_{13}, v_{23})\}, v_{p11}, v_{pk1}\}$

III) Element without sibling is converted to ADT and its parent is the root

After updating an element converted to an ADT whose parent is the root element converted to a table, the data values in XML documents are equivalent to the data values in the database. Logical statements to express this are as follows.

Definition:

$rt = \{E^*\}$, $E = \{o_1, o_2, o_k\}$
where rt is the root element, o is a simple element/attribute, o_k is ID.

If $rt \rightarrow R$ and $E \rightarrow ADT$
Then $R = \{ADT\}$, $ADT = \{o_1, o_2, o_k\}$
If Instance of $E = \{e_1, e_2\}$ and $e_1 = E\{(o_1, v_{11}), (o_2, v_{21}), (o_k, v_{k1})\}$ and
$e_2 = E\{(o_1, v_{12}), (o_2, v_{22}), (o_k, v_{k2})\}$ where v is value and
If Instance of $rt = \{\tilde{e}_1\}$, $\tilde{e}_1 = rt\{e_1, e_2\}$
Then Instance of $rt = \{rt\{e_1, e_2\}\}$
If instance of $ADT = \{r_1, r_2\}$ and instance of $R = \{t_1, t_2\}$
Then $r_1 = ADT(v_{11}, v_{21}, v_{k1})$, $r_2 = ADT(v_{12}, v_{22}, v_{k2})$,
$t_1 = \{r_1\} = \{ADT(v_{11}, v_{21}, v_{k1})\}$, $t_2 = \{r_2\} = \{ADT(v_{12}, v_{22}, v_{k2})\}$
Thus: the data value on the XML side is equivalent to the data value on the ORDB side

XML side		ORDB side
$\{rt\{e_1, e_2\}\}$	$\equiv$	$\{\{ADT(v_{11}, v_{21}, v_{k1})\}, \{ADT(v_{12}, v_{22},$
$rt\{e_1\}$	$\equiv$	$\{ADT(v_{11}, v_{21}, v_{k1})\} = \{r_1\} = t_1$
$rt\{e_2\}$	$\equiv$	$\{ADT(v_{12}, v_{22}, v_{k2})\} = \{r_2\} = t_2$
$e_1 = E\{(o_1, v_{11}), (o_2,v_{21}), (o_k,v_{k1})\}$	$\equiv$	$r_1 = ADT(v_{11}, v_{21}, v_{k1})$
$e_2 = E\{(o_1, v_{12}), (o_2,v_{22}), (o_k,v_{k2})\}$	$\equiv$	$r_2 = ADT(v_{12}, v_{22}, v_{k2})$

A) Logical statements for insert operation on XML:

XML: $\texttt{Ins}[rt, E\{(o_1, v_{13}), (o_2,v_{23}), (o_k,v_{k3})\}]$

Define: $e_3 = E\{(o_1, v_{13}), (o_2,v_{23}), (o_k,v_{k3})\}$

From update-rule 3: if an insert is performed on a node converted to an ADT without sibling and its parent is converted to a table, the insert operation is the insert of a row into the table in the database context.

The new data is inserted into R converted from rt as follows:

XML: $\texttt{Ins}[rt, E\{(o_1, v_{13}), (o_2,v_{23}), (o_k,v_{k3})\}] \rightarrow$

ORDB: $\texttt{Ins}[R, \{ADT(v_{13}, v_{23}, v_{k3})\}] \Rightarrow$

Instance of R = $\{\{ADT(v_{11}, v_{21}, v_{k1})\}, \{ADT(v_{12}, v_{22}, v_{k2})\}, \{ADT(v_{13}, v_{23}, v_{k3})\}\}$

From return-rule 5: insert ADT without sibling and its parent is the root element, nothing is returned.

The new data is inserted into rt as follows:

XML: $\texttt{Ins}[rt, E\{(o_1, v_{13}), (o_2,v_{23}), (o_k,v_{k3})\}] \rightarrow$

XML: $\texttt{Ins}[rt, E\{(o_1, v_{13}), (o_2,v_{23}), (o_k,v_{k3})\}] = \texttt{Ins}[rt, e_3] \Rightarrow$

Instance of rt = $\{rt\{e_1, e_2, e_3\}\}$

Thus: After insertion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		ORDB side
$\{rt\{e_1, e_2, e_3\}\}$	$\equiv$	$\{\{ADT(v_{11}, v_{21}, v_{k1})\}, \{ADT(v_{12}, v_{22}, v_{k2})\}, \{ADT(v_{13}, v_{23}, v_{k3})\}\}$

B) Logical statement for replace operation on XML:

XML: $\texttt{Upd}[E, E\{(o_1, v_{13}), (o_2, v_{23})\}, E\{(o_1 = v_{12})\}]$

From update-rule 3: if a replace is performed on a node converted to an ADT without sibling and its parent is converted to a table, the replace operation is an update operation on the row containing the ADT in the database context.

r_2, an instance of ADT where the primary key is v_{k2}, converted from e_2 contains $o_1 = v_{12}$; thus r_2 is updated as follows:

XML: Upd[E, E{(o_1, v_{13}), (o_2, v_{23})}, E{($o_1 = v_{12}$)}] $\rightarrow$
ORDB: Upd[R, {ADT.$o_1 = v_{13}$, ADT.$o_2 = v_{23}$}, ADT.$o_1 = v_{12}$] $\Rightarrow$
$r_2 = ADT(v_{13}, v_{23}, v_{k2})$

From return-rule 3: update ADT without sibling and its parent is the root element, the value for the primary key of updated row is returned

e_2 (an instance of E) contains $o_k = v_{k2}$; thus e_2 is updated as follows:

XML: Upd[E, E{(o_1, v_{13}), (o_2, v_{23})}, E{($o_1 = v_{12}$)}] $\rightarrow$
XML: Upd[E, E{(o_1, v_{13}), (o_2, v_{23})}, E{($o_k = v_{k2}$)}] $\Rightarrow$
$e_2 = E\{(o_1, v_{13}), (o_2, v_{23}), (o_k, v_{k2})\}$

Thus: After replacement, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$e_2 = E\{(o_1, v_{13}), (o_2, v_{23}), (o_k, v_{k2})\}$	$\equiv$	$r_2 = ADT(v_{13}, v_{23}, v_{k2})$

C) Logical statement for delete operation on XML:

XML: Del[E, E{($o_1 = v_{12}$)}]

From update-rule 3: if a delete is performed on a node converted to an ADT without sibling and its parent is converted to a table, the delete operation is a delete operation on the row containing the ADT in the database context.

t_2, an instance of R where the primary key is v_{k2}, converted from rt{e_2}contains ADT.$o_1 = v_{12}$; thus t_2 is deleted from R as follows:

XML: Del[E, E{($o_1 = v_{12}$)}] $\rightarrow$ **ORDB:** Del[R, ADT.$o_1 = v_{12}$)] $\Rightarrow$
Instance of R = {{ADT(v_{11}, v_{21}, v_{k1})}}

From return-rule 3: delete ADT without sibling and its parent is the root element, the value for the primary key of deleted row is returned.

e_2 (an instance of E) contains $o_k = v_{k2}$; thus e_2 is deleted from rt as follows:

XML: Del[E, E{($o_1 = v_{12}$)}] $\rightarrow$ **XML:** Del[E, E{($o_k = v_{k2}$)}] $\Rightarrow$
Instance of rt = {rt{e_1}}

Thus: After deletion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		ORDB side
$\{rt\{e_1\}\}$	$\equiv$	$\{\{ADT(v_{11}, v_{21}, v_{k1})\}\}$

IV) Element is converted to table and its ancestor is converted to table

After updating an element converted to a table and its ancestor is converted to a table, the data values in XML documents are equivalent to the data values in the database. Logical statements to express this are as follows.

Definition:

$E_2 = \{o_1, o_k\}$ where o_1 is a simple element/attribute, o_k is ID of E_2.

$E_1 = \{E_2, p_1, p_k\}$ where p_1 is a simple element/attribute and p_k is ID of E_1.

If $E_1 \rightarrow R_1$ and $E_2 \rightarrow R_2$

Then $R_1 = \{p_1, p_k\}$, $R_2 = \{o_1, o_k, p_k\}$

If Instance of $E_2 = \{e_1, e_2\}$ and

$e_1 = E_2\{(o_1, v_{11}), (o_k, v_{k1})\}$, $e_2 = E_2\{(o_1, v_{12}), (o_k, v_{k2})\}$ where v is value and

If Instance of $E_1 = \{\tilde{e}_1\}$ and

$\tilde{e}_1 = E_1\{e_1, e_2, (p_1, v_{p11}), (p_k, v_{pk1})\}$ and

If instance of $R_2 = \{r_1, r_2\}$ and instance of $R_1 = \{t_1\}$

Then $r_1 = \{v_{11}, v_{k1}, v_{pk1}\}$, $r_2 = \{v_{12}, v_{k2}, v_{pk1}\}$, $t_1 = \{v_{p11}, v_{pk1}\}$

Thus the initial states of data value on the XML side and the ORDB side are equivalent

XML side		ORDB side
$\tilde{e}_1 = E_1\{e_1, e_2, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$\{v_{11}, v_{k1}, v_{pk1}\}, \{v_{12}, v_{k2}, v_{pk1}\}, \{v_{p11}, v_{pk1}\}$
$E_1\{e_1, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$\{v_{11}, v_{k1}, v_{pk1}\}, \{v_{p11}, v_{pk1}\}$
$E_1\{e_2, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$\{v_{12}, v_{k2}, v_{pk1}\}, \{v_{p11}, v_{pk1}\}$

A) Logical statement for insert operation on XML:

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{13}), (o_k, v_{k3})\}, E_1\{p_1 = v_{p11}\}]$

Define: $e_3 = E_2\{(o_1, v_{13}), (o_k, v_{k3})\}$

From update-rule 3: if an insert is performed on a node converted to a table, the insert operation is an insert operation on the table in the database context.

Since PK of R_1 (converted from E_1) is copied to R_2 (converted from E_2) as FK to keep parent-child relationship, to insert new data to R_2, PK of R_1 must be selected and inserted into R_2 as FK. The value of e_3 is inserted into R_2 as follows:

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{13}), (o_k, v_{k3})\}, E_1\{p_1 = v_{p11}\}] \rightarrow$
ORDB: $\texttt{Ins}[R_2, \{v_{13}, v_{k3}, \texttt{Sel}[p_k(R_1)\ p_1 = v_{p11}]\}] \Rightarrow$
$\{v_{11}, v_{k1}, v_{pk1}\}, \{v_{12}, v_{k2}, v_{pk1}\}, \{v_{13}, v_{k3}, v_{pk1}\}, \{v_{p11}, v_{pk1}\}$

From return-rule 4: If new rows are inserted into a table which is not the root element, the values of foreign key linking to its ancestor are returned.

$\tilde{e}_1$ (an instance of E_1) contains $p_k = v_{pk1}$; thus e_3 is inserted into $\tilde{e}_1$ as follows:

XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{13}), (o_k, v_{k3})\}, E_1\{p_1 = v_{p11}\}] \rightarrow$
XML: $\texttt{Ins}[E_1, E_2\{(o_1, v_{13}), (o_k, v_{k3})\}, E_1\{p_k = v_{pk1}\}] = \texttt{Ins}[E_1, e_3, E_1\{p_k = v_{pk1}\}] \Rightarrow$
$\tilde{e}_1 = E_1\{e_1, e_2, e_3, (p_{11}, v_{p11}), (p_{k1}, v_{pk1})\}$

Thus: After insertion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$E_1\{e_1, e_2, e_3, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$\{v_{11}, v_{k1}, v_{pk1}\}, \{v_{12}, v_{k2}, v_{pk1}\}, \{v_{13}, v_{k3}, v_{pk1}\}, \{v_{p11}, v_{pk1}\}$

B) Logical statement for replace operation on XML:

XML: $\texttt{Upd}[E_2, E_2\{(o_1, v_{13}), E_2\{o_1 = v_{11}\}]$

From update-rule 3: if a replace is performed on a node converted to a table, the replace operation is an update operation on the table in the database context.

r_1, an instance of R_2 where the primary key is v_{k1}, converted from e_1 is the row containing $o_1 = v_{11}$; thus this row is updated as follows:

XML: $\texttt{Upd}[E_2, E_2\{(o_1, v_{13})\}, E_2\{o_1 = v_{11}\}] \rightarrow$
ORDB: $\texttt{Upd}[R_2, \{o_1 = v_{13}\}, \{o_1 = v_{11}\}] \Rightarrow r_1 = \{v_{13}, v_{k1}, v_{pk1}\}$

From return-rule 3: Update table, the value for the primary key of updated row is returned

e_1 (an instance of E_2) contains $o_k = v_{k1}$; thus e_1 is updated as follows:

XML: Upd[E_2, $E_2\{(o_1, v_{13})\}$, $E_2\{o_1 = v_{11}\}$] $\rightarrow$
XML: Upd[E_2, $E_2\{(o_1, v_{13})\}$, $E_2\{o_k = v_{k1}\}$] $\Rightarrow$ $e_1 = E_2\{(o_1, v_{13}), (o_k, v_{k1})\}$

Thus:

After replacement, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$E_1\{e_1, e_2, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$\{v_{13}, v_{k1}, v_{pk1}\}$, $\{v_{12}, v_{k2}, v_{pk1}\}$, $\{v_{p11}, v_{pk1}\}$

C) Logical statement for delete operation on XML:

XML: Del[E_2, $E_2\{o_1 = v_{11}\}$]

From update-rule 3: if a delete is performed on a node converted to a table, the delete operation is a delete operation on the table in the database context.

r_1, an instance of R_2 where the primary key is v_{k1}, converted from e_1 is the row containing $o_1 = v_{11}$; thus this row is deleted as follows:

XML: Del[E_2, $E_2\{o_1 = v_{11}\}$] $\rightarrow$ **ORDB:** Del[R_2, $o_1 = v_{11}$] $\Rightarrow$
$\{v_{12}, v_{k2}, v_{pk1}\}$, $\{v_{p11}, v_{pk1}\}$

From return-rule 3: Delete table, the primary key of deleted row is returned

Since e_1 (an instance of E_2) contains $o_k = v_{k1}$, e_1 is deleted from $\tilde{e}_1$ as follows:

XML: Del[E_2, $E_2\{o_1 = v_{11}\}$] $\rightarrow$ **XML:** Del[E_2, $E_2\{ o_k = v_{k1}\}$] $\Rightarrow$
$\tilde{e}_1 = E_1\{e_2, (p_1, v_{p11}), (p_k, v_{pk1})\}$

Thus: After deletion, the data value on the XML side is equivalent to the data value on the ORDB side

XML side		**ORDB side**
$E_1\{e_2, (p_1, v_{p11}), (p_k, v_{pk1})\}$	$\equiv$	$\{v_{12}, v_{k2}, v_{pk1}\}$, $\{v_{p11}, v_{pk1}\}$

Appendix C: Data-centric and document-centric functionalities

This appendix describes seventeen features of the data-centric and document-centric functionalities that are used in our experimental study.

Data-centric functionalities

The data-centric capability includes value-based join operation, aggregation and other relational features as follows.

- *Exact Match (C1)*

  This feature is used to test the capability of the system to handle simple string lookups with a fully specified path. The update can be a shallow update that matches a predicate at a low level of the tree, or deep update that matches predicate at a high level of the tree. For example, the command below replaces the type of publication authored by the author whose last name is 'Alismun' with the value 'journal'.

  ```
  For $p in doc("Library.xml")/Library/Publication,
      $n in $p/Author/Name
  Where $n/LName = "Alismun"
  Replace $p@PubType with @PubType = "journal"
  ```

- *Update a particular document without joins between documents (C2)*

  The standard SQL does not allow joins between tables in update commands. As in SQL, existing XML update languages do not allow joins between documents; thus this feature can be found for every command without joins between documents.

- *Change selectivity (C3)*

  Selectivity is used to control the amount of data which will be updated; that is it allows a range of values of selected attributes to be varied. For example, the command below replaces the type of publication authored between years 2000 and 2005 with the value 'article'.

  For $p in doc("Library.xml")/Library/Publication
  where $p/Year >= "2000"
  and $p/Year <= "2005"
  Replace $p@PubType with @PubType = "article"

- *Allow condition on text (C4)*

  Text searching is an essential feature of XML query languages since most of the content of XML documents is text. Text searching is used to test the information retrieval capabilities of the system. Testing is involved in two cases: uni-gram search where the condition contains only one particular word, and n-gram search where the condition contains multiple words. For example, in a uni-gram search, the first command below deletes the addresses of publishers whose names contain a string 'Sams'. For an n-gram search, the second command below deletes the addresses of publishers whose names contain a string 'Sams Publishing' and a string 'LTD.'.

  Uni-gram search:

  For $i in doc("Library.xml")/Library/Publication/Publisher
  where contains($i/PName, "Sams")
  delete $i/Address

  N-gram search:

  For $i in doc("Library.xml")/Library/Publication/Publisher
  where contains($i/PName, "Sams Publishing")
  and contains($i/PName, "LTD.")
  delete $i/Address

- *Support aggregation (C5)*

  Due to the structure of XML, it is possible that conditions for querying or updating data will be based on aggregation. This feature is used to test the ability of the system to group data and compute data in each group. For example, the command below inserts the type of publication authored by at least 2 authors with the value 'article'.

  ```
  For $p in doc("Library.xml")/Library/Publication
  Let $a := $p/Author
  Where count($a) >= 2
  Insert  @PubType = "article" into $p
  ```

- *Support quantifiers (C6)*

  An existential quantifier tests whether at least one item satisfies a condition while a universal quantifier tests whether every item satisfies a condition; thus their meaning can be translated into an aggregate function count() with the conditions of the quantifier; thus this feature tests the ability to group data and compute data in each group based on the conditions. For example, in the case of the existential quantifier, the first command below replaces the type of publication authored by at least one author whose last name is 'Dobson' with the value 'book'. For the use of a universal quantifier, the second command below replaces the type of publication authored by every author whose email is 'xml@sams.com' with the value 'book'.

  Existential quantifier:

  ```
  For $p in doc("Publications.xml")/Publications/Publication
  Where some $a in $p/Author~>/Authors/Author
      satisfies ($a/Name/LName = "Dobson")
  Replace $p@PubType with @PubType = "book"
  ```

  Universal quantifier:

  ```
  For $p in doc("Publications.xml")/Publications/Publication
  Where all $a in $p/Author~>/Authors/Author
      satisfies ($a/Email = "xml@sams.com")
  Replace $p@PubType with @PubType = "book"
  ```

- *Joins based on values (C7)*

  This feature is used to test the ability of the system to handle large intermediate results. This type of join involves comparing values at two different nodes that do not need to be related structurally. For example, the command below inserts the type of publication published in the same year as the setup year of the publisher of the publication with the value 'journal'.

  ```
  For $p in doc("Library.xml")/Library/Publication,
      $i in $p/Publisher
  Where $p/Year = $i/SetupYear
  insert  @PubType = "journal" into $p
  ```

- *Joins based on pointer (C8)*

  This join involves comparing values of two different nodes that are related structurally by ID-values. This join differs from the joins based on value since a tunable DBMS can exploit the structure to optimise XML processing. For example, the command below replaces the email of the author who is the contact author for his publication whose title is 'XML' with the value 'XML@hotmail.com'.

  ```
  For $p in doc("Library.xml")/Library/Publication,
      $a in $p/Author
  Where $p@ContactAuthor = $a@AuthorID
  and $p/Title = "XML"
  Replace $a/Email with <Email>XML@hotmail.com</Email>
  ```

- *Casting (C9)*

  Strings are a generic data type in XML data. Sometimes, however the strings in the commands may need to be cast to another data type that carries more semantics. This feature is quite challenging to DBMS since only the string data type can be defined in a DTD. For example, the command below inserts the shipping cost of a publication whose cost is double that of the special cost.

  ```
  For $p in doc("Publications.xml")/Publications/Publication
  Where $p/Cost = $p/SpecialCost * 2
  insert  <ShippingCost>50</ShippingCost> into $p
  ```

Document-centric functionalities

Document-centric capability includes ordering element, navigational capability and other features as follows.

- *Update multiple documents (C10)*

  In contrast to existing XML update languages, our XML update language supports the update of multiple documents and joins of documents in an update command; thus this feature is tested to measure its performance. For example, the command below replaces the email of authors and the type of publication whose publisher has ID of 'I555' with the values 'sams2@hotmail.com' and 'article' respectively.

  ```
  For $p in doc("Publications.xml")/Publications/Publication,
      $a in $p/Author ~>/Authors/Author,
      $i in $p/Publisher ~>/Publishers/Publisher
  Where $i@PID = "I555"
  Replace $p@PubType with @PubType = "article" ,
  Replace $a/Email with  <Email>sams2@hotmail.com</Email>
  ```

  Note: In the case of a single document, the command for this feature will be translated into a comparable command by using a path traversal instead.

- *Navigation by reference traversal (C11)*

  References allow richer relationships than simple hierarchical structures since they define horizontal traversals while hierarchical structures are navigated in a vertical way. For example, the command below replaces the title of publication whose title is 'Java 2' referred by the publication whose title is 'Java' with the value 'Java programming'.

  ```
  For $p in  doc("Library.xml")/Library/Publication
      $r in $p/Reference/@RefPub→Publication
  Where $p/Title = "Java"
  And $r/Tile = 'Java2'
  replace $r/Title with <Title>Java programming</Title>
  ```

- *Handling missing elements (C12)*

  Irregularity of schema is unavoidable in XML documents; hence missing elements always occur. We use this feature to test how well the system can cope with missing elements, especially the elements which are defined as optional elements in an XML schema. For example, the command below generates a middle name for authors who do not have middle names with the value 'no middle name'.

```
For $p in doc("Library.xml")/Library/Publication,
    $a in $p/Author
where empty($a/Name/MName)
insert <MName>no middle name</MName> into $a/Name
```

- *Element ordering (C13)*

  This feature is used to test how well the system can deal with constraints based on order in update commands since the cost of reordering existing elements in documents is expensive when new elements are inserted or existing ones are deleted. For example, the command below inserts middle name before last name of author whose first name is 'Johnny' and whose last name is 'Dobson'.

```
For $a in doc("Library.xml")/Library/Publication/Author
where $a/Name/FName = "Johnny"
and $a/Name/LName = "Dobson"
insert <MName>Murial</MName>
into $a/Name
before $a/Name/LName
```

- *Regular path expression (C14)*

  Regular path expressions are a major feature of most XML query languages since XML documents may have long paths; thus multiple element names in the paths may be unknown. The regular path expressions are used to test how efficiently the system can optimise path expressions and prune traversals of irrelevant path of the tree. For example, the command below inserts new telephone details for the author whose last name is 'Dobson'.

```
For $a in doc("Library.xml")//Author
insert <Telephone>
            <Location>School</Location>
            <TelNo>01912739145</TelNo>
    </Telephone>
into $a
where $a/Name/LName = "Dobson"
```

- *Mix between data-centric and document-centric (C15)*

  This feature is used to test the ability of the system to handle conditions based on both data-centric and document-centric features. For example, the command below deletes authors who do not have a middle name and the title of their publication is 'XML and Java'.

```
For $p in doc("Library.xml")/Library/Publication,
    $a in $p/Author
where empty($a/Name/MName)
and $p/Title = "XML and Java"
delete $a
```

- *Hierarchical and sequence update (C16)*

  This feature is used to test the updates of several parts of documents in one update command by using a sequence of update clauses. For example, the command below inserts the type of publication, the email and the middle name of authors where publication has title 'XML'.

```
For $p in doc("Library.xml")/Library/Publication,
    $a in $p/Author,
    $n in $a/Name
Where $p/Title = "XML"
insert @PubType = "journal" into $p ,
insert  <Email>sams@hotmail.com</Email> into $a,
insert <MName>Anonymous</MName> into $n
```

- *Recursion (C17)*

  This feature is used to test how efficiently the system can traverse XML structure recursively when the structure is repeated several times.

  For example, the recursive function below inserts type into publications which are both direct and indirect references of the publication whose title is 'Java 2' with value 'journal'.

```
define function allRef($pub as element()*)
{
  For $rp in $pub/Reference/@RefPub → Publication
   insert @PubType = "journal" into $rp
   allRef($rp)
}
For $p in doc("Library.xml")/Library/Publication
Where $p/Title = "Java 2"
allRef($p)
```

Appendix D: The experimental results

This appendix shows the rest of the results from the experimental study conducted in Chapter 7. Firstly, it shows the performance for seventeen update-features of the XML update language. Secondly, it shows translating XML update commands into SQL according to update-features. Thirdly, it shows translating update commands according to updating different types of ORDB structure. Finally, it shows the detail-table expressing the ratio of execution time in nxd, sxd and lxd.

Performance of seventeen update-features

This section shows the performance for seventeen update-features in cold cache for only 5 MB data size since the data having size 10-40 MB is shown in Chapter 7 already. It also shows the performance in warm and hot caches.

I) Performance of seventeen update-features in cold cache for 5 MB data size

Table D.1 Replace time of three databases in cold cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.13	3.75	4.18	2.13	3.76	4.13	2.19	3.77	3.97	2.19	3.78	3.80
C3	2.75	3.51	4.19	2.77	3.52	4.13	2.79	3.53	3.96	2.79	3.54	3.80
C4	3.37	3.55	3.25	3.43	3.56	3.24	3.05	3.56	3.07	2.09	3.57	2.90
C5	1.98	3.59	3.27	1.99	3.60	3.07	1.99	3.61	3.08	2.01	3.62	2.90
C6	2.29	3.62	3.53	2.30	3.63	3.03	3.03	3.63	2.88	3.05	3.66	2.67
C7	4.13	3.69	3.36	4.22	3.70	3.22	4.24	3.71	3.07	4.26	3.72	2.92
C8	2.19	3.29	3.01	2.21	3.30	2.81	2.29	3.31	2.68	2.30	3.33	2.46
C9	2.13	2.92	3.05	2.37	2.93	2.87	2.40	2.94	2.81	2.43	2.95	2.58
C10	2.55	3.74	3.37	2.57	3.75	3.10	2.59	3.05	2.98	2.61	3.06	2.81
C12	2.13	3.55	3.18	2.16	3.56	3.01	2.18	3.58	2.87	2.20	3.60	2.71
C14	2.24	3.50	3.18	3.54	3.50	2.96	6.16	3.51	2.80	10.90	3.53	2.64
C15	3.23	3.73	3.17	3.25	3.74	3.03	3.29	3.75	2.86	3.29	3.76	2.71
C16	2.78	3.01	3.24	2.79	3.02	3.11	2.80	3.11	2.96	2.83	3.13	2.83
C17	-	4.43	3.54	-	4.45	3.52	-	4.45	3.34	-	4.46	3.19
AVG	**2.64**	**3.57**	**3.39**	**2.68**	**3.58**	**3.23**	**2.74**	**3.54**	**3.09**	**2.67**	**3.55**	**2.92**
STDEV	0.65	0.37	0.37	0.66	0.37	0.42	0.62	0.39	0.40	0.64	0.39	0.41
%RSD	**24.73**	**10.44**	**10.85**	**24.57**	**10.46**	**12.93**	**22.70**	**11.01**	**13.00**	**23.97**	**10.90**	**14.03**

Table D.2 Delete time of three databases in cold cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.12	3.75	4.19	2.13	3.77	4.13	2.15	3.78	3.97	2.18	3.79	3.79
C3	2.31	3.52	4.20	2.32	3.51	4.14	2.35	3.54	3.97	2.36	3.51	3.80
C4	1.98	3.56	3.26	2.00	3.55	3.24	2.02	3.57	3.08	2.03	3.57	2.90
C5	2.01	3.57	3.26	2.08	3.59	3.07	2.10	3.60	3.07	2.12	3.60	2.90
C6	2.31	3.61	3.17	2.35	3.61	3.03	2.37	3.63	2.88	2.38	3.66	2.68
C7	2.23	3.68	3.35	2.25	3.70	3.15	2.27	3.72	3.06	2.28	3.72	2.92
C8	2.22	3.29	3.00	2.23	3.29	2.79	2.26	3.31	2.68	2.27	3.32	2.45
C9	2.16	2.92	3.06	2.18	2.93	2.87	2.19	2.95	2.82	2.21	2.94	2.56
C10	2.14	3.73	3.35	2.16	3.75	3.07	2.18	3.06	2.96	2.19	3.08	2.81
C12	2.13	3.45	3.18	2.15	3.57	3.06	2.16	3.58	2.86	2.18	3.58	2.71
C14	2.22	3.50	3.17	3.49	3.51	2.95	5.92	3.54	2.82	10.78	3.53	2.66
C15	2.11	3.73	3.18	2.13	3.74	3.01	2.24	3.71	2.87	2.25	3.73	2.72
C16	2.56	3.00	3.23	2.58	3.00	3.09	2.59	3.11	2.96	2.62	3.14	2.82
C17	-	4.41	3.52	-	4.36	3.50	-	4.44	3.33	-	4.46	3.51
AVG	**2.19**	**3.55**	**3.37**	**2.21**	**3.57**	**3.22**	**2.24**	**3.54**	**3.09**	**2.25**	**3.55**	**2.94**
STDEV	0.15	0.37	0.37	0.15	0.36	0.42	0.15	0.38	0.40	0.15	0.38	0.44
%RSD	**7.03**	**10.41**	**11.11**	**6.86**	**10.12**	**13.09**	**6.70**	**10.81**	**12.97**	**6.62**	**10.84**	**14.79**

Table D.3 Insert time of three databases in cold cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.14	3.75	4.18	2.15	3.76	4.14	2.16	3.78	3.89	2.18	3.78	3.86
C3	2.33	3.37	4.18	2.34	3.38	3.46	2.35	3.39	3.92	2.36	3.39	3.86
C4	1.99	3.56	3.27	2.01	3.57	3.25	2.03	3.58	3.14	2.04	3.59	2.89
C5	2.04	3.59	3.28	2.05	3.60	3.08	2.06	3.60	3.09	2.08	3.61	2.90
C6	2.32	3.61	3.17	2.34	3.61	3.03	2.35	3.61	2.92	2.37	3.63	2.68
C7	2.24	3.68	3.47	2.26	3.69	3.22	2.26	3.69	3.08	2.28	3.71	2.92
C8	2.24	3.29	3.03	2.24	3.29	2.81	2.26	3.31	2.67	2.27	3.31	2.46
C9	2.18	2.90	3.07	2.19	2.92	2.89	2.20	2.92	2.82	2.22	2.93	2.57
C10	2.16	3.76	3.47	2.18	3.75	3.11	2.19	3.76	2.98	2.21	3.76	2.79
C12	2.15	3.56	3.06	2.16	3.57	3.01	2.17	3.58	2.86	2.18	3.58	2.71
C131	2.18	3.70	3.34	2.19	3.71	3.07	2.19	3.72	2.92	2.22	3.73	2.69
C132	2.18	3.70	3.33	2.19	3.71	3.06	2.20	3.72	2.92	2.21	3.74	2.68
C14	2.24	3.50	3.17	3.52	3.51	2.94	6.02	3.53	2.80	10.82	3.54	2.65
C15	2.13	3.73	3.14	2.13	3.74	3.04	2.15	3.76	2.85	2.16	3.77	2.71
C16	2.58	3.74	3.17	2.59	3.00	3.11	2.59	3.00	2.97	2.62	3.01	2.82
C17	-	4.42	3.54	-	4.44	3.50	-	4.44	3.33	-	4.46	3.17
AVG	**2.21**	**3.63**	**3.37**	**2.22**	**3.59**	**3.17**	**2.23**	**3.60**	**3.07**	**2.25**	**3.51**	**2.90**
STDEV	0.15	0.33	0.16	0.15	0.38	0.17	0.15	0.38	0.16	0.15	0.38	0.18
%RSD	**6.78**	**9.22**	**4.80**	**6.69**	**10.49**	**5.30**	**6.54**	**10.45**	**5.36**	**6.61**	**10.77**	**6.05**

II) Performance of seventeen update-features in warm cache

Table D.4 Replace time of three databases in warm cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	4.79	1.51	1.40	1.57	1.53	1.38	1.61	1.54	1.37	1.64	1.55	1.36
C3	1.93	1.71	1.59	1.99	1.72	1.58	2.02	1.76	1.57	2.06	1.78	1.57
C4	3.25	1.44	0.91	3.43	1.46	0.90	3.56	1.50	0.89	3.58	1.51	0.88
C5	1.42	1.44	0.91	1.49	1.45	0.90	1.50	1.45	0.89	1.51	1.47	0.88
C6	1.73	1.45	1.13	1.77	1.47	1.05	1.80	1.48	1.01	1.80	1.49	0.97
C7	3.42	1.55	0.98	3.49	1.57	0.98	3.53	1.59	0.96	3.59	1.60	0.95
C8	1.62	1.47	0.93	1.66	1.50	0.92	1.69	1.53	0.92	1.70	1.53	0.90
C9	1.53	1.41	0.96	1.57	1.43	0.95	1.59	1.44	0.94	1.60	1.46	0.93
C10	1.99	1.50	1.05	1.99	1.52	1.04	2.03	1.53	1.03	2.06	1.55	1.03
C12	1.52	1.44	0.95	1.59	1.45	0.95	1.60	1.46	0.93	1.60	1.48	0.92
C14	2.11	1.43	0.90	3.26	1.47	0.94	5.47	1.50	0.93	9.67	1.51	0.91
C15	2.83	1.43	0.90	2.93	1.45	0.88	3.06	1.47	0.88	3.28	1.48	0.87
C16	2.05	1.70	1.20	2.10	1.73	1.18	2.29	1.77	1.17	2.32	1.78	1.16
C17	-	2.62	1.34	-	2.65	1.33	-	2.66	1.34	-	2.68	1.32
AVG	**2.34**	**1.57**	**1.08**	**2.13**	**1.59**	**1.07**	**2.19**	**1.61**	**1.06**	**2.23**	**1.62**	**1.05**
STDEV	1.03	0.33	0.22	0.73	0.33	0.22	0.76	0.33	0.22	0.80	0.33	0.22
%RSD	**44.03**	**20.81**	**20.47**	**34.40**	**20.57**	**20.35**	**34.90**	**20.30**	**20.62**	**35.81**	**20.34**	**20.95**

Table D.5 Replace time of three databases in warm cache (10 MB)

Cmd.	Redundant Records											
	20			40			80			160		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.96	2.05	1.69	1.97	2.07	1.48	2.00	2.14	1.34	2.02	2.23	1.33
C3	2.69	3.46	2.08	2.73	5.00	1.76	2.77	8.36	1.66	2.81	15.79	1.86
C4	4.94	1.81	0.96	5.01	1.83	0.95	5.13	1.92	0.94	5.21	1.97	0.89
C5	1.72	1.94	1.49	1.73	1.97	1.42	1.75	2.04	1.33	1.77	2.10	1.35
C6	2.21	1.82	1.30	2.21	1.87	1.24	2.23	1.92	1.11	2.24	1.99	1.04
C7	5.71	1.94	1.12	5.80	1.99	1.09	5.92	2.05	1.06	5.99	2.10	1.00
C8	2.01	1.91	1.05	2.01	1.95	1.04	2.04	2.05	1.00	2.06	2.10	0.97
C9	2.08	1.86	1.02	2.11	1.89	0.97	2.12	1.97	0.94	2.12	2.01	0.90
C10	2.71	1.88	1.16	2.72	1.94	1.12	2.77	2.17	1.07	2.80	2.24	1.03
C12	1.97	1.84	1.12	1.98	1.87	1.04	2.01	1.95	1.01	2.03	2.00	1.00
C14	7.14	1.81	0.98	9.63	1.86	0.95	14.64	1.95	0.94	24.32	2.00	0.92
C15	4.54	1.83	0.97	4.61	1.87	0.96	4.75	2.00	0.93	4.81	2.08	0.91
C16	3.14	2.72	1.55	3.18	2.90	1.49	3.25	3.11	1.44	3.28	3.64	1.33
C17	-	3.06	1.41	-	3.13	1.27	-	3.24	1.16	-	3.46	1.30
AVG	**2.97**	**2.04**	**1.28**	**3.00**	**2.09**	**1.20**	**3.06**	**2.19**	**1.14**	**3.10**	**2.30**	**1.13**
STDEV	1.35	0.39	0.33	1.37	0.42	0.25	1.41	0.44	0.22	1.44	0.56	0.27
%RSD	**45.27**	**19.20**	**25.89**	**45.70**	**20.09**	**21.11**	**46.22**	**20.27**	**19.59**	**46.48**	**24.44**	**24.09**

Table D.6 Replace time of three databases in warm cache (20 MB)

Cmd.	Redundant Records											
	40			80			160			320		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.78	2.66	2.02	2.85	2.95	1.92	2.87	3.23	1.73	2.98	3.47	1.84
C3	4.25	8.81	2.65	4.31	12.72	2.43	4.38	20.42	2.41	4.47	36.57	2.27
C4	8.78	2.75	1.11	8.79	2.97	1.08	9.04	3.10	1.07	9.34	3.29	1.01
C5	2.36	3.00	2.25	2.40	3.20	2.13	2.42	3.33	1.91	2.48	3.78	1.71
C6	3.26	2.74	1.53	3.29	2.93	1.47	3.33	3.16	1.38	3.43	3.36	1.25
C7	10.35	3.11	1.39	10.36	3.33	1.35	10.52	3.50	1.30	10.95	4.02	1.22
C8	2.88	3.06	1.27	2.93	3.33	1.23	2.96	3.56	1.20	3.02	3.84	1.13
C9	3.03	2.94	1.16	3.05	3.24	1.12	3.07	3.42	1.08	3.16	3.82	1.04
C10	4.25	3.21	1.41	4.33	3.45	1.37	4.37	3.67	1.31	4.47	3.96	1.24
C12	2.82	2.66	1.43	2.88	2.77	1.28	2.90	2.94	1.22	3.00	3.15	1.17
C14	25.50	2.78	1.16	42.32	3.04	1.10	76.54	3.17	1.09	145.95	3.61	1.02
C15	8.02	2.86	1.10	8.02	2.95	1.09	8.21	3.18	1.07	8.61	3.87	1.01
C16	5.25	5.12	2.12	5.24	5.35	2.04	5.36	5.83	1.95	5.58	6.75	1.75
C17	-	6.04	1.59	-	6.49	1.40	-	6.77	1.33	-	7.36	1.28
AVG	**4.84**	**3.30**	**1.58**	**4.87**	**3.54**	**1.50**	**4.95**	**3.76**	**1.43**	**5.12**	**4.18**	**1.35**
STDEV	2.71	1.04	0.49	2.69	1.10	0.44	2.77	1.16	0.41	2.89	1.31	0.39
%RSD	**56.10**	**31.53**	**30.66**	**55.31**	**31.12**	**29.58**	**55.86**	**30.95**	**28.65**	**56.49**	**31.42**	**28.57**

Table D.7 Replace time of three databases in warm cache (40 MB)

Cmd.	Redundant Records											
	80			160			320			640		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	4.55	6.05	3.56	4.58	6.19	3.13	4.68	6.75	2.92	4.75	7.17	2.67
C3	7.46	16.99	4.35	7.48	28.32	3.82	7.75	50.64	3.50	7.78	94.40	2.86
C4	16.47	5.19	1.47	16.53	5.31	1.44	16.97	5.75	1.37	17.19	6.08	1.23
C5	3.71	5.75	4.48	3.79	6.17	3.91	3.80	6.80	3.76	3.84	7.40	3.03
C6	5.49	5.22	2.44	5.49	5.42	2.22	5.63	5.77	2.02	5.66	6.39	1.86
C7	19.51	6.00	2.07	19.54	6.38	2.02	19.98	7.09	1.94	20.20	7.56	1.65
C8	4.73	5.81	1.77	4.76	6.13	1.75	4.81	6.41	1.66	4.85	6.98	1.45
C9	4.97	5.59	1.54	5.02	6.00	1.49	5.05	6.42	1.37	5.16	6.87	1.26
C10	7.48	5.80	2.00	7.47	6.03	1.96	7.69	6.36	1.89	7.78	6.73	1.67
C12	4.64	5.18	1.93	4.64	5.41	1.69	4.75	5.68	1.64	4.80	6.52	1.47
C14	84.54	5.60	1.46	150.22	5.76	1.43	287.05	6.07	1.38	548.67	7.40	1.26
C15	14.90	5.28	1.45	14.97	5.47	1.42	15.39	5.90	1.36	15.71	6.74	1.22
C16	9.34	9.84	3.40	9.41	10.79	3.22	9.67	11.22	3.12	9.71	12.72	2.74
C17	-	10.85	1.82	-	11.63	1.66	-	12.15	1.61	-	14.32	1.50
AVG	**8.60**	**6.32**	**2.41**	**8.64**	**6.67**	**2.22**	**8.85**	**7.10**	**2.11**	**8.95**	**7.91**	**1.85**
STDEV	5.37	1.82	1.08	5.38	2.05	0.90	5.53	2.09	0.84	5.61	2.54	0.67
%RSD	**62.46**	**28.84**	**44.78**	**62.28**	**30.77**	**40.57**	**62.53**	**29.37**	**40.05**	**62.70**	**32.16**	**36.36**

Table D.8 Delete time of three databases in warm cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.54	1.51	1.38	1.55	1.52	1.37	1.59	1.54	1.36	1.60	1.54	1.35
C3	1.48	1.72	1.60	1.49	1.71	1.59	1.53	1.77	1.58	1.54	1.77	1.57
C4	1.53	1.43	0.91	1.54	1.46	0.90	1.56	1.50	0.89	1.60	1.52	0.88
C5	1.42	1.42	0.91	1.43	1.46	0.90	1.48	1.46	0.89	1.53	1.46	0.89
C6	1.68	1.46	1.13	1.69	1.48	1.07	1.72	1.49	1.04	1.73	1.50	1.00
C7	1.59	1.57	0.98	1.66	1.57	0.96	1.68	1.58	0.97	1.71	1.60	0.97
C8	1.58	1.49	0.94	1.60	1.50	0.94	1.63	1.52	0.92	1.66	1.53	0.91
C9	1.52	1.42	0.96	1.54	1.42	0.95	1.55	1.44	0.94	1.59	1.46	0.93
C10	1.49	1.49	1.06	1.52	1.51	1.05	1.53	1.53	1.04	1.55	1.53	1.04
C12	1.49	1.44	0.94	1.52	1.45	0.95	1.54	1.47	0.93	1.55	1.47	0.91
C14	2.10	1.43	0.92	3.26	1.48	0.90	5.38	1.48	0.90	5.39	1.50	0.89
C15	1.46	1.42	0.90	1.46	1.45	0.88	1.48	1.47	0.88	1.49	1.47	0.87
C16	1.98	1.69	1.18	2.00	1.75	1.18	2.12	1.77	1.17	2.18	1.77	1.16
C17	-	2.61	1.32	-	2.65	1.31	-	2.65	1.32	-	2.68	1.31
AVG	**1.56**	**1.57**	**1.08**	**1.58**	**1.59**	**1.07**	**1.62**	**1.61**	**1.06**	**1.64**	**1.62**	**1.05**
STDEV	0.15	0.32	0.22	0.15	0.33	0.22	0.18	0.33	0.22	0.18	0.33	0.22
%RSD	**9.46**	**20.64**	**20.10**	**9.51**	**20.66**	**20.26**	**10.86**	**20.25**	**20.61**	**11.20**	**20.34**	**20.71**

Table D.9 Delete time of three databases in warm cache (10 MB)

Cmd.	Redundant Records											
	20			40			80			160		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.95	2.03	1.81	1.96	2.05	1.47	1.98	2.08	1.39	1.99	2.20	1.30
C3	1.91	3.44	2.03	1.93	4.94	1.85	1.93	8.11	1.73	1.94	14.81	1.71
C4	1.86	1.82	1.00	1.86	1.83	0.96	1.87	1.87	0.94	1.89	1.98	0.92
C5	1.72	1.94	1.47	1.72	1.96	1.38	1.75	2.01	1.39	1.75	2.08	1.36
C6	2.19	1.82	1.24	2.21	1.86	1.17	2.22	1.94	1.10	2.23	2.00	1.03
C7	1.99	1.93	1.20	2.01	1.98	1.10	2.02	2.04	1.08	2.04	2.08	1.03
C8	1.99	1.88	1.13	2.00	1.98	1.05	2.01	2.04	1.01	2.03	2.10	0.96
C9	2.08	1.83	1.09	2.10	1.86	1.03	2.11	1.94	0.95	2.13	2.01	0.94
C10	1.90	1.87	1.20	1.91	1.95	1.11	1.92	2.18	1.08	1.93	2.23	1.03
C12	1.95	1.84	1.10	1.96	1.85	1.05	1.97	1.94	1.02	1.99	1.98	0.99
C14	7.17	1.82	1.04	9.72	1.86	1.01	14.95	1.92	0.94	24.60	1.97	0.87
C15	1.79	1.82	1.03	1.80	1.84	0.97	1.81	1.97	0.94	1.83	2.03	0.88
C16	3.15	2.75	1.54	3.16	2.90	1.45	3.23	3.19	1.39	3.25	3.42	1.29
C17	-	3.08	1.47	-	3.14	1.30	-	3.27	1.19	-	3.47	1.14
AVG	**2.04**	**2.03**	**1.31**	**2.05**	**2.08**	**1.21**	**2.07**	**2.18**	**1.15**	**2.08**	**2.27**	**1.10**
STDEV	0.37	0.40	0.31	0.37	0.43	0.25	0.39	0.47	0.24	0.39	0.53	0.23
%RSD	**18.18**	**19.75**	**23.86**	**18.11**	**20.47**	**21.04**	**18.70**	**21.53**	**20.51**	**18.66**	**23.12**	**21.26**

Table D.10 Delete time of three databases in warm cache (20 MB)

Cmd.	Redundant Records											
	40			80			160			320		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.85	2.79	2.27	2.85	3.05	2.01	2.87	3.26	1.88	2.92	3.52	1.79
C3	2.71	8.70	2.64	2.73	12.69	2.36	2.73	19.91	2.16	2.81	36.62	2.01
C4	2.63	2.62	1.23	2.64	2.97	1.17	2.65	3.07	1.12	2.70	3.28	1.06
C5	2.38	2.94	2.22	2.39	3.20	2.15	2.41	3.30	1.91	2.48	3.56	1.78
C6	3.27	2.62	1.52	3.29	2.95	1.46	3.30	3.14	1.38	3.40	3.39	1.23
C7	2.93	3.01	1.53	2.93	3.26	1.44	2.97	3.40	1.32	3.02	3.84	1.25
C8	2.90	3.07	1.40	2.91	3.54	1.34	2.93	3.70	1.25	2.99	3.98	1.22
C9	3.03	2.88	1.19	3.05	3.18	1.14	3.06	3.43	1.13	3.13	3.57	1.07
C10	2.71	3.23	1.60	2.86	3.59	1.47	2.91	3.82	1.34	2.95	4.06	1.28
C12	2.85	2.55	1.42	2.85	2.73	1.32	2.86	2.91	1.22	2.95	3.13	1.19
C14	25.48	2.79	1.14	42.43	3.03	1.10	76.62	3.09	1.04	146.56	3.37	0.99
C15	2.49	2.83	1.23	2.50	2.91	1.17	2.50	3.20	1.07	2.57	3.56	1.00
C16	5.22	5.14	2.18	5.26	5.31	2.05	5.28	5.63	2.01	5.52	6.68	1.83
C17	-	5.88	1.60	-	6.40	1.44	-	7.00	1.36	-	7.41	1.31
AVG	**3.00**	**3.26**	**1.66**	**3.02**	**3.55**	**1.54**	**3.04**	**3.77**	**1.44**	**3.12**	**4.10**	**1.36**
STDEV	0.74	1.03	0.48	0.74	1.07	0.42	0.74	1.19	0.38	0.80	1.34	0.34
%RSD	**24.66**	**31.55**	**28.71**	**24.60**	**30.29**	**27.07**	**24.49**	**31.63**	**26.34**	**25.50**	**32.67**	**25.24**

Table D.11 Delete time of three databases in warm cache (40 MB)

Cmd.	Redundant Records											
	80			160			320			640		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	4.59	6.00	3.91	4.57	6.15	3.15	4.64	7.36	2.95	4.66	7.64	2.69
C3	4.27	17.09	4.66	4.27	28.84	3.76	4.36	50.81	3.69	4.46	94.02	3.56
C4	4.16	5.23	1.47	4.17	5.51	1.38	4.22	5.70	1.37	4.26	6.18	1.26
C5	3.71	5.83	4.65	3.71	6.19	4.04	3.77	6.66	3.69	3.85	7.21	3.00
C6	5.56	5.33	2.49	5.54	5.40	2.22	5.60	5.90	2.03	5.68	6.36	1.83
C7	4.78	6.34	2.06	4.77	6.63	1.99	4.86	7.22	1.91	4.91	7.97	1.68
C8	4.68	5.94	1.78	4.70	6.06	1.72	4.75	6.50	1.68	4.78	7.23	1.49
C9	4.92	5.76	1.55	4.98	6.33	1.43	5.03	6.15	1.42	5.06	6.72	1.32
C10	4.26	5.90	2.30	4.29	6.10	1.94	4.35	6.48	1.90	4.37	7.09	1.63
C12	4.66	5.15	1.87	4.66	5.40	1.67	4.69	5.71	1.64	4.76	6.17	1.53
C14	86.13	5.61	1.46	151.63	5.81	1.42	287.46	6.07	1.36	549.22	6.81	1.25
C15	3.85	5.37	1.41	3.87	5.64	1.38	3.94	5.88	1.35	3.98	6.43	1.27
C16	9.42	7.37	3.40	9.44	10.77	3.21	9.57	11.21	2.92	9.59	12.70	2.51
C17	-	10.89	1.89	-	11.50	1.68	-	11.94	1.63	-	13.04	1.51
AVG	**4.90**	**6.21**	**2.49**	**4.91**	**6.73**	**2.21**	**4.98**	**7.14**	**2.11**	**5.03**	**7.81**	**1.89**
STDEV	1.51	1.52	1.17	1.51	1.99	0.93	1.53	2.04	0.84	1.52	2.31	0.74
%RSD	**30.72**	**24.46**	**47.03**	**30.69**	**29.64**	**41.93**	**30.65**	**28.63**	**39.97**	**30.21**	**29.60**	**38.95**

Table D.12 Insert time of three databases in warm cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.55	1.53	1.40	1.55	1.54	1.39	1.59	1.53	1.38	1.61	1.55	1.38
C3	1.49	1.70	1.60	1.49	1.73	1.60	1.54	1.75	1.59	1.54	1.77	1.58
C4	1.53	1.44	0.91	1.57	1.44	0.91	1.59	1.49	0.90	1.60	1.50	0.89
C5	1.43	1.44	0.91	1.44	1.43	0.89	1.49	1.46	0.88	1.50	1.47	0.88
C6	1.69	1.46	1.14	1.70	1.46	1.07	1.73	1.48	1.04	1.74	1.50	1.00
C7	1.60	1.55	1.26	1.66	1.57	0.99	1.70	1.59	0.98	1.71	1.59	0.97
C8	1.59	1.47	0.97	1.61	1.51	0.90	1.64	1.53	0.90	1.65	1.54	0.89
C9	1.53	1.41	0.93	1.53	1.42	0.92	1.57	1.45	0.92	1.59	1.46	0.91
C10	1.49	1.48	1.02	1.51	1.51	1.02	1.54	1.53	1.00	1.56	1.54	1.00
C12	1.50	1.44	0.93	1.51	1.45	0.93	1.55	1.47	0.91	1.56	1.47	0.91
C131	1.58	1.45	1.27	1.59	1.45	1.26	1.56	1.47	1.25	1.58	1.48	1.25
C132	1.60	1.45	1.27	1.60	1.46	1.26	1.56	1.47	1.25	1.57	1.49	1.24
C14	2.11	1.44	0.91	3.26	1.45	0.91	5.39	1.50	0.89	5.39	1.53	0.89
C15	1.46	1.43	0.89	1.51	1.44	0.87	1.49	1.46	0.88	1.50	1.48	0.86
C16	1.99	1.16	1.20	1.99	1.72	1.19	2.13	1.74	1.18	2.18	1.78	1.17
C17	-	3.55	1.33	-	2.64	1.33	-	2.68	1.32	-	2.68	1.31
AVG	**1.57**	**1.58**	**1.12**	**1.59**	**1.57**	**1.09**	**1.63**	**1.59**	**1.08**	**1.64**	**1.60**	**1.07**
STDEV	0.15	0.60	0.17	0.14	0.33	0.16	0.17	0.33	0.16	0.18	0.33	0.16
%RSD	**9.29**	**37.68**	**15.07**	**9.01**	**20.91**	**14.81**	**10.63**	**20.86**	**14.79**	**11.17**	**20.66**	**14.86**

Table D.13 Insert time of three databases in warm cache (10 MB)

Cmd.	Redundant Records											
	20			40			80			160		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.97	1.94	1.72	1.97	2.01	1.39	2.00	2.20	1.32	2.02	2.28	1.29
C3	1.91	2.21	1.84	1.93	2.28	1.60	1.97	2.40	1.51	2.00	2.52	1.46
C4	1.88	1.79	0.98	1.91	1.90	0.95	1.96	1.89	0.94	2.01	1.93	0.88
C5	1.72	1.93	1.48	1.74	1.98	1.39	1.76	2.09	1.31	1.77	2.19	1.22
C6	2.20	1.84	1.25	2.21	1.87	1.15	2.23	1.95	1.09	2.25	2.07	1.02
C7	2.02	2.08	1.17	2.06	2.12	1.10	2.10	2.18	1.09	2.17	2.31	1.03
C8	2.01	1.98	1.08	2.02	2.05	1.04	2.04	2.16	0.99	2.06	2.22	0.97
C9	2.09	1.93	1.05	2.10	2.02	0.98	2.11	2.05	0.92	2.12	2.11	0.88
C10	1.92	1.94	1.22	1.93	1.98	1.16	1.96	2.23	1.12	1.99	2.28	1.08
C12	1.96	1.84	1.09	1.98	1.86	1.04	2.00	1.95	1.01	2.02	1.97	0.97
C131	2.02	1.85	1.45	2.02	1.88	1.31	2.07	1.98	1.29	2.10	2.01	1.12
C132	2.01	1.86	1.46	2.02	1.88	1.32	2.08	1.95	1.29	2.10	2.02	1.13
C14	7.37	1.91	1.00	9.95	1.94	0.95	14.88	2.08	0.93	24.75	2.11	0.87
C15	1.82	1.87	0.98	1.86	1.90	1.17	1.89	2.02	0.94	1.96	2.12	0.95
C16	3.14	2.92	1.52	3.19	2.96	1.45	3.26	3.10	1.37	3.27	3.44	1.30
C17	-	3.46	1.48	-	3.56	1.28	-	3.75	1.20	-	4.06	1.14
AVG	**2.05**	**2.08**	**1.30**	**2.07**	**2.13**	**1.21**	**2.11**	**2.24**	**1.15**	**2.14**	**2.34**	**1.08**
STDEV	0.36	0.50	0.21	0.37	0.52	0.16	0.38	0.55	0.16	0.37	0.64	0.13
%RSD	**17.54**	**24.20**	**16.07**	**17.71**	**24.22**	**13.51**	**17.91**	**24.39**	**13.88**	**17.35**	**27.37**	**12.07**

Table D.14 Insert time of three databases in warm cache (20 MB)

Cmd.	Redundant Records											
	40			80			160			320		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.83	2.67	2.07	2.84	3.00	2.02	2.86	3.34	1.98	2.94	3.61	1.84
C3	2.74	3.50	2.40	2.75	3.62	2.20	2.78	3.96	2.21	2.93	4.33	2.02
C4	2.69	2.76	1.15	2.70	2.98	1.12	2.79	3.08	1.09	2.94	3.28	1.04
C5	2.38	2.95	2.25	2.40	3.19	2.03	2.43	3.29	1.96	2.52	3.54	1.81
C6	3.29	2.68	1.53	3.31	2.97	1.47	3.33	3.15	1.39	3.45	3.37	1.23
C7	2.99	3.15	1.44	3.01	3.32	1.36	3.08	3.42	1.33	3.27	3.77	1.24
C8	2.92	2.97	1.34	2.93	3.36	1.28	2.94	3.63	1.23	3.03	3.83	1.17
C9	3.04	2.90	1.19	3.05	3.19	1.12	3.06	3.41	1.08	3.15	3.62	1.07
C10	2.72	2.99	1.46	2.74	3.22	1.37	2.77	3.46	1.34	2.89	3.86	1.26
C12	2.86	2.59	1.44	2.86	2.82	1.32	2.90	3.00	1.21	3.00	3.14	1.17
C131	2.94	2.95	1.68	2.96	3.08	1.61	2.99	3.34	1.57	3.14	3.48	1.53
C132	2.94	3.05	1.70	2.96	3.08	1.60	2.98	3.36	1.54	3.15	3.49	1.55
C14	25.77	2.80	1.14	42.54	3.04	1.11	76.56	3.08	1.09	146.63	3.37	1.02
C15	2.55	2.88	1.15	2.57	2.81	1.11	2.65	3.20	1.09	2.82	3.58	1.04
C16	5.21	5.02	2.21	5.26	5.23	2.08	5.34	5.54	2.00	5.61	6.49	1.80
C17	-	6.08	1.57	-	6.39	1.42	-	6.74	1.32	-	7.27	1.47
AVG	**3.02**	**3.23**	**1.61**	**3.03**	**3.45**	**1.51**	**3.08**	**3.67**	**1.46**	**3.21**	**3.98**	**1.39**
STDEV	0.72	1.03	0.36	0.73	1.07	0.31	0.74	1.11	0.30	0.78	1.29	0.27
%RSD	**23.94**	**32.06**	**22.26**	**24.14**	**30.91**	**20.81**	**24.02**	**30.36**	**20.76**	**24.25**	**32.35**	**19.63**

Table D.15 Insert time of three databases in warm cache (40 MB)

Cmd.	Redundant Records											
	80			160			320			640		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	4.55	6.01	3.31	4.59	6.40	3.14	4.65	7.17	2.98	4.73	7.79	2.83
C3	4.34	6.51	4.08	4.37	6.78	3.41	4.45	7.42	3.20	4.55	7.57	2.97
C4	4.21	5.24	1.51	4.32	5.40	1.46	4.41	5.73	1.37	4.53	6.15	1.25
C5	3.69	5.80	4.79	3.75	6.19	4.15	3.86	6.77	1.97	3.89	7.28	3.09
C6	5.59	5.24	2.41	5.59	5.40	2.12	5.64	5.90	8.85	5.73	6.39	1.79
C7	4.88	6.33	2.12	4.94	6.77	2.02	5.09	7.25	1.92	5.14	7.67	1.69
C8	4.73	5.97	1.80	4.76	6.50	1.76	4.80	6.70	1.68	4.84	7.35	1.49
C9	4.95	5.94	1.56	4.97	5.87	1.46	5.00	6.13	1.36	5.08	6.61	1.24
C10	4.31	6.02	2.14	4.35	6.26	2.01	4.45	6.99	1.91	4.56	7.24	1.53
C12	4.67	5.18	1.88	4.69	5.37	1.68	4.73	5.73	1.61	4.82	6.21	1.47
C131	4.82	5.15	2.51	4.87	5.25	2.26	4.91	5.38	2.13	5.06	6.26	1.87
C132	4.83	5.17	2.52	4.87	5.24	2.25	4.91	5.38	2.11	5.05	6.29	1.89
C14	86.02	5.63	1.50	151.36	5.95	1.45	287.67	6.13	1.35	550.02	6.86	1.25
C15	3.92	5.28	1.48	4.01	5.53	1.43	4.18	5.90	1.35	4.26	6.50	1.23
C16	9.46	9.57	3.38	9.45	10.70	3.22	9.70	11.06	3.05	9.76	12.64	2.71
C17	-	10.92	1.90	-	11.59	1.66	-	12.05	1.65	-	14.46	1.50
AVG	**4.94**	**6.23**	**2.43**	**4.98**	**6.56**	**2.22**	**5.08**	**6.95**	**2.41**	**5.16**	**7.71**	**1.86**
STDEV	1.49	1.82	0.90	1.47	2.08	0.77	1.51	2.13	1.94	1.50	2.65	0.56
%RSD	**30.22**	**29.29**	**37.11**	**29.49**	**31.72**	**34.73**	**29.69**	**30.67**	**80.50**	**29.14**	**34.34**	**29.91**

III) Performance of seventeen update-features in hot cache

Table D.16 Replace time of three databases in hot cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.55	1.74	0.74	1.57	1.75	0.74	1.62	1.76	0.73	1.64	1.73	0.73
C3	1.93	1.82	0.93	1.98	2.34	0.92	2.04	3.53	0.91	2.05	6.01	0.91
C4	3.25	1.44	0.87	3.42	1.56	0.85	3.54	2.08	0.85	3.58	2.10	0.84
C5	1.42	1.45	1.01	1.49	1.46	1.00	1.49	1.58	0.99	1.51	1.59	0.98
C6	1.72	1.44	0.95	1.77	1.45	0.94	1.80	1.46	0.93	1.78	1.46	0.93
C7	3.42	1.49	0.94	3.50	1.46	0.93	3.52	1.47	0.92	3.58	1.49	0.92
C8	1.63	1.45	0.93	1.66	1.46	0.93	1.68	1.46	0.92	1.69	1.47	0.92
C9	1.52	1.45	0.88	1.58	1.46	0.88	1.59	1.47	0.86	1.58	1.48	0.86
C10	1.98	1.44	0.93	1.99	1.45	0.92	2.02	1.46	0.92	2.12	1.46	0.92
C12	1.52	1.44	0.87	1.59	1.46	0.86	1.60	1.46	0.85	1.58	1.47	0.85
C14	2.11	1.43	0.85	3.25	1.45	0.85	5.47	1.47	0.84	9.66	1.47	0.83
C15	2.82	1.44	0.89	2.93	1.46	0.89	3.06	1.47	0.88	3.28	1.47	0.87
C16	2.03	1.62	1.11	2.10	1.63	1.10	2.29	1.64	1.09	2.32	1.64	1.08
C17	-	2.10	0.95	-	2.11	0.94	-	2.12	0.93	-	2.20	0.93
AVG	**2.07**	**1.53**	**0.92**	**2.13**	**1.55**	**0.91**	**2.19**	**1.61**	**0.90**	**2.23**	**1.62**	**0.90**
STDEV	0.70	0.19	0.08	0.73	0.19	0.08	0.76	0.24	0.08	0.80	0.25	0.08
%RSD	**33.93**	**12.60**	**9.09**	**34.46**	**12.28**	**9.03**	**34.75**	**14.72**	**9.11**	**35.81**	**15.48**	**9.10**

Table D.17 Replace time of three databases in hot cache (10 MB)

Cmd.	Redundant Records											
	20			40			80			160		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.97	1.96	1.05	1.98	2.12	1.04	2.01	2.16	1.01	2.11	2.37	0.98
C3	2.69	4.85	1.11	2.73	6.08	1.11	2.78	8.32	1.08	2.79	12.07	1.03
C4	4.92	1.82	0.97	5.02	1.86	0.96	5.10	1.90	0.94	5.17	1.98	0.91
C5	1.73	1.94	1.16	1.73	2.03	1.14	1.76	2.14	1.09	1.83	2.19	1.02
C6	2.20	1.85	1.05	2.22	1.90	1.04	2.23	2.02	1.02	2.42	2.14	0.98
C7	5.71	2.10	1.10	5.80	2.16	1.08	5.90	2.31	1.06	5.93	2.41	1.01
C8	2.00	2.00	1.04	2.01	2.13	1.04	2.04	2.24	1.01	2.15	2.30	0.97
C9	2.08	1.93	0.99	2.11	1.99	0.97	2.12	2.08	0.95	2.22	2.13	0.92
C10	2.68	2.02	1.12	2.73	2.10	1.10	2.78	2.29	1.08	2.79	2.37	1.04
C12	1.97	1.84	0.97	1.99	1.86	0.96	2.03	2.06	0.96	2.09	2.01	0.92
C14	7.18	1.89	0.96	9.71	1.98	0.96	14.74	2.07	0.94	14.76	2.13	0.92
C15	4.53	1.99	0.97	4.61	1.93	0.96	4.73	2.08	0.93	4.79	2.16	0.90
C16	3.15	2.81	1.50	3.20	3.00	1.45	3.26	3.19	1.39	3.56	3.80	1.29
C17	-	3.52	1.05	-	3.64	1.01	-	4.24	0.97	-	4.63	0.92
AVG	**2.97**	**2.13**	**1.07**	**3.01**	**2.21**	**1.06**	**3.06**	**2.37**	**1.03**	**3.16**	**2.51**	**0.99**
STDEV	1.34	0.49	0.14	1.37	0.52	0.13	1.41	0.64	0.12	1.39	0.79	0.10
%RSD	**45.20**	**22.92**	**12.83**	**45.63**	**23.65**	**12.21**	**45.92**	**27.23**	**11.54**	**44.05**	**31.34**	**10.06**

Table D.18 Replace time of three databases in hot cache (20 MB)

Cmd.	Redundant Records											
	40			80			160			320		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.84	2.61	1.26	2.85	2.87	1.35	2.87	3.12	1.20	2.98	3.48	1.43
C3	4.31	6.40	1.40	4.31	10.30	1.48	4.39	17.87	1.33	4.59	32.32	1.54
C4	8.84	2.73	1.11	8.85	2.96	1.19	8.99	3.07	1.07	9.32	3.26	1.32
C5	2.40	3.00	1.41	2.42	3.17	1.40	2.41	3.30	1.40	2.50	3.51	1.23
C6	3.31	2.72	1.27	3.29	2.86	1.27	3.33	3.08	1.31	3.44	3.29	1.14
C7	10.40	2.87	1.37	10.42	2.96	1.45	10.47	3.07	1.30	10.91	3.31	1.52
C8	2.92	2.86	1.26	2.93	3.14	1.32	2.97	3.34	1.20	3.06	3.57	1.44
C9	3.07	2.74	1.14	3.08	3.00	1.14	3.09	3.19	1.17	3.17	3.41	1.04
C10	4.29	3.08	1.39	4.33	3.28	1.44	4.38	3.41	1.31	4.52	3.67	1.53
C12	2.87	2.69	1.19	2.88	2.70	1.09	2.92	2.90	1.08	3.03	3.15	1.01
C14	25.30	2.60	1.12	42.26	2.76	1.10	77.14	2.88	1.15	145.90	3.28	1.04
C15	8.03	2.76	1.10	8.03	2.76	1.19	8.20	3.08	1.06	8.54	3.34	1.32
C16	5.29	5.05	2.10	5.31	5.15	2.01	5.35	5.38	1.92	5.53	6.34	1.72
C17	-	5.46	1.13	-	5.71	1.22	-	5.97	1.16	-	6.73	1.04
AVG	**4.88**	**3.17**	**1.30**	**4.89**	**3.33**	**1.33**	**4.95**	**3.52**	**1.26**	**5.13**	**3.87**	**1.31**
STDEV	2.71	0.94	0.26	2.71	0.95	0.24	2.75	0.98	0.22	2.86	1.19	0.23
%RSD	**55.51**	**29.73**	**19.68**	**55.42**	**28.58**	**17.72**	**55.51**	**27.68**	**17.15**	**55.80**	**30.81**	**17.57**

Table D.19 Replace time of three databases in hot cache (40 MB)

Cmd.	Redundant Records											
	80			160			320			640		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	4.59	5.92	1.79	4.61	6.11	1.73	4.66	6.79	1.66	4.69	7.35	1.45
C3	7.51	12.43	2.00	7.54	20.16	2.00	7.71	34.57	1.89	7.79	64.09	1.65
C4	16.42	5.19	1.46	16.60	5.38	1.41	16.80	5.75	1.36	16.99	5.91	1.23
C5	3.57	5.88	2.22	3.74	5.78	2.12	3.77	6.47	2.01	3.81	7.03	1.69
C6	5.51	5.16	1.82	5.54	5.32	1.74	5.59	5.69	1.67	5.66	6.34	1.50
C7	19.55	5.45	2.04	19.68	5.76	1.97	19.91	6.13	1.87	19.91	6.49	1.63
C8	4.72	5.57	1.76	4.76	5.78	1.71	4.79	5.98	1.66	4.88	6.84	1.46
C9	5.03	5.16	1.51	5.04	5.52	1.47	5.13	5.73	1.38	5.14	6.34	1.26
C10	7.54	5.58	1.99	7.59	5.67	1.94	7.66	6.11	1.85	7.70	6.46	1.64
C12	4.69	5.18	1.39	4.79	5.40	1.37	4.82	5.69	1.34	4.84	6.15	1.22
C14	85.18	5.12	1.43	150.82	5.43	1.43	286.19	5.67	1.35	546.83	6.30	1.26
C15	14.96	5.16	1.44	15.03	5.32	1.37	15.24	5.80	1.33	15.48	6.19	1.22
C16	9.43	9.43	3.36	9.58	10.25	3.18	9.61	10.62	2.97	9.64	11.99	2.58
C17	-	9.68	1.42	-	10.23	1.38	-	10.99	1.35	-	12.97	1.25
AVG	**8.62**	**6.04**	**1.83**	**8.71**	**6.30**	**1.77**	**8.81**	**6.72**	**1.69**	**8.88**	**7.41**	**1.50**
STDEV	5.38	1.59	0.52	5.41	1.76	0.48	5.48	1.84	0.44	5.51	2.29	0.36
%RSD	**62.39**	**26.28**	**28.37**	**62.15**	**27.95**	**27.34**	**62.18**	**27.41**	**26.10**	**62.08**	**30.89**	**23.85**

Table D.20 Delete time of three databases in hot cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.54	1.75	0.73	1.56	1.75	0.74	1.58	1.76	0.74	1.61	1.77	0.73
C3	1.48	1.81	0.94	1.49	2.35	0.94	1.54	3.53	0.93	1.53	6.03	0.93
C4	1.51	1.57	0.87	1.53	1.57	0.87	1.56	1.58	0.85	1.62	1.59	0.85
C5	1.42	1.45	0.99	1.41	1.46	0.98	1.48	1.46	0.98	1.53	1.48	0.97
C6	1.69	1.45	0.94	1.68	1.44	0.94	1.73	1.46	0.91	1.74	1.46	0.91
C7	1.58	1.46	0.95	1.65	1.47	0.94	1.66	1.45	0.92	1.69	1.47	0.93
C8	1.57	1.44	0.93	1.59	1.46	0.93	1.63	1.46	0.92	1.66	1.47	0.91
C9	1.52	1.45	0.86	1.54	1.46	0.85	1.55	1.47	0.84	1.58	1.48	0.84
C10	1.49	1.44	0.94	1.52	1.45	0.93	1.52	1.45	0.93	1.54	1.45	0.92
C12	1.49	1.45	0.87	1.52	1.47	0.86	1.52	1.48	0.85	1.55	1.49	0.85
C14	2.11	1.43	0.85	3.26	1.44	0.84	5.39	1.45	0.83	5.38	1.45	0.83
C15	1.46	1.44	0.89	1.45	1.45	0.90	1.48	1.47	0.88	1.50	1.47	0.88
C16	1.97	1.62	1.10	1.99	1.63	1.09	2.12	1.63	1.10	2.17	1.64	1.08
C17	-	1.74	0.94	-	2.10	0.94	-	2.11	0.93	-	2.11	0.92
AVG	**1.56**	**1.51**	**0.91**	**1.58**	**1.55**	**0.91**	**1.61**	**1.56**	**0.90**	**1.64**	**1.56**	**0.90**
STDEV	0.15	0.12	0.08	0.15	0.19	0.08	0.17	0.19	0.08	0.18	0.19	0.08
%RSD	**9.42**	**7.67**	**8.97**	**9.46**	**12.25**	**8.79**	**10.84**	**12.22**	**9.11**	**11.09**	**12.15**	**8.89**

Table D.21 Delete time of three databases in hot cache (10 MB)

Cmd.	Redundant Records											
	20			40			80			160		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.95	2.09	1.08	1.96	2.10	1.03	1.98	2.20	1.00	2.04	2.47	0.95
C3	1.90	4.89	1.15	1.92	6.07	1.10	1.93	8.36	1.06	1.98	12.12	1.01
C4	1.85	1.81	0.97	1.88	1.83	0.95	1.87	1.91	0.94	1.92	1.97	0.91
C5	1.71	2.02	1.16	1.72	2.01	1.13	1.74	2.11	1.06	1.77	2.21	1.01
C6	2.19	1.83	1.04	2.20	1.90	1.03	2.22	1.99	1.00	2.51	2.08	0.96
C7	1.99	2.12	1.14	2.01	2.18	1.08	2.03	2.25	1.05	2.16	2.32	0.99
C8	1.98	1.99	1.08	1.99	2.14	1.02	2.01	2.25	1.01	2.12	2.30	0.95
C9	2.07	1.93	1.03	2.08	1.96	1.01	2.10	2.07	0.95	2.24	2.14	0.90
C10	1.90	1.98	1.16	1.91	2.09	1.10	1.92	2.32	1.07	1.96	2.37	1.02
C12	1.95	1.84	0.95	1.96	1.88	0.95	1.97	1.99	0.93	2.06	2.01	0.91
C14	7.18	1.91	1.02	9.78	1.97	1.01	14.98	2.11	0.94	15.03	2.13	0.90
C15	1.79	1.89	1.00	1.79	1.90	0.95	1.80	2.05	0.93	1.83	2.15	0.90
C16	3.14	2.83	1.48	3.16	2.98	1.45	3.24	3.20	1.38	3.48	3.67	1.26
C17	-	3.50	1.04	-	3.61	1.01	-	3.78	0.95	-	4.22	0.91
AVG	**2.04**	**2.13**	**1.09**	**2.05**	**2.20**	**1.06**	**2.07**	**2.32**	**1.02**	**2.17**	**2.46**	**0.97**
STDEV	0.37	0.49	0.13	0.37	0.52	0.13	0.39	0.54	0.12	0.45	0.68	0.10
%RSD	**18.21**	**22.83**	**11.93**	**18.10**	**23.51**	**11.82**	**18.90**	**23.43**	**11.54**	**20.92**	**27.64**	**9.85**

Table D.22 Delete time of three databases in hot cache (20 MB)

Cmd.	Redundant Records											
	40			80			160			320		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.82	2.61	1.36	2.84	2.88	1.30	2.86	3.13	1.20	2.94	3.34	1.14
C3	2.71	6.27	1.49	2.72	10.21	1.44	2.73	17.96	1.32	2.81	31.81	1.23
C4	2.62	2.60	1.50	2.63	2.98	1.47	2.64	3.05	1.09	2.72	3.26	1.03
C5	2.39	2.91	1.40	2.34	3.15	1.39	2.41	3.27	1.34	2.51	3.50	1.22
C6	3.30	2.74	1.26	3.28	2.90	1.26	3.30	3.07	1.23	3.42	3.27	1.13
C7	2.94	2.77	1.76	2.88	2.92	1.71	2.95	2.99	1.29	3.03	3.29	1.34
C8	2.89	2.91	1.36	2.89	3.23	1.26	2.90	3.39	1.20	2.99	3.50	1.20
C9	3.03	2.76	1.14	3.03	2.98	1.12	3.05	3.18	1.17	3.14	3.31	1.03
C10	2.70	3.03	1.48	2.70	3.24	1.42	2.72	3.43	1.31	2.79	3.63	1.25
C12	2.84	2.56	1.18	2.85	2.84	1.12	2.88	2.87	1.14	2.98	3.15	1.03
C14	25.32	2.50	1.12	42.39	2.82	1.10	76.87	2.87	1.19	146.81	3.15	1.03
C15	2.48	2.75	1.49	2.49	2.78	1.45	2.50	3.09	1.06	2.59	3.23	1.01
C16	5.25	4.94	2.09	5.24	5.10	1.99	5.32	5.34	1.91	5.56	6.19	1.71
C17	-	5.40	1.13	-	5.65	1.12	-	5.96	1.18	-	6.48	1.05
AVG	**3.00**	**3.11**	**1.41**	**2.99**	**3.34**	**1.37**	**3.02**	**3.51**	**1.26**	**3.13**	**3.79**	**1.17**
STDEV	0.75	0.93	0.27	0.75	0.92	0.25	0.76	0.97	0.21	0.81	1.14	0.19
%RSD	**25.00**	**29.80**	**18.98**	**25.07**	**27.58**	**18.28**	**25.22**	**27.73**	**16.31**	**25.79**	**30.07**	**16.09**

Table D.23 Delete time of three databases in hot cache (40 MB)

Cmd.	Redundant Records											
	80			160			320			640		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	4.53	5.91	1.84	4.53	6.24	1.78	4.61	6.98	1.66	4.62	7.38	1.45
C3	4.25	12.38	2.00	4.28	20.19	1.95	4.34	33.14	1.88	4.41	63.97	1.66
C4	4.07	5.19	1.44	4.15	5.44	1.38	4.19	5.67	1.35	4.22	6.13	1.23
C5	3.69	5.74	2.21	3.71	5.94	2.13	3.74	6.64	2.01	3.79	7.51	1.69
C6	5.53	5.20	1.81	5.54	5.36	1.74	5.58	5.75	1.66	5.64	6.19	1.51
C7	4.77	5.69	1.97	4.79	5.80	2.00	4.81	6.04	1.88	4.86	6.61	1.64
C8	4.68	5.59	1.74	4.71	5.77	1.71	4.72	6.01	1.65	4.75	6.70	1.46
C9	4.86	5.20	1.46	4.95	5.49	1.46	4.97	5.74	1.39	5.03	6.38	1.28
C10	4.26	5.59	2.03	4.31	5.86	2.03	4.33	6.00	1.92	4.36	6.52	1.64
C12	4.63	5.16	1.41	4.66	5.42	1.36	4.65	5.59	1.34	4.71	6.07	1.22
C14	85.44	5.21	1.42	150.78	5.36	1.42	287.40	5.49	1.35	548.18	6.26	1.25
C15	3.83	5.16	1.39	3.84	5.43	1.36	3.84	5.72	1.34	3.92	5.97	1.22
C16	9.28	9.47	3.36	9.37	10.39	3.18	9.49	10.70	2.98	9.60	12.01	2.47
C17	-	9.75	1.41	-	10.09	1.39	-	10.71	1.37	-	11.85	1.26
AVG	**4.86**	**6.07**	**1.82**	**4.90**	**6.35**	**1.78**	**4.94**	**6.70**	**1.70**	**4.99**	**7.35**	**1.50**
STDEV	1.48	1.59	0.53	1.49	1.75	0.49	1.52	1.83	0.44	1.53	2.08	0.33
%RSD	**30.34**	**26.29**	**28.89**	**30.43**	**27.49**	**27.56**	**30.73**	**27.30**	**26.15**	**30.71**	**28.35**	**22.17**

Table D.24 Insert time of three databases in hot cache (5 MB)

Cmd.	Redundant Records											
	10			20			40			80		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.55	1.76	0.77	1.55	1.77	0.76	1.59	1.77	0.76	1.60	1.78	0.75
C3	1.48	1.73	0.93	1.50	1.74	0.93	1.53	1.74	0.93	1.54	1.72	0.92
C4	1.53	1.44	0.88	1.56	1.45	0.87	1.59	1.46	0.86	1.60	1.46	0.86
C5	1.43	1.45	1.00	1.42	1.45	0.99	1.50	1.46	0.98	1.49	1.47	0.98
C6	1.69	1.44	0.93	1.70	1.44	0.93	1.74	1.46	0.92	1.75	1.47	0.92
C7	1.59	1.49	0.95	1.65	1.48	0.94	1.69	1.48	0.93	1.71	1.50	0.92
C8	1.59	1.45	0.93	1.61	1.46	0.92	1.64	1.47	0.92	1.66	1.47	0.91
C9	1.53	1.46	0.89	1.53	1.47	0.88	1.57	1.47	0.87	1.59	1.48	0.86
C10	1.50	1.44	0.93	1.51	1.45	0.93	1.55	1.47	0.92	1.56	1.46	0.91
C12	1.51	1.44	0.88	1.51	1.45	0.87	1.55	1.46	0.86	1.57	1.47	0.86
C131	1.58	1.46	1.01	1.59	1.47	1.01	1.56	1.47	1.00	1.55	1.48	0.99
C132	1.59	1.47	1.01	1.59	1.47	1.00	1.55	1.49	1.00	1.56	1.50	0.99
C14	2.11	1.43	0.85	3.26	1.43	0.84	5.38	1.44	0.84	5.40	1.44	0.83
C15	1.46	1.44	0.86	1.51	1.45	0.86	1.49	1.46	0.85	1.50	1.47	0.84
C16	1.99	1.62	1.11	1.90	1.62	1.10	2.11	1.63	1.09	2.18	1.65	1.09
C17	-	2.10	0.94	-	2.11	0.94	-	2.12	0.93	-	2.13	0.93
AVG	**1.57**	**1.53**	**0.93**	**1.58**	**1.53**	**0.92**	**1.63**	**1.54**	**0.92**	**1.65**	**1.55**	**0.91**
STDEV	0.14	0.18	0.07	0.12	0.18	0.07	0.17	0.18	0.07	0.19	0.18	0.07
%RSD	**9.21**	**12.06**	**7.68**	**7.69**	**11.96**	**7.69**	**10.44**	**11.93**	**7.76**	**11.31**	**11.92**	**7.91**

Table D.25 Insert time of three databases in hot cache (10 MB)

Cmd.	Redundant Records											
	20			40			80			160		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	1.97	1.91	1.04	1.97	1.94	1.02	1.98	2.07	1.00	1.99	2.15	0.96
C3	1.92	1.89	1.04	1.93	1.96	1.03	1.96	2.06	1.00	1.98	2.14	0.96
C4	1.89	1.80	0.96	1.94	1.80	0.95	1.96	1.87	0.93	2.11	1.93	0.90
C5	1.73	1.91	1.16	1.74	1.92	1.14	1.75	1.98	1.05	1.78	2.03	0.99
C6	2.19	1.82	1.04	2.22	1.86	1.04	2.24	1.89	0.99	2.35	1.97	0.95
C7	2.02	1.90	1.08	2.06	1.95	1.07	2.10	1.99	1.04	1.86	2.04	0.99
C8	2.00	1.89	1.04	2.02	1.97	1.02	2.04	2.03	0.99	2.32	2.07	0.95
C9	2.08	1.83	0.99	2.09	1.84	0.96	2.10	1.95	1.13	2.09	2.00	0.90
C10	1.91	1.86	1.11	1.93	1.91	1.08	1.96	1.99	1.06	2.19	2.09	1.01
C12	1.96	1.84	0.96	1.98	1.82	0.95	2.01	1.87	0.92	2.16	1.94	0.90
C131	2.01	1.80	1.24	2.04	1.83	1.22	2.09	1.87	1.20	2.16	1.95	1.04
C132	2.01	1.79	1.24	2.04	1.83	1.22	2.09	1.86	1.20	2.18	1.94	1.04
C14	7.37	1.81	0.97	10.00	1.82	0.95	14.91	1.94	0.91	15.06	1.96	0.89
C15	1.82	1.83	0.96	1.85	1.86	0.95	1.89	1.94	0.93	1.98	2.00	0.89
C16	3.12	2.85	1.48	3.17	2.78	1.44	3.25	3.02	1.39	3.33	3.32	1.27
C17	-	3.08	1.03	-	3.13	1.02	-	3.26	0.94	-	3.45	0.91
AVG	**2.05**	**2.00**	**1.08**	**2.07**	**2.02**	**1.07**	**2.10**	**2.10**	**1.04**	**2.18**	**2.19**	**0.97**
STDEV	0.35	0.43	0.15	0.36	0.42	0.14	0.38	0.46	0.14	0.39	0.53	0.10
%RSD	**17.24**	**21.47**	**13.65**	**17.47**	**20.88**	**13.41**	**17.89**	**21.84**	**13.25**	**17.91**	**24.05**	**10.53**

Table D.26 Insert time of three databases in hot cache (20 MB)

Cmd.	Redundant Records											
	40			80			160			320		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	2.82	2.62	1.26	2.82	2.93	1.34	2.83	3.11	1.20	2.95	3.35	1.43
C3	2.73	3.12	1.27	2.74	3.15	1.35	2.80	3.40	1.21	2.92	3.77	1.45
C4	2.69	2.71	1.11	2.70	2.93	1.19	2.77	3.05	1.07	2.96	3.23	1.30
C5	2.41	2.91	1.41	2.41	3.13	1.40	2.43	3.26	1.40	2.55	3.50	1.23
C6	3.29	2.75	1.27	3.32	2.92	1.27	3.32	3.05	1.31	3.48	3.29	1.13
C7	2.99	2.90	1.37	2.97	2.93	1.44	3.09	3.03	1.30	3.28	3.28	1.52
C8	2.92	2.83	1.25	2.93	3.14	1.32	2.94	3.33	1.20	3.03	3.47	1.44
C9	3.04	2.72	1.13	3.05	2.95	1.13	3.05	3.16	1.16	3.15	3.36	1.03
C10	2.72	2.91	1.39	2.73	3.13	1.45	2.77	3.32	1.31	2.92	3.56	1.53
C12	2.86	2.61	1.19	2.86	2.75	1.08	2.90	2.85	1.07	3.05	3.14	0.99
C131	2.90	2.84	1.42	2.96	3.04	1.43	3.03	3.28	1.41	3.15	3.43	1.35
C132	2.92	2.96	1.42	2.97	3.03	1.43	3.05	3.28	1.42	3.16	3.43	1.35
C14	25.55	2.69	1.11	42.45	2.79	1.12	76.73	2.83	1.15	146.70	3.13	1.01
C15	2.54	2.72	1.10	2.56	2.78	1.18	2.65	2.93	1.06	2.82	3.28	1.30
C16	5.26	4.75	2.10	5.27	5.03	2.00	5.38	5.32	1.93	5.64	6.28	1.71
C17	-	5.44	1.13	-	5.63	1.22	-	6.00	1.16	-	6.57	1.02
AVG	**3.02**	**3.09**	**1.31**	**3.03**	**3.27**	**1.33**	**3.08**	**3.45**	**1.27**	**3.23**	**3.75**	**1.30**
STDEV	0.74	0.88	0.26	0.74	0.91	0.23	0.75	0.99	0.23	0.78	1.17	0.22
%RSD	**24.36**	**28.60**	**19.86**	**24.24**	**27.87**	**17.44**	**24.38**	**28.57**	**17.74**	**24.28**	**31.21**	**17.18**

Table D.27 Insert time of three databases in hot cache (40 MB)

Cmd.	Redundant Records											
	80			160			320			640		
	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd	nxd	sxd	lxd
C1	4.57	5.93	1.81	4.58	6.26	1.75	4.61	6.99	1.66	4.67	7.47	1.45
C3	4.33	5.84	1.82	4.34	6.03	1.78	4.41	6.87	1.68	4.56	7.48	1.48
C4	4.19	5.15	1.47	4.29	5.36	1.42	4.41	5.63	1.35	4.44	6.09	1.23
C5	3.75	5.72	2.26	3.78	5.98	2.12	3.83	6.57	2.04	3.91	7.02	1.68
C6	5.62	5.19	1.82	5.54	5.32	1.75	5.62	5.77	1.66	5.68	6.39	1.48
C7	4.85	5.58	2.06	4.88	5.74	1.97	5.01	6.17	1.88	5.14	6.58	1.64
C8	4.72	5.42	1.76	4.74	5.77	1.75	4.76	6.04	1.66	4.79	6.54	1.46
C9	4.94	5.17	1.51	4.95	5.44	1.47	4.98	5.69	1.39	5.03	6.27	1.27
C10	4.32	5.59	2.03	4.34	5.79	1.98	4.42	6.21	1.84	4.52	6.35	1.70
C12	4.65	4.95	1.38	4.66	5.39	1.36	4.70	5.61	1.34	4.79	6.10	1.20
C131	4.75	5.34	1.92	4.76	5.18	1.90	4.89	5.33	1.79	5.00	6.04	1.57
C132	4.75	5.32	1.91	4.77	5.12	1.89	4.87	5.36	1.78	4.98	6.11	1.57
C14	85.32	5.21	1.48	150.74	2.20	1.44	285.99	5.62	1.36	546.56	6.15	1.25
C15	3.94	5.14	1.44	3.96	5.32	1.40	4.14	5.68	1.33	4.20	6.54	1.22
C16	9.37	9.26	3.34	9.35	10.41	3.16	9.59	10.57	2.97	9.71	11.96	2.57
C17	-	9.74	1.41	-	9.98	1.39	-	10.66	1.35	-	11.53	1.24
AVG	**4.94**	**5.91**	**1.84**	**4.95**	**5.95**	**1.78**	**5.04**	**6.53**	**1.69**	**5.12**	**7.14**	**1.50**
STDEV	1.47	1.60	0.52	1.45	2.11	0.48	1.49	1.85	0.44	1.50	2.06	0.36
%RSD	**29.71**	**27.03**	**28.14**	**29.26**	**35.38**	**26.69**	**29.49**	**28.34**	**26.06**	**29.30**	**28.84**	**23.91**

Translating XML update commands into SQL according to update features

This section shows translating XML update commands into SQL for deletion and insertion in the case of sxd (replacing for sxd is shown in Chapter 7). It also shows the translation for lxd.

Table D.28 Translating XML update commands into SQL for deletion in sxd

Feature	XML update command	SQL in the form of PL/SQL
C1	For $p in doc("Library.xml")/Library/Publication, $i in $p/Publisher/Address, $n in $p/Author/Name Where $i/Country = "Thailand" Delete $n/MName	update Author A set A.Name.MName = null where A.AuthorID in (select A.AuthorID from Author A, Publisher P, Library L where P.Address.Country = 'Thailand' and A.PubID = L.Publication.PubID and P.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;
C3	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author delete $a/Telephone where $p/Year >= "2000" and $p/Year <= "2001"	select distinct A.AuthorID bulk collect into l_array from Author A, Library L where L.Publication.Year >= '2000' and L.Publication.Year <= '2001' and A.PubID = L.Publication.PubID; forall i in l_array.first..l_array.last delete table (select A.Telephone from Author A where A.AuthorID in l_array(i)) ;
C4	For $i in doc("Library.xml")/Library/Publication/Publisher where contains($i/PName, "Sams Publishing") and contains($i/PName, "LTD.") delete $i/Address	update Publisher P set P.Address.No = null, P.Address.Street = null, P.Address.City = null, P.Address.Country = null, P.Address.ZipCode = null where P.PName like '%Sams Publishing%' and P.PName like '%LTD.%' returning P.PID bulk collect into l_array;
C5	For $p in doc("Library.xml")/Library/Publication Let $a := $p/Author Where count($a) >= 2 delete $p@PubType	update Library L set L.Publication.PubType = null where L.Publication.PubID in (select L.Publication.PubID from Library L, Author A where A.PubID = L.Publication.PubID group by L.Publication.PubID having count(*) >= 2) returning L.Publication.PubID bulk collect into l_array;
C6	For $p in doc("Library.xml")/Library/Publication Where some $a in $p/Author satisfies ($a/Name/LName = "Dobson") delete $p@PubType Replace $p@PubType with @PubType = "book"	update Library L set L.Publication.PubType = null where L.Publication.PubID in (select L.Publication.PubID from Library L, Author A where A.Name.LName = 'Dobson' and A.PubID = L.Publication.PubID group by L.Publication.PubID having count(*) > 0) returning L.Publication.PubID bulk collect into l_array;

C7	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author, $i in $p/Publisher Where $p/Year = $i/SetupYear delete $i/Address	update Publisher P set P.Address.No = null , P.Address.Street = null , P.Address.City = null , P.Address.Country = null , P.Address.ZipCode = null where P.PID in (select P.PID from Publisher P, Library L where L.Publication.Year = P.SetupYear and P.PubID = L.Publication.PubID) returning P.PID bulk collect into l_array;
C8	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author Where $p@ContactAuthor = $a@AuthorID Delete $a/Email	update Author A set A.Email = null where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.ContactAuthor = A.AuthorID and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;
C9	For $p in doc("Library.xml")/Library/Publication Where $p/Cost = $p/SpecialCost * 2 delete $p/ShippingCost	update Library L set L.Publication.ShippingCost = null where L.Publication.Cost = L.Publication.SpecialCost *2 returning L.Publication.PubID bulk collect into l_array;
C10	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author, $i in $p/Publisher Where $i/SetupYear >= "2004" and $p/Year = "2005" delete $a/Telephone	select distinct A.AuthorID bulk collect into l_array from Author A, Publisher P, Library L where P.SetupYear >= '2004' and L.Publication.Year = '2005' and A.PubID = L.Publication.PubID and P.PubID = L.Publication.PubID; forall i in l_array.first..l_array.last delete table(select A.Telephone from Author A where A.AuthorID in l_array(i)) ;
C12	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author where empty($a/Name/MName) delete $a/Email	update Author A set A.Email = null where A.Name.MName is NULL returning A.AuthorID bulk collect into l_array;
C14	For $n in doc("Library.xml")//Name delete $n/MName	update Author A set A.Name.MName = null returning A.AuthorID bulk collect into l_array;
C15	For $p in doc("Library.xml")/Library/Publication, $i in $p/Publisher where contains($i/PName, "Sams") and $p/Year >= "2005" delete $i/Address	update Publisher P set P.Address.No = null , P.Address.Street = null , P.Address.City = null , P.Address.Country = null , P.Address.ZipCode = null where P.PID in (select P.PID from Publisher P, Library L where P.PName like '%Sams%' and L.Publication.Year >= '2005' and P.PubID = L.Publication.PubID) returning P.PID bulk collect into l_array;
C16	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author, $n in $a/Name Where $p/Cost >= "1500" delete $a/Email, delete $n/MName	update Author A set A.Name.MName = null where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.Cost >= '1500' and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array; update Author A set A.Email = null where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.Cost >= '1500' and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;

C17 & C11	For $p in doc("Library.xml")/Library/Publication
	Where $p/Title = "XML"
	allRef($p)
	define function allRef($pub as element()*)
	{ For $rp in $pub/Reference/@RefPub ->
	Publication
	Delete $rp@PubType
	allRef($rp)
	}
		delete from Array;
		insert into Array
		select L.Publication.PubID from Library L
		where L.Publication.Title = 'XML';
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R, Library L, Reference R2
		where R2.PubID = L.Publication.PubID
		and R.RefID = R2.RefID
		and L.Publication.PubID in (select * from ProcessingArr);
		insert into Collector select * from Array;
		update Library L
		set L.Publication.PubType = null
		where L.Publication.PubID in (select * from Array);
		Loop
		If SQL%RowCount > 0 then
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R,
		Library L, Reference R2
		where R2.PubID = L.Publication.PubID
		and R.RefID = R2.RefID
		and L.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Library L
		set L.Publication.PubType = null
		where L.Publication.PubID in
		(select * from Array);
		Else Exit;
		End If;
		End Loop;
		delete from Collector returning id bulk collect into l_array;

Table D.29 Translating XML update commands into SQL for insertion in sxd

Feature	XML update command	SQL in the form of PL/SQL
C1	For $p in doc("Library.xml")/Library/Publication, $i in $p/Publisher/Address, $n in $p/Author/Name Where $i/Country = "Thailand"	update Author A set A.Name.MName = 'Anantasamakom' where A.AuthorID in (select A.AuthorID from Author A, Publisher P, Library L where P.Address.Country = 'Thailand' and A.PubID = L.Publication.PubID and P.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;

C3	For $p in doc("Library.xml")/Library/Publication,
	$a in $p/Author
	insert <Telephone>
	<Location>Home</Location>
	<TelNo>5555555</TelNo>
	</Telephone>
	Into $a
	where $p/Year >= "2000"
	and $p/Year <= "2001"
		select distinct A.AuthorID bulk collect into l_array
		from Author A, Library L
		where L.Publication.Year >= '2000'
		and L.Publication.Year <= '2001'
		and A.PubID = L.Publication.PubID;
		
		forall i in l_array.first..l_array.last
		insert into table(select A.Telephone from Author A
		where A.AuthorID in l_array(i))
		values('Home', '5555555');

C4	For $i in doc("Library.xml")/Library/ Publication/Publisher where contains($i/PName, "Sams Publishing") and contains($i/PName, "LTD.") insert <Address> <No>12</No> <Street>Times</Street> <City>Ohio</City> <Country>USA</Country> <ZipCode>Ne5 ST7</ZipCode> </Address> Into $i	update Publisher P set P.Address.No = '12', P.Address.Street = 'Times', P.Address.City = 'Ohio', P.Address.Country = 'USA', P.Address.ZipCode = 'Ne5 ST7' where P.PName like '%Sams Publishing%' and P.PName like '%LTD.%' returning P.PID bulk collect into l_array;
C5	For $p in doc("Library.xml")/Library/Publication Let $a := $p/Author Where count($a) >= 2 Insert @PubType = "book" into $p	update Library L set L.Publication.PubType = 'book' where L.Publication.PubID in (select L.Publication.PubID from Library L, Author A where A.PubID = L.Publication.PubID group by L.Publication.PubID having count(*) >= 2) returning L.Publication.PubID bulk collect into l_array;
C6	For $p in doc("Library.xml")/Library/Publication Where some $a in $p/Author satisfies ($a/Name/LName = "Dobson") insert @PubType = "book" into $p	update Library L set L.Publication.PubType = 'book' where L.Publication.PubID in (select L.Publication.PubID from Library L, Author A where A.Name.LName = 'Dobson' and A.PubID = L.Publication.PubID group by L.Publication.PubID having count(*) > 0) returning L.Publication.PubID bulk collect into l_array;
C7	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author, $i in $p/Publisher Where $p/Year = $i/SetupYear insert <Address> <No>77</No> <Street>Avenue 5</Street> <City>New York</City> <Country>USA</Country> <ZipCode>Nr4 587</ZipCode> </Address> into $i	update Publisher P set P.Address.No = '77', P.Address.Street = 'Avenue 5', P.Address.City = 'New York', P.Address.Country = 'USA', P.Address.ZipCode = 'Nr4 587' where P.PID in (select P.PID from Publisher P, Library L where L.Publication.Year = P.SetupYear and P.PubID = L.Publication.PubID) returning P.PID bulk collect into l_array;

C8	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author Where $p@ContactAuthor = $a@AuthorID Insert <Email>contact@hotmail.com</Email> into $a	update Author A set A.Email = 'contact@hotmail.com' where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.ContactAuthor = A.AuthorID and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;
C9	For $p in doc("Library.xml")/Library/Publication Where $p/Cost = $p/SpecialCost * 2 insert <ShippingCost>50</ShippingCost> into $p	update Library L set L.Publication.ShippingCost = 50 where L.Publication.Cost = L.Publication.SpecialCost *2 returning L.Publication.PubID bulk collect into l_array;

C10	For $p in doc("Library.xml")/Library/Publication,
	$a in $p/Author,
	$i in $p/Publisher
	Where $i/SetupYear >= "2004"
	and $p/Year = "2005"
	insert
	<Telephone>
	<Location>School</Location>
	<TelNo>01912739145</TelNo>
	</Telephone>
	into $a
		select distinct A.AuthorID bulk collect into l_array
		from Author A, Publisher P, Library L
		where P.SetupYear >= '2004'
		and L.Publication.Year = '2005'
		and A.PubID = L.Publication.PubID
		and P.PubID = L.Publication.PubID;
		
		forall i in l_array.first..l_array.last
		insert into table(select A.Telephone from Author A
		where A.AuthorID in l_array(i))
		values('School', '01912739145');

C12	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author where empty($a/Name/MName) insert <Email>sams@hotmail.com</Email> into $a	update Author A set A.Email = 'sams@hotmail.com' where A.Name.MName is NULL returning A.AuthorID bulk collect into l_array;
C131	For $a in doc("Library.xml")/Library/Publication/Author insert <Telephone> <Location>School</Location> <TelNo>01912739145</TelNo> </Telephone> into $a where $a/Name/FName = "John" and $a/Name/LName = "Dobson" before $a/Telephone[Location = "Home"]	select distinct A.AuthorID bulk collect into l_array from Author A where A.Name.FName = 'John' and A.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last insert into table(select A.Telephone from Author A where A.AuthorID in l_array(i)) values('School', '01912739145');
C132	For $a in doc("Library.xml")/Library/Publication/Author insert <Telephone> <Location>School</Location> <TelNo>01912739145</TelNo> </Telephone> into $a where $a/Name/FName = "John" and $a/Name/LName = "Dobson" after $a/Telephone[Location = "Home"]	select distinct A.AuthorID bulk collect into l_array from Author A where A.Name.FName = 'John' and A.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last insert into table(select A.Telephone from Author A where A.AuthorID in l_array(i)) values('School', '01912739145');
C14	For $n in doc("Library.xml")//Name insert <MName>Anonymous</MName> into $n	update Author A set A.Name.MName = 'Anonymous' returning A.AuthorID bulk collect into l_array;
C15	For $p in doc("Library.xml")/Library/Publication, $i in $p/Publisher where contains($i/PName, "Sams") and $p/Year >= "2005" insert <Address> <No>12</No> <Street>Times</Street> <City>Ohio</City> <Country>USA</Country> <ZipCode>Ne5 ST7</ZipCode> </Address> Into $i	update Publisher P set P.Address.No = '12', P.Address.Street = 'Times', P.Address.City = 'Ohio', P.Address.Country = 'USA', P.Address.ZipCode = 'Ne5 ST7' where P.PID in (select P.PID from Publisher P, Library L where P.PName like '%Sams%' and L.Publication.Year >= '2005' and P.PubID = L.Publication.PubID) returning P.PID bulk collect into l_array;

C16	For $p in doc("Library.xml")/Library/Publication, $a in $p/Author, $n in $a/Name Where $p/Cost >= "1500" insert <Email>sams@hotmail.com</Email> into $a, insert <MName>Anonymous</MName> into $n	update Author A set A.Name.MName = 'Anonymous' where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.Cost >= '1500' and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array; update Author A set A.Email = 'sams@hotmail.com' where A.AuthorID in (select A.AuthorID from Author A, Library L where L.Publication.Cost >= '1500' and A.PubID = L.Publication.PubID) returning A.AuthorID bulk collect into l_array;

C17 &
C11
	For $p in doc("Library.xml")/Library/Publication
	Where $p/Title = "XML"
	allRef($p)
	define function allRef($pub as element()*)
	{
	For $rp in $pub/Reference/
	@RefPub ->Publication
	insert @PubType = "journal" into $rp
	allRef($rp)
	}
		delete from Array;
		insert into Array
		select L.Publication.PubID from Library L
		where L.Publication.Title = 'XML';
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R, Library L, Reference R2
		where R2.PubID = L.Publication.PubID
		and R.RefID = R2.RefID
		and L.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Library L
		set L.Publication.PubType = 'journal'
		where L.Publication.PubID in (select * from Array);
		Loop
		If SQL%RowCount > 0 then
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R,
		Library L, Reference R2
		where R2.PubID = L.Publication.PubID
		and R.RefID = R2.RefID
		and L.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Library L
		set L.Publication.PubType = 'journal'
		where L.Publication.PubID in
		(select * from Array);
		Else Exit;
		End If;
		End Loop;
		delete from Collector returning id bulk collect into l_array;

Table D.30 Translating XML update commands into SQL for replacing in lxd

Feature	XML update command	SQL in the form of PL/SQL
C1	For $p in doc("Publications.xml")/ Publications/Publication, $n in $p/Author ~>/Authors/Author/Name, $i in $p/Publisher ~>/Publishers/Publisher/Address Where $i/Country = "Thailand" Replace $n/MName with <MName>Anantasamakom</MName>	update Authors A set A.Author.Name.MName = 'Anantasamakom' where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publishers P, PublicationAuthor P2, Publications P3 where P.Publisher.Address.Country = 'Thailand' and P2.PubID = P3.Publication.PubID and P2.AuthorID = A.Author.AuthorID and P3.Publication.PID = P.Publisher.PID) returning A.Author.AuthorID bulk collect into l_array;
C3	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author ~>//Author Replace $a/Telephone with <Telephone> <Location>Office</Location> <TelNo>9999999</TelNo> </Telephone> where $p/Year >= "2000" and $p/Year <= "2001"	select distinct A.Author.AuthorID bulk collect into l_array from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Year >= '2000' and P.Publication.Year <= '2001' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID; forall i in l_array.first..l_array.last update table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) set Location = 'Office', TelNo = '9999999' ;
C4	For $i in doc("Publications.xml")/Publications/ Publication/Publisher ~>/Publishers/Publisher where contains($i/PName, "Sams Publishing") and contains($i/PName, "LTD.") Replace $i/Address with <Address> <No>22</No> <Street>Times</Street> <City>Ohio</City> <Country>USA</Country> <ZipCode>Ne5 ST7</ZipCode> </Address>	update Publishers P set P.Publisher.Address.No = '22', P.Publisher.Address.Street = 'Times', P.Publisher.Address.City = 'Ohio', P.Publisher.Address.Country = 'USA', P.Publisher.Address.ZipCode = 'Ne5 ST7' where P.Publisher.PName like '%Sams Publishing%' and P.Publisher.PName like '%LTD.%' returning P.Publisher.PID bulk collect into l_array;
C5	For $p in doc("Publications.xml")/ Publications/Publication Let $a := $p/Author~>/Authors/Author Where count($a) >= 2 Replace $p@PubType with @PubType = "book"	update Publications P set P.Publication.PubType = 'book' where P.Publication.PubID in (select P.Publication.PubID from Publications P, PublicationAuthor P2, Authors A where P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID group by P.Publication.PubID having count(*) >= 2) returning P.Publication.PubID bulk collect into l_array;
C6	For $p in doc("Publications.xml")/ Publications/Publication Where some $a in $p/Author~>/Authors/Author satisfies ($a/Name/LName = "Dobson") Replace $p@PubType with @PubType = "book"	update Publications P set P.Publication.PubType = 'book' where P.Publication.PubID in (select P.Publication.PubID from Publications P, Authors A, PublicationAuthor P2 where A.Author.Name.LName = 'Dobson' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID group by P.Publication.PubID having count(*) > 0) returning P.Publication.PubID bulk collect into l_array;
C7	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author ~>/Authors/Author, $i in $p/Publisher ~>/Publishers/Publisher Where $p/Year = $i/SetupYear Replace $i/Address with <Address> <No>77</No> <Street>Avenue 5</Street> <City>New York</City> <Country>USA</Country> <ZipCode>Nr4 587</ZipCode> </Address>	update Publishers P set P.Publisher.Address.No = '77', P.Publisher.Address.Street = 'Avenue 5', P.Publisher.Address.City = 'New York', P.Publisher.Address.Country = 'USA', P.Publisher.Address.ZipCode = 'Nr4 587' where P.Publisher.PID in (select P.Publisher.PID from Publishers P, Publications P2 where P2.Publication.Year = P.Publisher.SetupYear and P2.Publication.PID = P.Publisher.PID) returning P.Publisher.PID bulk collect into l_array;

C8	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author~>/Authors/Author Where $p@ContactAuthor = $a@AuthorID Replace $a/Email with <Email>contact@hotmail.com</Email>	update Authors A set A.Author.Email = 'contact@hotmail.com' where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.ContactAuthor = A.Author.AuthorID and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array;
C9	For $p in doc("Publications.xml")/ Publications/Publication Where $p/Cost = $p/SpecialCost * 2 Replace $p/ShippingCost with <ShippingCost>50</ShippingCost>	update Publications P set P.Publication.ShippingCost = 50 Where P.Publication.Cost =P.Publication.SpecialCost *2 returning P.Publication.PubID bulk collect into l_array;
C10	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author ~>/Authors/Author, $i in $p/Publisher ~>/Publishers/Publisher Where $i/SetupYear >= "2004" and $p/Year = "2005" Replace $a/Telephone with <Telephone> <Location>School</Location> <TelNo>01912739145</TelNo> </Telephone>	select distinct A.Author.AuthorID bulk collect into l_array from Authors A, Publishers P, Publications P2, PublicationAuthor P3 where P.Publisher.SetupYear >= '2004' and P2.Publication.Year = '2005' and P3.PubID = P2.Publication.PubID and P3.AuthorID = A.Author.AuthorID and P2.Publication.PID = P.Publisher.PID; forall i in l_array.first..l_array.last update table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) set Location = 'School', TelNo = '01912739145' ;
C12	For $a in doc("Authors.xml")/Authors/Author where empty($a/Name/MName) Replace $a/Email with <Email>sams@hotmail.com</Email>	update Authors A set A.Author.Email = 'sams@hotmail.com' where A.Author.Name.MName is NULL returning A.Author.AuthorID bulk collect into l_array;
C14	For $n in doc("Authors.xml")//Name replace $n/MName with <MName>Anonymous</MName>	update Authors A set A.Author.Name.MName = 'Anonymous' returning A.Author.AuthorID bulk collect into l_array;
C15	For $p in doc("Publications.xml")/ Publications/Publication, $i in $p/Publisher~>/Publishers/Publisher where contains($i/PName, "Sams") and $p/Year >= "2005" Replace $i/Address with <Address> <No>22</No> <Street>Times</Street> <City>Ohio</City> <Country>USA</Country> <ZipCode>Ne5 ST7</ZipCode> </Address>	update Publishers P set P.Publisher.Address.No = '22', P.Publisher.Address.Street = 'Times', P.Publisher.Address.City = 'Ohio', P.Publisher.Address.Country = 'USA', P.Publisher.Address.ZipCode = 'Ne5 ST7' where P.Publisher.PID in (select P.Publisher.PID from Publishers P, Publications P2 where P.Publisher.PName like '%Sams%' and P2.Publication.Year >= '2005' and P2.Publication.PID = P.Publisher.PID) returning P.Publisher.PID bulk collect into l_array;
C16	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author ~>//Author, $n in $a/Name Where $p/Cost >= "1500" replace $a/Email with <Email>sams2@hotmail.com</Email> , replace $n/MName with <MName>Anonymous2</MName>	update Authors A set A.Author.Name.MName = 'Anonymous2' where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Cost >= '1500' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array; update Authors A set A.Author.Email = 'sams2@hotmail.com' where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Cost >= '1500' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array;

C17 & C11	For $p in doc("Publications.xml")/
	Publications/Publication
	Where $p/Title = "XML"
	allRef($p)
	
	define function allRef($pub as element()*)
	{
	For $rp in $pub/Reference~>//Reference/
	Publication~>//Publication
	Replace $rp@PubType with @PubType = "journal"
	allRef($rp)
	}
		delete from Array;
		insert into Array
		select P.Publication.PubID from Publications P
		where P.Publication.Title = 'XML';
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID from ReferencePublication R,
		Publications P, References R2
		where R.RefID = R2.Reference.RefID
		and P.Publication.RefID = R2.Reference.RefID
		and P.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Publications P
		set P.Publication.PubType = 'journal'
		where P.Publication.PubID in (select * from Array);
		Loop
		If SQL%RowCount > 0 then
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R,
		Publications P, References R2
		where R.RefID = R2.Reference.RefID
		and P.Publication.RefID = R2.Reference.RefID
		and P.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Publications P
		set P.Publication.PubType = 'journal'
		where P.Publication.PubID in
		(select * from Array);
		Else Exit;
		End If;
		End Loop;
		delete from Collector returning id
		bulk collect into l_array;

Table D.31 Translating XML update commands into SQL for deletion in lxd

Feature	XML update command	SQL in the form of PL/SQL
C1	For $p in doc("Publications.xml")/ Publications/Publication, $n in $p/Author ~>/Authors/Author/Name, $i in $p/Publisher ~>/Publishers/Publisher/Address Where $i/Country = "Thailand" Delete $n/MName	update Authors A set A.Author.Name.MName = null where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publishers P, PublicationAuthor P2, Publications P3 where P.Publisher.Address.Country = 'Thailand' and P2.PubID = P3.Publication.PubID and P2.AuthorID = A.Author.AuthorID and P3.Publication.PID = P.Publisher.PID) returning A.Author.AuthorID bulk collect into l_array;
C3	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author ~>//Author delete $a/Telephone where $p/Year >= "2000" and $p/Year <= "2001"	select distinct A.Author.AuthorID bulk collect into l_array from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Year >= '2000' and P.Publication.Year <= '2001' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID; forall i in l_array.first..l_array.last delete table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) ;
C4	For $i in doc("Publications.xml")/Publications/ Publication/Publisher~>/Publishers/Publisher where contains($i/PName, "Sams Publishing") and contains($i/PName, "LTD.") delete $i/Address	update Publishers P set P.Publisher.Address.No = null , P.Publisher.Address.Street = null , P.Publisher.Address.City = null , P.Publisher.Address.Country = null , P.Publisher.Address.ZipCode = null where P.Publisher.PName like '%Sams Publishing%' and P.Publisher.PName like '%LTD.%' returning P.Publisher.PID bulk collect into l_array;
C5	For $p in doc("Publications.xml")/ Publications/Publication Let $a := $p/Author~>/Authors/Author Where count($a) >= 2 delete $p@PubType	update Publications P set P.Publication.PubType = null where P.Publication.PubID in (select P.Publication.PubID from Publications P, PublicationAuthor P2, Authors A where P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID group by P.Publication.PubID having count(*) >= 2) returning P.Publication.PubID bulk collect into l_array;
C6	For $p in doc("Publications.xml")/ Publications/Publication Where some $a in $p/Author~>/Authors/Author satisfies ($a/Name/LName = "Dobson") delete $p@PubType	update Publications P set P.Publication.PubType = null where P.Publication.PubID in (select P.Publication.PubID from Publications P, Authors A, PublicationAuthor P2 where A.Author.Name.LName = 'Dobson' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID group by P.Publication.PubID having count(*) > 0) returning P.Publication.PubID bulk collect into l_array;
C7	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author ~>/Authors/Author, $i in $p/Publisher ~>/Publishers/Publisher Where $p/Year = $i/SetupYear delete $i/Address	update Publishers P set P.Publisher.Address.No = null , P.Publisher.Address.Street = null , P.Publisher.Address.City = null , P.Publisher.Address.Country = null , P.Publisher.Address.ZipCode = null where P.Publisher.PID in (select P.Publisher.PID from Publishers P, Publications P2 where P2.Publication.Year = P.Publisher.SetupYear and P2.Publication.PID = P.Publisher.PID) returning P.Publisher.PID bulk collect into l_array;

C8	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author~>/Authors/Author Where $p@ContactAuthor = $a@AuthorID Delete $a/Email	update Authors A set A.Author.Email = null where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.ContactAuthor =A.Author.AuthorID and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array;
C9	For $p in doc("Publications.xml")/ Publications/Publication Where $p/Cost = $p/SpecialCost * 2 Replace $p/ShippingCost with <ShippingCost>50</ShippingCost>	update Publications P set P.Publication.ShippingCost = null where P.Publication.Cost =P.Publication.SpecialCost *2 returning P.Publication.PubID bulk collect into l_array;
C10	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author ~>/Authors/Author, $i in $p/Publisher ~>/Publishers/Publisher Where $i/SetupYear >= "2004" and $p/Year = "2005" delete $a/Telephone	select distinct A.Author.AuthorID bulk collect into l_array from Authors A, Publishers P, Publications P2, PublicationAuthor P3 where P.Publisher.SetupYear >= '2004' and P2.Publication.Year = '2005' and P3.PubID = P2.Publication.PubID and P3.AuthorID = A.Author.AuthorID and P2.Publication.PID = P.Publisher.PID; forall i in l_array.first..l_array.last delete table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) ;
C12	For $a in doc("Authors.xml")/Authors/Author where empty($a/Name/MName) delete $a/Email	update Authors A set A.Author.Email = null where A.Author.Name.MName is NULL returning A.Author.AuthorID bulk collect into l_array;
C14	For $n in doc("Authors.xml")//Name delete $n/MName	update Authors A set A.Author.Name.MName = null returning A.Author.AuthorID bulk collect into l_array;
C15	For $p in doc("Publications.xml")/Publications/Publication, $i in $p/Publisher~>/Publishers/Publisher where contains($i/PName, "Sams") and $p/Year >= "2005" delete $i/Address	update Publishers P set P.Publisher.Address.No = null , P.Publisher.Address.Street = null , P.Publisher.Address.City = null , P.Publisher.Address.Country = null , P.Publisher.Address.ZipCode = null where P.Publisher.PID in (select P.Publisher.PID from Publishers P, Publications P2 where P.Publisher.PName like '%Sams%' and P2.Publication.Year >= '2005' and P2.Publication.PID = P.Publisher.PID) returning P.Publisher.PID bulk collect into l_array;
C16	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author ~>//Author, $n in $a/Name Where $p/Cost >= "1500" delete $a/Email, delete $n/MName	update Authors A set A.Author.Name.MName = null where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Cost >= '1500' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array; update Authors A set A.Author.Email = null where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Cost >= '1500' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array;

C17 & C11	For $p in doc("Publications.xml")/
	Publications/Publication
	Where $p/Title = "XML"
	allRef($p)
	
	define function allRef($pub as element()*)
	{
	For $rp in $pub/Reference~>//Reference/
	Publication~>//Publication
	Replace $rp@PubType with @PubType = "journal"
	allRef($rp)
	}
		delete from Array;
		insert into Array
		select P.Publication.PubID from Publications P
		where P.Publication.Title = 'XML';
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID from ReferencePublication R,
		Publications P, References R2
		where R.RefID = R2.Reference.RefID
		and P.Publication.RefID = R2.Reference.RefID
		and P.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Publications P
		set P.Publication.PubType =null
		where P.Publication.PubID in (select * from Array);
		Loop
		If SQL%RowCount > 0 then
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R,
		Publications P, References R2
		where R.RefID = R2.Reference.RefID
		and P.Publication.RefID = R2.Reference.RefID
		and P.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Publications P
		set P.Publication.PubType = null
		where P.Publication.PubID in
		(select * from Array);
		Else Exit;
		End If;
		End Loop;
		delete from Collector returning id bulk collect into
		l_array;

Table D.32 Translating XML update commands into SQL for insertion in lxd

Feature	XML update command	SQL in the form of PL/SQL
C1	For $p in doc("Publications.xml")/ Publications/Publication, $n in $p/Author ~>/Authors/Author/Name, $i in $p/Publisher ~>/Publishers/Publisher/Address Where $i/Country = "Thailand" Insert <MName>Anantasamakom</MName> into $n	update Authors A set A.Author.Name.MName = 'Anantasamakom' where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publishers P, PublicationAuthor P2, Publications P3 where P.Publisher.Address.Country = 'Thailand' and P2.PubID = P3.Publication.PubID and P2.AuthorID = A.Author.AuthorID and P3.Publication.PID = P.Publisher.PID) returning A.Author.AuthorID bulk collect into l_array;

C3	For $p in doc("Publications.xml")/
	Publications/Publication,
	$a in $p/Author ~>//Author
	insert <Telephone>
	<Location>Home</Location>
	<TelNo>5555555</TelNo>
	</Telephone>
	Into $a
	where $p/Year >= "2000"
	and $p/Year <= "2001"
		select distinct A.Author.AuthorID bulk collect into l_array
		from Authors A, Publications P, PublicationAuthor P2
		where P.Publication.Year >= '2000'
		and P.Publication.Year <= '2001'
		and P2.PubID = P.Publication.PubID
		and P2.AuthorID = A.Author.AuthorID;
		
		forall i in l_array.first..l_array.last
		insert into table(select A.Author.Telephone
		from Authors A
		where A.Author.AuthorID in l_array(i))
		values('Home', '5555555');

C4	For $i in doc("Publications.xml")/Publications/ Publication/Publisher~>/Publishers/Publisher where contains($i/PName, "Sams Publishing") and contains($i/PName, "LTD.") insert <Address> <No>12</No> <Street>Times</Street> <City>Ohio</City> <Country>USA</Country> <ZipCode>Ne5 ST7</ZipCode> </Address> Into $i	update Publishers P set P.Publisher.Address.No = '22', P.Publisher.Address.Street = 'Times', P.Publisher.Address.City = 'Ohio', P.Publisher.Address.Country = 'USA', P.Publisher.Address.ZipCode = 'Ne5 ST7' where P.Publisher.PName like '%Sams Publishing%' and P.Publisher.PName like '%LTD.%' returning P.Publisher.PID bulk collect into l_array;
C5	For $p in doc("Publications.xml")/ Publications/Publication Let $a := $p/Author~>/Authors/Author Where count($a) >= 2 Insert @PubType = "book" into $p	update Publications P set P.Publication.PubType = 'book' where P.Publication.PubID in (select P.Publication.PubID from Publications P, PublicationAuthor P2, Authors A where P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID group by P.Publication.PubID having count(*) >= 2) returning P.Publication.PubID bulk collect into l_array;
C6	For $p in doc("Publications.xml")/ Publications/Publication Where some $a in $p/Author~>/Authors/Author satisfies ($a/Name/LName = "Dobson") insert @PubType = "book" into $p	update Publications P set P.Publication.PubType = 'book' where P.Publication.PubID in (select P.Publication.PubID from Publications P, Authors A, PublicationAuthor P2 where A.Author.Name.LName = 'Dobson' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID group by P.Publication.PubID having count(*) > 0) returning P.Publication.PubID bulk collect into l_array;

C7	For $p in doc("Publications.xml")/Publications/Publication, $a in $p/Author ~>/Authors/Author, $i in $p/Publisher ~>/Publishers/Publisher Where $p/Year = $i/SetupYear insert <Address> <No>77</No> <Street>Avenue 5</Street> <City>New York</City> <Country>USA</Country> <ZipCode>Nr4 587</ZipCode> </Address> into $i	update Publishers P set P.Publisher.Address.No = '77', P.Publisher.Address.Street = 'Avenue 5', P.Publisher.Address.City = 'New York', P.Publisher.Address.Country = 'USA', P.Publisher.Address.ZipCode = 'Nr4 587' where P.Publisher.PID in (select P.Publisher.PID from Publishers P, Publications P2 where P2.Publication.Year = P.Publisher.SetupYear and P2.Publication.PID = P.Publisher.PID) returning P.Publisher.PID bulk collect into l_array;
C8	For $p in doc("Publications.xml")/ Publications/Publication, $a in $p/Author~>/Authors/Author Where $p@ContactAuthor = $a@AuthorID Insert <Email>contact@hotmail.com</Email> into $a	update Authors A set A.Author.Email = 'contact@hotmail.com' where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.ContactAuthor =A.Author.AuthorID and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array;
C9	For $p in doc("Publications.xml")/ Publications/Publication Where $p/Cost = $p/SpecialCost * 2 insert <ShippingCost>50</ShippingCost> into $p	update Publications P set P.Publication.ShippingCost = 50 where P.Publication.Cost = P.Publication.SpecialCost *2 returning P.Publication.PubID bulk collect into l_array;

C10	For $p in doc("Publications.xml")/
	Publications/Publication,
	$a in $p/Author ~>/Authors/Author,
	$i in $p/Publisher ~>/Publishers/Publisher
	Where $i/SetupYear >= "2004"
	and $p/Year = "2005"
	insert
	<Telephone>
	<Location>School</Location>
	<TelNo>01912739145</TelNo>
	</Telephone>
	into $a
		select distinct A.Author.AuthorID bulk collect into l_array
		from Authors A, Publishers P,
		Publications P2, PublicationAuthor P3
		where P.Publisher.SetupYear >= '2004'
		and P2.Publication.Year = '2005'
		and P3.PubID = P2.Publication.PubID
		and P3.AuthorID = A.Author.AuthorID
		and P2.Publication.PID = P.Publisher.PID;
		forall i in l_array.first..l_array.last
		insert into table(select A.Author.Telephone
		from Authors A
		where A.Author.AuthorID in l_array(i))
		values('School', '01912739145');

C12	For $a in doc("Authors.xml")/Authors/Author where empty($a/Name/MName) insert <Email>sams@hotmail.com</Email> into $a	update Authors A set A.Author.Email = 'sams@hotmail.com' where A.Author.Name.MName is NULL returning A.Author.AuthorID bulk collect into l_array;
C131	For $a in doc("Authors.xml")/Authors/Author insert <Telephone> <Location>School</Location> <TelNo>01912739145</TelNo> </Telephone> into $a where $a/Name/FName = "John" and $a/Name/LName = "Dobson" before $a/Telephone[Location = "Home"]	select distinct A.Author.AuthorID bulk collect into l_array from Authors A where A.Author.Name.FName = 'John' and A.Author.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last insert into table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) values('School', '01912739145');
C132	For $a in doc("Authors.xml")/Authors/Author insert <Telephone> <Location>School</Location> <TelNo>01912739145</TelNo> </Telephone> into $a where $a/Name/FName = "John" and $a/Name/LName = "Dobson" after $a/Telephone[Location = "Home"]	select distinct A.Author.AuthorID bulk collect into l_array from Authors A where A.Author.Name.FName = 'John' and A.Author.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last insert into table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) values('School', '01912739145');
C14	For $n in doc("Authors.xml")//Name insert <MName>Anonymous</MName> into $n	update Authors A set A.Author.Name.MName = 'Anonymous' returning A.Author.AuthorID bulk collect into l_array;

C15	For $p in doc("Publications.xml")/Publications/Publication, $i in $p/Publisher~>/Publishers/Publisher where contains($i/PName, "Sams") and $p/Year >= "2005"Into $i	update Publishers P set P.Publisher.Address.No = '22', P.Publisher.Address.Street = 'Times', P.Publisher.Address.City = 'Ohio', P.Publisher.Address.Country = 'USA', P.Publisher.Address.ZipCode = 'Ne5 ST7' where P.Publisher.PID in (select P.Publisher.PID from Publishers P, Publications P2 where P.Publisher.PName like '%Sams%' and P2.Publication.Year >= '2005' and P2.Publication.PID = P.Publisher.PID) returning P.Publisher.PID bulk collect into l_array;
C16	For $p in doc("Publications.xml")/Publications/Publication, $a in $p/Author ~>//Author, $n in $a/Name Where $p/Cost >= "1500" insert <Email>sams@hotmail.com</Email> into $a, insert <MName>Anonymous</MName> into $n	update Authors A set A.Author.Name.MName = 'Anonymous2' where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Cost >= '1500' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array; update Authors A set A.Author.Email = 'sams2@hotmail.com' where A.Author.AuthorID in (select A.Author.AuthorID from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Cost >= '1500' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID) returning A.Author.AuthorID bulk collect into l_array;

C17 & C11	For $p in doc("Publications.xml")/
	Publications/Publication
	Where $p/Title = "XML"
	allRef($p)
	define function allRef($pub as element()*)
	{ For $rp in$pub/Reference~>//Reference/
	Publication~>//Publication
	insert @PubType = "journal" into $rp
	allRef($rp)
	}
		delete from Array;
		insert into Array
		select P.Publication.PubID from Publications P
		where P.Publication.Title = 'XML';
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID from ReferencePublication R,
		Publications P, References R2
		where R.RefID = R2.Reference.RefID
		and P.Publication.RefID = R2.Reference.RefID
		and P.Publication.PubID in (select * from
		ProcessingArr);
		insert into Collector select * from Array;
		update Publications P
		set P.Publication.PubType = 'journal'
		where P.Publication.PubID in (select * from Array);
		Loop
		If SQL%RowCount > 0 then
		delete from ProcessingArr;
		insert into ProcessingArr (select * from Array);
		delete from Array;
		insert into Array
		select R.PubID
		from ReferencePublication R,
		Publications P, References R2
		where R.RefID = R2.Reference.RefID
		and P.Publication.RefID = R2.Reference.RefID
		and P.Publication.PubID in
		(select * from ProcessingArr);
		insert into Collector select * from Array;
		update Publications P
		set P.Publication.PubType = 'journal'
		where P.Publication.PubID in
		(select * from Array);
		Else Exit;
		End If;
		End Loop;
		delete from Collector returning id bulk collect into
		l_array;

Translating update commands according to updating types of ORDB structure

Table D.33 Translating the update commands according to ORDB structure for sxd

Structure	XML update command	SQL in the form of PL/SQL
Replace Table	For $p in doc("Library.xml")//Publication Replace $p with <Publication PubID = "P777" PubType = "article"> <Title>XML Programming</Title> <Author AuthorID = "A002"> <Name> <FName>Sam</FName> <LName>Moore</LName> </Name> <Telephone> <Location>Home</Location> <TelNo>5555555</TelNo> </Telephone> </Author> <Year>2005</Year> <Publisher PID = "I666"> <PName>Addison-Wesly</PName> <Address> <No>12</No> <Street>Times</Street> <City>Ohio</City> <Country>USA</Country> <ZipCode>Ne5 ST7</ZipCode> </Address> <SetupYear>1999</SetupYear> </Publisher> <Reference RefID = "R555" RefType = "Bibliography" RefPub = "P111"> </Reference> </Publication> where $p/Title = "XQuery"	delete from Library L where L.Publication.Title = 'XQuery' returning L.Publication.PubID bulk collect into l_array; insert into Library L values(TPublication('P777', 'article', 'XML Programming', '2005', null, null, null, null)) ; insert into Author A values('A002', TName('Sam', null, 'Moore'), null, NTTelephone(TTelephone('Home', '5555555')), 'P777') ; insert into Publisher P values('I666', 'Addison-Wesly', TAddress('12', 'Times', 'Ohio', 'USA', 'Ne5 ST7'), '1999', 'P777') ; insert into Reference R values('R555', 'Bibliography', 'P777') ; insert into ReferencePublication R values('R555', 'P111') ;
Insert table	For $l in doc("Library.xml")/Library Insert <Publication PubID = "P999" PubType = "book"> <Title>XQuery</Title> <Year>2004</Year> <Author AuthorID = "A005"> <Name> <FName>Mc</FName> <LName>Donal</LName> </Name> <Email>mc@hotmail.com</Email> <Telephone> <Location>Office</Location> <TelNo>1111118</TelNo> </Telephone> </Author> <Publisher PID = "I777"> <PName>Mc GrawHill</PName> <Address> <No>212</No> <Street>Park Road</Street> <City>Miame</City> <Country>USA</Country> <ZipCode>Ne3 5TH</ZipCode> </Address> <SetupYear>1988</SetupYear> </Publisher> <Reference RefID = "R777" RefType = "References"> </Reference> </ Publication > Into $l	insert into Library L values(TPublication('P999', 'book', 'XQuery', '2004', null, null, null, null)) ; insert into Author A values('A005', TName('Mc', null, 'Donal'), 'mc@hotmail.com', NTTelephone(TTelephone('Office', '1111118')), 'P999') ; insert into Publisher P values('I777', 'Mc GrawHill', TAddress('212', 'Park Road', 'Miame', 'USA', 'Ne3 5TH'), '1988', 'P999') ; insert into Reference R values('R777', 'References', 'P999') ;
Delete Table	For $l in doc("Library.xml")/Library delete $l/Publication where $l/Publication/Title = "XQuery"	delete from Library L where L.Publication.Title = 'XQuery' returning L.Publication.PubID bulk collect into l_array;

Replace NTable	For $p in doc("Library.xml")//Publication, $a in $p/Author Where $a/Name/LName = "Dobson" replace $a/Telephone with <Telephone> <Location>Department</Location> <TelNo>01912739145</TelNo> </Telephone> where $a/Telephone/Location = "Home"	select distinct A.AuthorID bulk collect into l_array from Author A where A.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last update table(select A.Telephone from Author A where A.AuthorID in l_array(i)) set Location = 'Department', TelNo = '01912739145' where Location='Home';
Insert NTable	For $p in doc("Library.xml")//Publication, $a in $p/Author insert <Telephone> <Location>School</Location> <TelNo>01912739145</TelNo> </Telephone> into $a where $a/Name/FName = "John" and $a/Name/LName = "Dobson"	select distinct A.AuthorID bulk collect into l_array from Author A where A.Name.FName = 'John' and A.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last insert into table(select A.Telephone from Author A where A.AuthorID in l_array(i)) values('School', '01912739145');
Delete NTable	For $p in doc("Library.xml")//Publication, $a in $p/Author Where $a/Name/LName = "Dobson" delete $a/Telephone where $a/Telephone/Location = "Office"	select distinct A.AuthorID bulk collect into l_array from Author A where A.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last delete table(select A.Telephone from Author A where A.AuthorID in l_array(i)) where Location='Office';
Replace NTF	For $p in doc("Library.xml")//Publication, $a in $p/Author Where $a/Name/LName = "Dobson" replace $a/Telephone/TelNo with <TelNo>01912739145</TelNo> where $a/Telephone/Location = "Home"	select distinct A.AuthorID bulk collect into l_array from Author A where A.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last update table(select A.Telephone from Author A where A.AuthorID in l_array(i)) set TelNo = '01912739145' where Location='Home';
Insert NTF	For $p in doc("Library.xml")//Publication, $a in $p/Author Where $a/Name/LName = "Dobson" insert <TelNo>01912739145</TelNo> into $a/Telephone where $a/Telephone/Location = "Home"	select distinct A.AuthorID bulk collect into l_array from Author A where A.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last update table(select A.Telephone from Author A where A.AuthorID in l_array(i)) set TelNo = '01912739145' where Location='Home';
Delete NTF	For $p in doc("Library.xml")//Publication, $a in $p/Author Where $a/Name/LName = "Dobson" delete $a/Telephone/TelNo	select distinct A.AuthorID bulk collect into l_array from Author A where A.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last update table(select A.Telephone from Author A where A.AuthorID in l_array(i)) set TelNo = null where Location='Home';
Replace ADT	For $p in doc("Library.xml")//Publication, $a in $p/Author Replace $a/Name with <Name> <FName>Mc</FName> <LName>Donal</LName> </Name> where $a/Name/LName = "Dobson"	update Author A set A.Name.FName = 'Mc', A.Name.LName = 'Donal' where A.Name.LName = 'Dobson' returning A.AuthorID bulk collect into l_array;

Insert ADT	For $p in doc("Library.xml")//Publication, $a in $p/Author insert <Name> <FName>Mc</FName> <LName>Donal</LName> </Name> into $a where $a/Name/LName = "Dobson"	update Author A set A.Name.FName = 'Mc', A.Name.LName = 'Donal' where A.Name.LName = 'Dobson' returning A.AuthorID bulk collect into l_array;
Delete ADT	For $p in doc("Library.xml")//Publication, $a in $p/Author delete $a/Name	update Author A set A.Name.FName = null , A.Name.MName = null , A.Name.LName = null returning A.AuthorID bulk collect into l_array;
Replace ADTF	For $p in doc("Library.xml")//Publication, $a in $p/Author Replace $a/Name/MName with <MName>Ray</MName> where $a/Name/FName = "John" and $a/Name/LName = "Dobson"	update Author A set A.Name.MName = 'Ray' where A.Name.LName = 'Dobson' returning A.AuthorID bulk collect into l_array;
Insert ADTF	For $p in doc("Library.xml")//Publication, $a in $p/Author insert <MName>Ray</MName> into $a/Name where $a/Name/LName = "Dobson" after $a/Name/FName	update Author A set A.Name.MName = 'Ray' where A.Name.LName = 'Dobson' returning A.AuthorID bulk collect into l_array;
Delete ADTF	For $p in doc("Library.xml")//Publication, $a in $p/Author delete $a/Name/MName where $a/Name/LName = "Dobson"	update Author A set A.Name.MName = null where A.Name.LName = 'Dobson' returning A.AuthorID bulk collect into l_array;

Table D.34 Translating the update commands according to ORDB structure for lxd

Structure	XML update command	SQL in the form of PL/SQL

Replace Table	For $p in doc("Publications.xml")//Publication
	where $p/Title = "XQuery"
	Replace $p with
	<Publication PubID = "P777" PubType = "article">
	<Title>XML</Title>
	<Year>2005</Year>
	rlink(Author, //Author[Name/FName = "John"])
	rlink(Author, //Author[@AuthorID = "A444"])
	rlink(Publisher, //Publisher[@PID = "I111"])
	</Publication>
		delete from Publications P
		where P.Publication.Title = 'XQuery'
		returning P.Publication.PubID bulk collect into l_array;
		
		insert into Publications P
		values(TPublication('P777', 'article', 'XML', '2005', 'I111',
		null, null, null, null, null)) ;
		
		select distinct A.Author.AuthorID bulk collect into l_array
		from Authors A where A.Author.Name.FName = 'John';
		forall i in l_array.first..l_array.last
		insert into PublicationAuthor P
		values('P777', l_array(i)) ;
		
		insert into PublicationAuthor P values('P777', 'A444') ;

Insert table	For $p in doc("Publications.xml")/Publications,
	$a in $p/Publication/Author~>/Authors,
	$r in $p/Publication/Reference~>/References
	insert
	<Publication PubID = "P005" PubType = "article">
	<Title>JDK</Title>
	<Year>2004</Year>
	rlink(Publisher, //Publisher[@PID = "I111"])
	</Publication>
	into $p,
	insert <Author AuthorID = "A114">
	<Name> <FName>Pensri</FName>
	<MName>Ajarn</MName>
	<LName>Amornsin</LName>
	</Name>
	<Email>kokkoy@hotmail.com</Email>
	<Telephone> <Location>Home</Location>
	<TelNo>019127399145</TelNo>
	</Telephone>
	</Author>
	into $a,
	insert
	<Reference RefID = "R003" RefType = "Bibliography">
	rlink(Publication, //Publication[@PubID = "P444"])
	rlink(Publication, //Publication[@PubID = "P222"])
	</Reference>
	into $r
		insert into Authors A
		values(TAuthor('A114', TName('Pensri', 'Ajarn',
		'Amornsin'), 'kokkoy@hotmail.com',
		NTTelephone(TTelephone('Home',
		'019127399145'))))
		returning A.Author.AuthorID bulk collect into l_array;
		
		insert into References R
		values(TReference('R003', 'Bibliography'))
		returning R.Reference.RefID bulk collect into l_array;
		
		insert into Publications P
		values(TPublication('P005', 'article', 'JDK', '2004', 'I111',
		'R003', null, null, null, null)) ;
		
		insert into ReferencePublication R
		values('R003', 'P444') ;
		
		insert into ReferencePublication R
		values('R003', 'P222') ;
		
		insert into PublicationAuthor P
		values('P005', 'A114') ;

Delete Table	For $p in doc("Publications.xml")//Publication delete $p where $p/Title = "JDK";	delete from Publications P where P.Publication.Title = 'JDK' returning P.Publication.PubID bulk collect into l_array;
Replace NTable	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author where $p/Title = "XML" Replace $a/Telephone with <Telephone> <Location>Office</Location> <TelNo>01912739145</TelNo> </Telephone> where $a/Telephone/Location = "Home"	select distinct A.Author.AuthorID bulk collect into l_array from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Title = 'XML' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID; forall i in l_array.first..l_array.last update table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) set Location = 'Office', TelNo = '01912739145' where Location='Home';
Insert NTable	For $a in doc("Authors.xml")//Author insert <Telephone> <Location>Office</Location> <TelNo>01912737645</TelNo> </Telephone> into $a where $a/Name/FName = "Sam"	select distinct A.Author.AuthorID bulk collect into l_array from Authors A where A.Author.Name.FName = 'Sam'; forall i in l_array.first..l_array.last insert into table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) values('Office', '01912737645');

Delete NTable	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author Where $a/Name/LName = "Dobson" delete $a/Telephone where $a/Telephone/Location = "Office"	select distinct A.Author.AuthorID bulk collect into l_array from Authors A where A.Author.Name.LName = 'Dobson'; forall i in l_array.first..l_array.last delete table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) where Location='Office';
Replace NTF	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author where $p/Title = "XML" Replace $a/Telephone/TelNo with <TelNo>01912739145</TelNo> where $a/Telephone/Location = "Home"	select distinct A.Author.AuthorID bulk collect into l_array from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Title = 'XML' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID; forall i in l_array.first..l_array.last update table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) set TelNo = '01912739145' where Location='Home';
Insert NTF	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author where $p/Title = "XML" insert <TelNo>01912739145</TelNo> into $a/Telephone where $a/Telephone/Location = "Home"	select distinct A.Author.AuthorID bulk collect into l_array from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Title = 'XML' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID; forall i in l_array.first..l_array.last update table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) set TelNo = '01912739145' where Location='Home';
Delete NTF	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author where $p/Title = "XML" delete $a/Telephone/TelNo where $a/Telephone/Location = "Home"	select distinct A.Author.AuthorID bulk collect into l_array from Authors A, Publications P, PublicationAuthor P2 where P.Publication.Title = 'XML' and P2.PubID = P.Publication.PubID and P2.AuthorID = A.Author.AuthorID; forall i in l_array.first..l_array.last update table(select A.Author.Telephone from Authors A where A.Author.AuthorID in l_array(i)) set TelNo = null where Location='Home';
Replace ADT	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author Replace $a/Name <Name> <FName>Mc</FName> <LName>Donal</LName> </Name> where $a/Name/LName = "Dobson"	update Authors A set A.Author.Name.FName = 'Mc', A.Author.Name.LName = 'Donal' where A.Author.Name.LName = 'Dobson' returning A.Author.AuthorID bulk collect into l_array;
Insert ADT	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author Insert <Name> <FName>Mc</FName> <LName>Donal</LName> </Name> into $a where $a/Name/LName = "Dobson"	update Authors A set A.Author.Name.FName = 'Mc', A.Author.Name.LName = 'Donal' where A.Author.Name.LName = 'Dobson' returning A.Author.AuthorID bulk collect into l_array;
Delete ADT	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author delete $a/Name where $a/Name/LName = "Dobson"	update Authors A set A.Author.Name.FName = null , A.Author.Name.MName = null , A.Author.Name.LName = null where A.Author.Name.LName = 'Dobson' returning A.Author.AuthorID bulk collect into l_array;

Replace ADTF	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author Replace $a/Name/MName with <MName>Ray</MName> where $a/Name/FName = "John" and $a/Name/LName = "Dobson"	update Authors A set A.Author.Name.MName = 'Ray' where A.Author.Name.FName = 'John' and A.Author.Name.LName = 'Dobson' returning A.Author.AuthorID bulk collect into l_array;
Insert ADTF	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author insert <MName>Ray</MName> into $a/Name where $a/Name/LName = "Dobson" after $a/Name/FName	update Authors A set A.Author.Name.MName = 'Ray' where A.Author.Name.LName = 'Dobson' returning A.Author.AuthorID bulk collect into l_array;
Delete ADTF	For $p in doc("Publications.xml")//Publication, $a in $p/Author ~>//Author delete $a/Name/MName where $a/Name/LName = "Dobson"	update Authors A set A.Author.Name.MName = null where A.Author.Name.LName = 'Dobson' returning A.Author.AuthorID bulk collect into l_array;

Detail-Table

This section shows the detail-table for nxd, sxd and lxd. The execution time for nxd consists of update-time of database and serialization-time to XML document. The execution time for sxd and lxd consists of translate-time of the language, update-time of ORDB, update-time of DOM and serialization-time to XML documents.

I) Detail-Table for 5 MB. data size in cold cache

Table D.35 Replace time of 5 MB. data size, 10 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.13	3.37	5.49	3.75	2.75	1.00	1.57	5.32	4.18	3.83	0.35	0.56	4.74
C3	2.75	3.36	6.11	3.51	2.58	0.93	1.58	5.09	4.19	3.84	0.35	0.56	4.75
C4	3.37	3.42	6.79	3.55	2.62	0.92	1.57	5.12	3.25	2.90	0.35	0.56	3.81
C5	1.98	3.49	5.46	3.59	2.66	0.93	1.60	5.19	3.27	2.91	0.36	0.56	3.83
C6	2.29	3.43	5.71	3.62	2.68	0.93	1.59	5.21	3.53	3.17	0.36	0.55	4.08
C7	4.13	3.38	7.51	3.69	2.75	0.93	1.60	5.29	3.36	3.01	0.35	0.55	3.91
C8	2.19	3.40	5.59	3.29	2.36	0.94	1.61	4.90	3.01	2.67	0.34	0.56	3.57
C9	2.13	3.39	5.52	2.92	1.99	0.93	1.58	4.50	3.05	2.70	0.35	0.55	3.60
C10	2.55	3.41	5.95	3.74	2.81	0.92	1.58	5.32	3.37	3.02	0.35	0.55	3.92
C12	2.13	3.37	5.51	3.55	2.63	0.92	1.59	5.14	3.18	2.82	0.35	0.55	3.73
C14	2.24	3.45	5.68	3.50	2.58	0.92	1.56	5.07	3.18	2.82	0.35	0.56	3.73
C15	3.23	3.38	6.61	3.73	2.81	0.92	1.57	5.30	3.17	2.82	0.35	0.56	3.73
C16	2.78	3.37	6.15	3.01	1.88	1.13	1.58	4.60	3.24	2.78	0.46	0.55	3.79
C17				4.43	3.53	0.91	1.58	6.01	3.54	3.18	0.36	0.56	4.10

Table D.36 Replace time of 5 MB. data size, 20 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.13	3.41	5.55	3.76	2.76	1.00	1.57	5.34	4.13	3.78	0.35	0.54	4.67
C3	2.77	3.44	6.20	3.52	2.58	0.94	1.58	5.10	4.13	3.78	0.35	0.54	4.67
C4	3.43	3.41	6.84	3.56	2.63	0.93	1.58	5.14	3.24	2.89	0.35	0.54	3.77
C5	1.99	3.43	5.42	3.60	2.66	0.93	1.59	5.19	3.07	2.71	0.36	0.54	3.60
C6	2.30	3.35	5.64	3.63	2.69	0.94	1.61	5.24	3.03	2.68	0.35	0.54	3.57
C7	4.22	3.44	7.65	3.70	2.76	0.94	1.58	5.28	3.22	2.87	0.34	0.54	3.75
C8	2.21	3.45	5.66	3.30	2.36	0.94	1.59	4.89	2.81	2.47	0.34	0.54	3.35
C9	2.37	3.43	5.80	2.93	1.99	0.94	1.59	4.52	2.87	2.52	0.35	0.54	3.40
C10	2.57	3.35	5.92	3.75	2.82	0.93	1.59	5.33	3.10	2.76	0.34	0.53	3.64
C12	2.16	3.30	5.46	3.56	2.62	0.93	1.57	5.13	3.01	2.67	0.34	0.54	3.55
C14	3.54	3.35	6.89	3.50	2.58	0.92	1.60	5.10	2.96	2.61	0.35	0.53	3.49
C15	3.25	3.31	6.55	3.74	2.81	0.93	1.58	5.32	3.03	2.68	0.35	0.53	3.56
C16	2.79	3.34	6.13	3.02	1.88	1.14	1.58	4.60	3.11	2.65	0.46	0.53	3.64
C17				4.45	3.53	0.92	1.59	6.04	3.52	3.16	0.36	0.54	4.05

Table D.37 Replace time of 5 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.19	3.40	5.59	3.77	2.77	1.00	1.64	5.42	3.97	3.62	0.34	0.53	4.50
C3	2.79	3.42	6.21	3.53	2.59	0.94	1.61	5.14	3.96	3.61	0.35	0.53	4.49
C4	3.05	3.46	6.51	3.56	2.63	0.93	1.57	5.13	3.07	2.72	0.35	0.52	3.60
C5	1.99	3.43	5.43	3.61	2.67	0.94	1.60	5.21	3.08	2.72	0.36	0.53	3.60
C6	3.03	3.42	6.45	3.63	2.69	0.94	1.57	5.21	2.88	2.53	0.35	0.53	3.41
C7	4.24	3.38	7.62	3.71	2.77	0.95	1.58	5.29	3.07	2.73	0.34	0.53	3.59
C8	2.29	3.40	5.69	3.31	2.36	0.94	1.61	4.92	2.68	2.33	0.34	0.52	3.20
C9	2.40	3.38	5.78	2.94	2.00	0.94	1.61	4.55	2.81	2.48	0.34	0.52	3.34
C10	2.59	3.42	6.01	3.05	2.12	0.92	1.57	4.62	2.98	2.63	0.35	0.52	3.50
C12	2.18	3.42	5.60	3.58	2.64	0.94	1.59	5.16	2.87	2.52	0.35	0.53	3.40
C14	6.16	3.38	9.54	3.51	2.59	0.91	1.58	5.09	2.80	2.46	0.34	0.52	3.32
C15	3.29	3.37	6.65	3.75	2.81	0.94	1.58	5.33	2.86	2.52	0.34	0.53	3.39
C16	2.80	3.38	6.18	3.11	1.88	1.23	1.60	4.71	2.96	2.51	0.45	0.53	3.49
C17				4.45	3.53	0.92	1.66	6.11	3.34	2.99	0.36	0.53	3.87

Table D.38 Replace time of 5 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.19	3.44	5.63	3.78	2.77	1.01	1.58	5.36	3.80	3.46	0.34	0.50	4.29
C3	2.79	3.49	6.28	3.54	2.59	0.95	1.58	5.12	3.80	3.45	0.35	0.49	4.28
C4	2.09	3.44	5.53	3.57	2.63	0.94	1.58	5.15	2.90	2.56	0.34	0.48	3.38
C5	2.01	3.42	5.43	3.62	2.67	0.95	1.64	5.26	2.90	2.55	0.35	0.48	3.39
C6	3.05	3.33	6.38	3.66	2.70	0.96	1.62	5.28	2.67	2.33	0.35	0.49	3.17
C7	4.26	3.36	7.62	3.72	2.77	0.96	1.61	5.34	2.92	2.58	0.34	0.49	3.41
C8	2.30	3.34	5.65	3.33	2.37	0.96	1.60	4.93	2.46	2.12	0.35	0.48	2.95
C9	2.43	3.37	5.79	2.95	2.00	0.95	1.58	4.52	2.58	2.23	0.34	0.48	3.05
C10	2.61	3.37	5.97	3.06	2.12	0.94	1.58	4.65	2.81	2.47	0.34	0.48	3.29
C12	2.20	3.36	5.56	3.60	2.64	0.95	1.56	5.16	2.71	2.37	0.34	0.49	3.19
C14	10.90	3.36	14.25	3.53	2.60	0.93	1.58	5.11	2.64	2.30	0.34	0.48	3.12
C15	3.29	3.39	6.68	3.76	2.82	0.94	1.58	5.34	2.71	2.37	0.34	0.48	3.20
C16	2.83	3.35	6.17	3.13	1.89	1.24	1.56	4.69	2.83	2.38	0.44	0.49	3.31
C17				4.46	3.54	0.92	1.61	6.06	3.19	2.83	0.35	0.49	3.68

Table D.39 Delete time of 5 MB. data size, 10 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.12	3.30	5.42	3.75	2.76	0.99	1.58	5.33	4.19	3.84	0.35	0.55	4.75
C3	2.31	3.34	5.64	3.52	2.58	0.94	1.59	5.11	4.20	3.84	0.35	0.56	4.75
C4	1.98	3.38	5.36	3.56	2.62	0.93	1.55	5.10	3.26	2.91	0.35	0.55	3.82
C5	2.01	3.44	5.45	3.57	2.64	0.93	1.56	5.13	3.26	2.90	0.36	0.56	3.82
C6	2.31	3.48	5.78	3.61	2.67	0.94	1.58	5.19	3.17	2.80	0.36	0.55	3.72
C7	2.23	3.46	5.68	3.68	2.75	0.93	1.57	5.25	3.35	3.01	0.33	0.55	3.90
C8	2.22	3.46	5.67	3.29	2.36	0.93	1.58	4.87	3.00	2.65	0.34	0.55	3.55
C9	2.16	3.47	5.62	2.92	1.99	0.93	1.57	4.50	3.06	2.70	0.35	0.56	3.62
C10	2.14	3.42	5.57	3.73	2.82	0.91	1.55	5.29	3.35	3.01	0.34	0.55	3.91
C12	2.13	3.39	5.51	3.45	2.52	0.93	1.58	5.03	3.18	2.83	0.34	0.55	3.73
C14	2.22	3.37	5.59	3.50	2.58	0.92	1.57	5.08	3.17	2.82	0.36	0.56	3.73
C15	2.11	3.38	5.49	3.73	2.81	0.92	1.58	5.31	3.18	2.82	0.36	0.55	3.74
C16	2.56	3.40	5.96	3.00	1.87	1.13	1.57	4.58	3.23	2.77	0.46	0.55	3.78
C17				4.41	3.53	0.88	1.58	5.99	3.52	3.17	0.35	0.56	4.08

Table D.40 Delete time of 5 MB. data size, 20 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.13	3.40	5.52	3.77	2.77	1.01	1.59	5.36	4.13	3.79	0.35	0.54	4.67
C3	2.32	3.40	5.73	3.51	2.58	0.93	1.58	5.09	4.14	3.79	0.35	0.54	4.67
C4	2.00	3.38	5.38	3.55	2.62	0.93	1.58	5.14	3.24	2.90	0.35	0.54	3.78
C5	2.08	3.45	5.53	3.59	2.66	0.93	1.54	5.12	3.07	2.71	0.36	0.54	3.60
C6	2.35	3.42	5.77	3.61	2.68	0.93	1.55	5.16	3.03	2.67	0.36	0.54	3.57
C7	2.25	3.37	5.62	3.70	2.76	0.94	1.57	5.27	3.15	2.81	0.34	0.53	3.68
C8	2.23	3.32	5.55	3.29	2.36	0.93	1.58	4.87	2.79	2.45	0.34	0.54	3.33
C9	2.18	3.33	5.51	2.93	1.99	0.94	1.61	4.54	2.87	2.52	0.35	0.53	3.40
C10	2.16	3.36	5.52	3.75	2.82	0.93	1.60	5.35	3.07	2.73	0.34	0.54	3.60
C12	2.15	3.41	5.56	3.57	2.64	0.93	1.58	5.14	3.06	2.71	0.34	0.54	3.59
C14	3.49	3.36	6.85	3.51	2.59	0.92	1.57	5.09	2.95	2.60	0.35	0.54	3.49
C15	2.13	3.41	5.54	3.74	2.81	0.93	1.57	5.31	3.01	2.65	0.35	0.53	3.54
C16	2.58	3.46	6.04	3.00	1.87	1.13	1.57	4.56	3.09	2.65	0.45	0.54	3.63
C17				4.36	3.45	0.91	1.56	5.93	3.50	3.15	0.35	0.53	4.03

Table D.41 Delete time of 5 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.15	3.44	5.59	3.78	2.77	1.01	1.58	5.36	3.97	3.62	0.34	0.53	4.49
C3	2.35	3.47	5.81	3.54	2.59	0.95	1.58	5.12	3.97	3.62	0.34	0.53	4.50
C4	2.02	3.43	5.44	3.57	2.63	0.94	1.57	5.13	3.08	2.73	0.34	0.52	3.60
C5	2.10	3.45	5.55	3.60	2.66	0.93	1.56	5.15	3.07	2.72	0.35	0.52	3.60
C6	2.37	3.43	5.80	3.63	2.69	0.94	1.57	5.21	2.88	2.52	0.35	0.53	3.41
C7	2.27	3.38	5.65	3.72	2.77	0.95	1.57	5.29	3.06	2.72	0.33	0.52	3.58
C8	2.26	3.36	5.62	3.31	2.36	0.95	1.59	4.90	2.68	2.34	0.34	0.52	3.21
C9	2.19	3.37	5.56	2.95	2.00	0.95	1.54	4.49	2.82	2.48	0.34	0.53	3.35
C10	2.18	3.39	5.57	3.06	2.12	0.94	1.60	4.67	2.96	2.61	0.35	0.52	3.48
C12	2.16	3.40	5.56	3.58	2.64	0.94	1.57	5.15	2.86	2.51	0.35	0.52	3.38
C14	5.92	3.40	9.33	3.54	2.61	0.93	1.56	5.10	2.82	2.46	0.36	0.53	3.35
C15	2.24	3.42	5.66	3.71	2.77	0.94	1.60	5.31	2.87	2.52	0.35	0.52	3.39
C16	2.59	3.40	5.99	3.11	1.89	1.22	1.60	4.71	2.96	2.52	0.44	0.53	3.49
C17				4.44	3.53	0.91	1.62	6.06	3.33	2.98	0.35	0.52	3.85

Table D.42 Delete time of 5 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.18	3.41	5.59	3.79	2.78	1.01	1.58	5.37	3.79	3.45	0.34	0.50	4.29
C3	2.36	3.42	5.78	3.51	2.56	0.95	1.58	5.10	3.80	3.46	0.34	0.48	4.29
C4	2.03	3.41	5.44	3.57	2.63	0.94	1.57	5.14	2.90	2.56	0.34	0.48	3.38
C5	2.12	3.46	5.58	3.60	2.67	0.93	1.57	5.17	2.90	2.55	0.35	0.49	3.39
C6	2.38	3.43	5.81	3.66	2.71	0.95	1.56	5.21	2.68	2.33	0.35	0.49	3.17
C7	2.28	3.39	5.67	3.72	2.77	0.95	1.56	5.28	2.92	2.58	0.33	0.48	3.40
C8	2.27	3.36	5.63	3.32	2.36	0.96	1.57	4.90	2.45	2.12	0.33	0.49	2.94
C9	2.21	3.35	5.56	2.94	1.99	0.95	1.58	4.52	2.56	2.22	0.34	0.49	3.05
C10	2.19	3.35	5.54	3.08	2.13	0.95	1.62	4.71	2.81	2.46	0.34	0.48	3.28
C12	2.18	3.38	5.56	3.58	2.63	0.94	1.60	5.18	2.71	2.37	0.35	0.48	3.19
C14	10.78	3.38	14.15	3.53	2.60	0.93	1.58	5.11	2.66	2.31	0.34	0.49	3.14
C15	2.25	3.40	5.65	3.73	2.80	0.93	1.60	5.33	2.72	2.37	0.35	0.48	3.20
C16	2.62	3.41	6.03	3.14	1.90	1.24	1.67	4.80	2.82	2.37	0.44	0.49	3.31
C17				4.46	3.53	0.93	1.58	6.04	3.51	3.17	0.35	0.49	4.01

Table D.43 Insert time of 5 MB. data size, 10 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.14	3.44	5.58	3.75	2.76	1.00	1.56	5.31	4.18	3.83	0.35	0.56	4.74
C3	2.33	3.41	5.73	3.37	2.42	0.94	1.56	4.93	4.18	3.83	0.35	0.56	4.74
C4	1.99	3.50	5.48	3.56	2.63	0.93	1.58	5.14	3.27	2.91	0.35	0.56	3.82
C5	2.04	3.39	5.42	3.59	2.66	0.93	1.56	5.15	3.28	2.92	0.37	0.56	3.84
C6	2.32	3.37	5.70	3.61	2.68	0.94	1.58	5.20	3.17	2.81	0.36	0.55	3.72
C7	2.24	3.40	5.64	3.68	2.75	0.93	1.61	5.29	3.47	3.12	0.34	0.55	4.02
C8	2.24	3.37	5.61	3.29	2.35	0.94	1.55	4.84	3.03	2.68	0.35	0.56	3.59
C9	2.18	3.38	5.56	2.90	1.98	0.92	1.56	4.46	3.07	2.71	0.36	0.55	3.62
C10	2.16	3.40	5.55	3.76	2.84	0.91	1.56	5.32	3.47	3.12	0.34	0.56	4.02
C12	2.15	3.39	5.53	3.56	2.63	0.93	1.58	5.14	3.06	2.71	0.34	0.55	3.61
C13_1	2.18	3.35	5.52	3.70	2.76	0.94	1.57	5.27	3.34	2.87	0.46	0.56	3.89
C13_2	2.18	3.36	5.54	3.70	2.76	0.95	1.56	5.26	3.33	2.87	0.46	0.56	3.89
C14	2.24	3.36	5.59	3.50	2.58	0.92	1.55	5.05	3.17	2.81	0.35	0.55	3.72
C15	2.13	3.36	5.49	3.73	2.81	0.92	1.56	5.29	3.14	2.79	0.35	0.55	3.70
C16	2.58	3.36	5.94	3.74	1.87	1.88	1.58	5.32	3.17	2.71	0.46	0.55	3.72
C17				4.42	3.52	0.90	1.53	5.95	3.54	3.18	0.35	0.56	4.09

Table D.44 Insert time of 5 MB. data size, 20 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.15	3.38	5.53	3.76	2.76	1.00	1.56	5.33	4.14	3.79	0.35	0.54	4.67
C3	2.34	3.41	5.75	3.38	2.43	0.95	1.56	4.94	3.46	3.11	0.35	0.54	4.00
C4	2.01	3.46	5.47	3.57	2.64	0.93	1.59	5.16	3.25	2.89	0.36	0.54	3.79
C5	2.05	3.46	5.51	3.60	2.67	0.93	1.56	5.16	3.08	2.72	0.36	0.54	3.62
C6	2.34	3.39	5.73	3.61	2.67	0.93	1.58	5.19	3.03	2.67	0.36	0.53	3.56
C7	2.26	3.39	5.65	3.69	2.76	0.93	1.57	5.26	3.22	2.88	0.34	0.53	3.76
C8	2.24	3.38	5.62	3.29	2.36	0.93	1.58	4.87	2.81	2.47	0.34	0.54	3.35
C9	2.19	3.36	5.55	2.92	1.99	0.93	1.59	4.51	2.89	2.54	0.35	0.53	3.43
C10	2.18	3.34	5.52	3.75	2.83	0.92	1.57	5.33	3.11	2.76	0.35	0.54	3.64
C12	2.16	3.36	5.52	3.57	2.64	0.93	1.59	5.16	3.01	2.67	0.34	0.54	3.54
C13_1	2.19	3.42	5.61	3.71	2.76	0.95	1.57	5.29	3.07	2.63	0.44	0.54	3.61
C13_2	2.19	3.42	5.61	3.71	2.76	0.95	1.55	5.26	3.06	2.63	0.44	0.54	3.60
C14	3.52	3.39	6.90	3.51	2.59	0.92	1.55	5.06	2.94	2.59	0.35	0.53	3.48
C15	2.13	3.46	5.60	3.74	2.82	0.93	1.59	5.33	3.04	2.68	0.35	0.54	3.57
C16	2.59	3.42	6.01	3.00	1.87	1.13	1.57	4.57	3.11	2.65	0.45	0.53	3.64
C17				4.44	3.53	0.91	1.62	6.05	3.50	3.15	0.35	0.54	4.04

Table D.45 Insert time of 5 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.16	3.45	5.60	3.78	2.77	1.01	1.57	5.34	3.89	3.55	0.35	0.53	4.42
C3	2.35	3.43	5.78	3.39	2.43	0.95	1.58	4.97	3.92	3.58	0.34	0.53	4.45
C4	2.03	3.41	5.44	3.58	2.64	0.94	1.59	5.17	3.14	2.79	0.35	0.52	3.66
C5	2.06	3.33	5.39	3.60	2.67	0.93	1.58	5.18	3.09	2.73	0.36	0.52	3.61
C6	2.35	3.35	5.70	3.61	2.68	0.93	1.58	5.19	2.92	2.57	0.35	0.53	3.44
C7	2.26	3.38	5.65	3.69	2.76	0.93	1.59	5.28	3.08	2.74	0.34	0.53	3.60
C8	2.26	3.44	5.70	3.31	2.36	0.95	1.57	4.88	2.67	2.34	0.34	0.53	3.20
C9	2.20	3.40	5.60	2.92	1.99	0.93	1.60	4.53	2.82	2.48	0.34	0.53	3.35
C10	2.19	3.41	5.60	3.76	2.83	0.93	1.59	5.35	2.98	2.63	0.35	0.52	3.50
C12	2.17	3.38	5.55	3.58	2.64	0.94	1.60	5.18	2.86	2.52	0.34	0.53	3.39
C13_1	2.19	3.37	5.56	3.72	2.77	0.95	1.61	5.32	2.92	2.49	0.43	0.53	3.45
C13_2	2.20	3.36	5.56	3.72	2.77	0.95	1.58	5.30	2.92	2.49	0.44	0.52	3.44
C14	6.02	3.41	9.42	3.53	2.61	0.93	1.58	5.11	2.80	2.45	0.35	0.52	3.32
C15	2.15	3.42	5.57	3.76	2.82	0.94	1.58	5.34	2.85	2.51	0.34	0.52	3.37
C16	2.59	3.39	5.98	3.00	1.88	1.13	1.61	4.62	2.97	2.52	0.45	0.53	3.50
C17				4.44	3.53	0.91	1.61	6.05	3.33	2.98	0.35	0.52	3.85

Table D.46 Insert time of 5 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.18	3.50	5.68	3.78	2.77	1.01	1.58	5.36	3.86	3.51	0.35	0.49	4.35
C3	2.36	3.47	5.83	3.39	2.44	0.96	1.58	4.98	3.86	3.51	0.35	0.48	4.33
C4	2.04	3.44	5.48	3.59	2.65	0.94	1.59	5.18	2.89	2.55	0.34	0.48	3.38
C5	2.08	3.39	5.46	3.61	2.67	0.93	1.60	5.21	2.90	2.55	0.35	0.47	3.37
C6	2.37	3.35	5.73	3.63	2.68	0.95	1.58	5.21	2.68	2.33	0.35	0.49	3.17
C7	2.28	3.37	5.65	3.71	2.76	0.94	1.58	5.29	2.92	2.58	0.34	0.48	3.40
C8	2.27	3.46	5.73	3.31	2.37	0.95	1.59	4.90	2.46	2.12	0.33	0.48	2.94
C9	2.22	3.44	5.66	2.93	2.00	0.93	1.59	4.53	2.57	2.23	0.34	0.48	3.05
C10	2.21	3.41	5.63	3.76	2.84	0.92	1.60	5.36	2.79	2.46	0.34	0.48	3.28
C12	2.18	3.39	5.56	3.58	2.65	0.93	1.58	5.16	2.71	2.37	0.34	0.49	3.20
C13_1	2.22	3.39	5.61	3.73	2.77	0.96	1.61	5.34	2.69	2.25	0.43	0.48	3.17
C13_2	2.21	3.34	5.56	3.74	2.77	0.96	1.62	5.36	2.68	2.25	0.43	0.49	3.17
C14	10.82	3.39	14.21	3.54	2.61	0.93	1.59	5.13	2.65	2.31	0.34	0.48	3.13
C15	2.16	3.38	5.55	3.77	2.83	0.94	1.58	5.35	2.71	2.37	0.34	0.48	3.19
C16	2.62	3.39	6.00	3.01	1.88	1.13	1.60	4.61	2.82	2.38	0.45	0.49	3.31
C17				4.46	3.54	0.92	1.61	6.07	3.17	2.83	0.34	0.48	3.65

II) Detail-Table for data size 5 MB. in warm cache

Table D.47 Replace time of 5 MB. data size, 10 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.79	3.27	8.06	1.51	0.52	0.99	1.57	3.08	1.40	1.05	0.35	0.55	1.95
C3	1.93	3.27	5.20	1.71	0.71	0.99	1.57	3.27	1.59	1.24	0.35	0.55	2.14
C4	3.25	3.30	6.55	1.44	0.55	0.89	1.57	3.01	0.91	0.55	0.35	0.56	1.46
C5	1.42	3.29	4.71	1.44	0.56	0.88	1.58	3.02	0.91	0.55	0.36	0.55	1.46
C6	1.73	3.28	5.00	1.45	0.58	0.87	1.61	3.06	1.13	0.78	0.36	0.55	1.69
C7	3.42	3.29	6.71	1.55	0.67	0.88	1.58	3.13	0.98	0.65	0.34	0.55	1.53
C8	1.62	3.26	4.88	1.47	0.61	0.86	1.58	3.05	0.93	0.58	0.34	0.56	1.48
C9	1.53	3.27	4.79	1.41	0.55	0.86	1.60	3.00	0.96	0.60	0.36	0.56	1.52
C10	1.99	3.23	5.22	1.50	0.63	0.86	1.58	3.07	1.05	0.69	0.35	0.56	1.60
C12	1.52	3.24	4.76	1.44	0.56	0.87	1.58	3.02	0.95	0.60	0.35	0.55	1.50
C14	2.11	3.28	5.39	1.43	0.58	0.85	1.58	3.01	0.90	0.57	0.34	0.55	1.45
C15	2.83	3.27	6.10	1.43	0.57	0.87	1.56	2.99	0.90	0.54	0.35	0.56	1.45
C16	2.05	3.25	5.30	1.70	0.57	1.12	1.59	3.28	1.20	0.73	0.46	0.56	1.75
C17				2.62	1.77	0.85	1.57	4.19	1.34	0.98	0.36	0.56	1.90

Table D.48 Replace time of 5 MB. data size, 20 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.57	3.28	4.85	1.53	0.53	1.00	1.57	3.10	1.38	1.04	0.35	0.54	1.92
C3	1.99	3.26	5.25	1.72	0.72	1.00	1.56	3.29	1.58	1.23	0.35	0.54	2.12
C4	3.43	3.26	6.69	1.46	0.56	0.90	1.57	3.04	0.90	0.55	0.35	0.54	1.43
C5	1.49	3.30	4.78	1.45	0.56	0.89	1.57	3.02	0.90	0.54	0.36	0.54	1.43
C6	1.77	3.30	5.07	1.47	0.59	0.88	1.58	3.04	1.05	0.69	0.36	0.54	1.58
C7	3.49	3.29	6.79	1.57	0.69	0.89	1.59	3.17	0.98	0.64	0.34	0.53	1.51
C8	1.66	3.28	4.93	1.50	0.62	0.88	1.61	3.11	0.92	0.58	0.34	0.54	1.45
C9	1.57	3.28	4.85	1.43	0.56	0.87	1.58	3.01	0.95	0.59	0.36	0.54	1.48
C10	1.99	3.27	5.26	1.52	0.64	0.87	1.58	3.09	1.04	0.69	0.35	0.53	1.58
C12	1.59	3.28	4.87	1.45	0.57	0.88	1.58	3.03	0.95	0.60	0.35	0.54	1.48
C14	3.26	3.26	6.52	1.47	0.61	0.86	1.58	3.05	0.94	0.60	0.34	0.54	1.48
C15	2.93	3.23	6.17	1.45	0.58	0.87	1.60	3.05	0.88	0.54	0.34	0.54	1.42
C16	2.10	3.26	5.35	1.73	0.59	1.14	1.59	3.32	1.18	0.73	0.45	0.53	1.71
C17				2.65	1.79	0.86	1.55	4.20	1.33	0.98	0.35	0.53	1.86

Table D.49 Replace time of 5 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.61	3.23	4.84	1.54	0.54	1.00	1.58	3.12	1.37	1.03	0.35	0.52	1.90
C3	2.02	3.24	5.26	1.76	0.76	1.01	1.58	3.34	1.57	1.23	0.34	0.53	2.10
C4	3.56	3.27	6.82	1.50	0.59	0.90	1.58	3.08	0.89	0.55	0.34	0.53	1.41
C5	1.50	3.27	4.76	1.45	0.57	0.89	1.59	3.04	0.89	0.54	0.36	0.52	1.42
C6	1.80	3.31	5.10	1.48	0.59	0.89	1.58	3.06	1.01	0.66	0.35	0.53	1.54
C7	3.53	3.28	6.81	1.59	0.69	0.90	1.58	3.17	0.96	0.63	0.33	0.53	1.49
C8	1.69	3.27	4.96	1.53	0.65	0.88	1.57	3.10	0.92	0.58	0.34	0.52	1.44
C9	1.59	3.24	4.83	1.44	0.56	0.88	1.58	3.02	0.94	0.59	0.35	0.53	1.47
C10	2.03	3.26	5.29	1.53	0.66	0.88	1.55	3.08	1.03	0.69	0.35	0.53	1.56
C12	1.60	3.27	4.86	1.46	0.58	0.88	1.59	3.05	0.93	0.59	0.34	0.53	1.46
C14	5.47	3.29	8.76	1.50	0.63	0.86	1.59	3.09	0.93	0.59	0.33	0.53	1.46
C15	3.06	3.31	6.37	1.47	0.59	0.88	1.55	3.01	0.88	0.54	0.34	0.53	1.41
C16	2.29	3.29	5.58	1.77	0.60	1.16	1.60	3.37	1.17	0.72	0.45	0.53	1.70
C17				2.66	1.79	0.87	1.58	4.23	1.34	0.99	0.35	0.53	1.86

Table D.50 Replace time of 5 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.64	3.30	4.94	1.55	0.54	1.01	1.57	3.12	1.36	1.02	0.34	0.50	1.86
C3	2.06	3.33	5.39	1.78	0.77	1.01	1.55	3.33	1.57	1.22	0.34	0.49	2.06
C4	3.58	3.39	6.97	1.51	0.60	0.91	1.57	3.08	0.88	0.54	0.34	0.49	1.37
C5	1.51	3.32	4.83	1.47	0.57	0.90	1.59	3.06	0.88	0.53	0.35	0.49	1.37
C6	1.80	3.34	5.13	1.49	0.60	0.89	1.58	3.06	0.97	0.63	0.34	0.50	1.47
C7	3.59	3.32	6.91	1.60	0.70	0.90	1.56	3.16	0.95	0.63	0.33	0.50	1.45
C8	1.70	3.32	5.02	1.53	0.65	0.88	1.57	3.10	0.90	0.57	0.34	0.49	1.39
C9	1.60	3.32	4.92	1.46	0.57	0.89	1.57	3.04	0.93	0.58	0.35	0.49	1.43
C10	2.06	3.28	5.34	1.55	0.67	0.88	1.59	3.15	1.03	0.69	0.34	0.48	1.51
C12	1.60	3.28	4.88	1.48	0.59	0.88	1.57	3.04	0.92	0.59	0.34	0.49	1.41
C14	9.67	3.27	12.94	1.51	0.65	0.86	1.57	3.08	0.91	0.59	0.33	0.49	1.40
C15	3.28	3.28	6.56	1.48	0.60	0.88	1.56	3.04	0.87	0.53	0.34	0.49	1.36
C16	2.32	3.31	5.63	1.78	0.61	1.17	1.54	3.32	1.16	0.71	0.45	0.49	1.65
C17				2.68	1.80	0.88	1.58	4.26	1.32	0.98	0.34	0.50	1.82

Table D.51 Delete time of 5 MB. data size, 10 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.54	3.30	4.83	1.51	0.52	0.99	1.57	3.08	1.38	1.02	0.35	0.56	1.93
C3	1.48	3.35	4.83	1.72	0.71	1.01	1.57	3.29	1.60	1.24	0.36	0.56	2.16
C4	1.53	3.28	4.81	1.43	0.55	0.87	1.59	3.01	0.91	0.56	0.35	0.55	1.46
C5	1.42	3.26	4.68	1.42	0.56	0.87	1.56	2.98	0.91	0.55	0.36	0.55	1.47
C6	1.68	3.31	4.99	1.46	0.58	0.88	1.56	3.01	1.13	0.78	0.36	0.56	1.69
C7	1.59	3.30	4.89	1.57	0.68	0.89	1.60	3.17	0.98	0.64	0.34	0.56	1.54
C8	1.58	3.30	4.87	1.49	0.62	0.87	1.58	3.07	0.94	0.59	0.35	0.55	1.49
C9	1.52	3.31	4.83	1.42	0.55	0.87	1.57	2.99	0.96	0.59	0.36	0.55	1.51
C10	1.49	3.27	4.77	1.49	0.63	0.86	1.59	3.08	1.06	0.70	0.36	0.55	1.61
C12	1.49	3.25	4.75	1.44	0.57	0.86	1.56	3.00	0.94	0.60	0.35	0.55	1.49
C14	2.10	3.25	5.34	1.43	0.58	0.85	1.58	3.00	0.92	0.57	0.35	0.55	1.47
C15	1.46	3.31	4.76	1.42	0.56	0.86	1.56	2.98	0.90	0.55	0.35	0.56	1.45
C16	1.98	3.32	5.29	1.69	0.57	1.12	1.56	3.26	1.18	0.72	0.45	0.56	1.73
C17				2.61	1.77	0.84	1.56	4.17	1.32	0.97	0.35	0.55	1.87

Table D.52 Delete time of 5 MB. data size, 20 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.55	3.30	4.85	1.52	0.53	0.99	1.05	2.57	1.37	1.02	0.35	0.54	1.91
C3	1.49	3.29	4.77	1.71	0.71	1.00	1.04	2.75	1.59	1.24	0.35	0.54	2.13
C4	1.54	3.28	4.81	1.46	0.57	0.90	1.05	2.51	0.90	0.55	0.35	0.54	1.43
C5	1.43	3.27	4.71	1.46	0.56	0.89	1.03	2.49	0.90	0.54	0.36	0.54	1.44
C6	1.69	3.28	4.97	1.48	0.59	0.89	1.05	2.53	1.07	0.71	0.36	0.54	1.61
C7	1.66	3.33	4.99	1.57	0.69	0.88	1.06	2.63	0.96	0.64	0.33	0.54	1.50
C8	1.60	3.32	4.91	1.50	0.63	0.87	1.07	2.57	0.94	0.59	0.35	0.54	1.48
C9	1.54	3.29	4.83	1.42	0.56	0.86	1.03	2.46	0.95	0.59	0.36	0.54	1.48
C10	1.52	3.37	4.88	1.51	0.65	0.87	1.04	2.55	1.05	0.70	0.35	0.54	1.59
C12	1.52	3.29	4.80	1.45	0.57	0.87	1.05	2.50	0.95	0.60	0.35	0.53	1.48
C14	3.26	3.31	6.57	1.48	0.62	0.86	1.05	2.52	0.90	0.57	0.34	0.54	1.44
C15	1.46	3.29	4.75	1.45	0.58	0.87	1.06	2.51	0.88	0.54	0.34	0.53	1.42
C16	2.00	3.30	5.30	1.75	0.60	1.15	1.05	2.80	1.18	0.72	0.46	0.54	1.71
C17				2.65	1.79	0.86	1.04	3.69	1.31	0.97	0.34	0.54	1.85

Table D.53 Delete time of 5 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.59	3.23	4.82	1.54	0.54	1.00	1.58	3.12	1.36	1.01	0.35	0.53	1.89
C3	1.53	3.27	4.79	1.77	0.76	1.01	1.55	3.32	1.58	1.23	0.35	0.52	2.11
C4	1.56	3.29	4.85	1.50	0.59	0.90	1.54	3.04	0.89	0.55	0.35	0.53	1.42
C5	1.48	3.35	4.83	1.46	0.57	0.90	1.55	3.01	0.89	0.54	0.35	0.53	1.42
C6	1.72	3.31	5.04	1.49	0.60	0.89	1.55	3.04	1.04	0.69	0.35	0.53	1.57
C7	1.68	3.33	5.02	1.58	0.70	0.88	1.58	3.15	0.97	0.63	0.34	0.53	1.50
C8	1.63	3.28	4.91	1.52	0.65	0.87	1.59	3.12	0.92	0.58	0.34	0.53	1.45
C9	1.55	3.31	4.86	1.44	0.56	0.88	1.56	3.00	0.94	0.58	0.36	0.53	1.47
C10	1.53	3.27	4.80	1.53	0.66	0.87	1.55	3.08	1.04	0.69	0.35	0.53	1.57
C12	1.54	3.32	4.85	1.47	0.57	0.90	1.57	3.04	0.93	0.59	0.34	0.53	1.45
C14	5.38	3.29	8.67	1.48	0.62	0.85	1.59	3.07	0.90	0.56	0.34	0.53	1.43
C15	1.48	3.30	4.78	1.47	0.59	0.88	1.59	3.06	0.88	0.54	0.34	0.53	1.41
C16	2.12	3.25	5.37	1.77	0.61	1.16	1.55	3.32	1.17	0.72	0.45	0.53	1.70
C17				2.65	1.79	0.87	1.56	4.21	1.32	0.97	0.35	0.53	1.85

Table D.54 Delete time of 5 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.60	3.32	4.92	1.54	0.54	1.00	1.58	3.12	1.35	1.00	0.34	0.50	1.84
C3	1.54	3.31	4.85	1.77	0.77	1.00	1.54	3.32	1.57	1.23	0.35	0.49	2.07
C4	1.60	3.34	4.95	1.52	0.61	0.91	1.55	3.07	0.88	0.54	0.34	0.49	1.38
C5	1.53	3.34	4.87	1.46	0.58	0.88	1.56	3.02	0.89	0.54	0.35	0.49	1.38
C6	1.73	3.29	5.03	1.50	0.61	0.89	1.58	3.08	1.00	0.65	0.35	0.49	1.49
C7	1.71	3.28	4.99	1.60	0.70	0.90	1.59	3.19	0.97	0.63	0.34	0.49	1.46
C8	1.66	3.28	4.94	1.53	0.64	0.89	1.59	3.12	0.91	0.58	0.33	0.49	1.40
C9	1.59	3.31	4.90	1.46	0.57	0.89	1.57	3.04	0.93	0.58	0.35	0.49	1.43
C10	1.55	3.31	4.86	1.53	0.66	0.88	1.56	3.10	1.04	0.69	0.35	0.50	1.54
C12	1.55	3.32	4.87	1.47	0.59	0.88	1.55	3.02	0.91	0.58	0.33	0.49	1.41
C14	5.39	3.34	8.72	1.50	0.64	0.86	1.58	3.08	0.89	0.56	0.33	0.50	1.39
C15	1.49	3.32	4.81	1.47	0.59	0.89	1.59	3.06	0.87	0.53	0.34	0.49	1.36
C16	2.18	3.32	5.50	1.77	0.60	1.17	1.58	3.35	1.16	0.71	0.44	0.49	1.65
C17				2.68	1.81	0.87	1.58	4.26	1.31	0.96	0.35	0.50	1.81

Table D.55 Insert time of 5 MB. data size, 10 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.55	3.38	4.93	1.53	0.53	0.99	1.58	3.11	1.40	1.04	0.35	0.56	1.96
C3	1.49	3.35	4.84	1.70	0.70	1.00	1.58	3.27	1.60	1.24	0.35	0.56	2.16
C4	1.53	3.30	4.83	1.44	0.56	0.88	1.57	3.01	0.91	0.56	0.35	0.55	1.46
C5	1.43	3.32	4.74	1.44	0.57	0.87	1.56	3.00	0.91	0.55	0.36	0.55	1.46
C6	1.69	3.29	4.97	1.46	0.59	0.88	1.59	3.05	1.14	0.78	0.36	0.56	1.70
C7	1.60	3.30	4.90	1.55	0.68	0.88	1.57	3.13	1.26	0.91	0.34	0.55	1.81
C8	1.59	3.29	4.88	1.47	0.60	0.87	1.57	3.04	0.97	0.65	0.33	0.56	1.53
C9	1.53	3.27	4.80	1.41	0.55	0.86	1.58	2.98	0.93	0.59	0.34	0.56	1.49
C10	1.49	3.30	4.80	1.48	0.63	0.86	1.57	3.06	1.02	0.69	0.33	0.56	1.58
C12	1.50	3.27	4.77	1.44	0.57	0.86	1.57	3.00	0.93	0.60	0.33	0.56	1.48
C13_1	1.58	3.26	4.85	1.45	0.59	0.86	1.56	3.01	1.27	0.69	0.58	0.56	1.83
C13_2	1.60	3.28	4.88	1.45	0.59	0.86	1.56	3.01	1.27	0.69	0.59	0.56	1.83
C14	2.11	3.28	5.39	1.44	0.59	0.85	1.57	3.01	0.91	0.57	0.35	0.56	1.47
C15	1.46	3.29	4.75	1.43	0.57	0.86	1.57	3.00	0.89	0.55	0.34	0.55	1.44
C16	1.99	3.30	5.28	1.16	0.58	0.59	1.58	2.74	1.20	0.74	0.46	0.55	1.76
C17				3.55	1.77	1.78	1.58	5.13	1.33	0.98	0.35	0.55	1.89

Table D.56 Insert time of 5 MB. data size, 20 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.55	3.28	4.83	1.54	0.54	1.00	1.05	2.59	1.39	1.04	0.35	0.54	1.93
C3	1.49	3.29	4.78	1.73	0.73	1.00	1.03	2.75	1.60	1.24	0.36	0.54	2.13
C4	1.57	3.31	4.88	1.44	0.56	0.89	1.06	2.50	0.91	0.55	0.35	0.54	1.44
C5	1.44	3.28	4.71	1.43	0.55	0.87	1.05	2.48	0.89	0.54	0.35	0.54	1.43
C6	1.70	3.29	4.99	1.46	0.59	0.87	1.07	2.53	1.07	0.72	0.35	0.53	1.61
C7	1.66	3.32	4.98	1.57	0.68	0.89	1.05	2.62	0.99	0.64	0.35	0.53	1.52
C8	1.61	3.29	4.89	1.51	0.63	0.88	1.04	2.55	0.90	0.57	0.33	0.54	1.44
C9	1.53	3.33	4.86	1.42	0.55	0.87	1.05	2.47	0.92	0.59	0.33	0.54	1.46
C10	1.51	3.28	4.79	1.51	0.65	0.86	1.05	2.56	1.02	0.69	0.33	0.54	1.55
C12	1.51	3.34	4.85	1.45	0.58	0.87	1.06	2.51	0.93	0.60	0.33	0.54	1.47
C13_1	1.59	3.29	4.89	1.45	0.59	0.86	1.05	2.50	1.26	0.69	0.57	0.53	1.79
C13_2	1.60	3.28	4.88	1.46	0.59	0.87	1.05	2.51	1.26	0.68	0.58	0.53	1.79
C14	3.26	3.29	6.55	1.45	0.59	0.85	1.06	2.50	0.91	0.57	0.35	0.54	1.45
C15	1.51	3.31	4.81	1.44	0.58	0.86	1.07	2.51	0.87	0.54	0.33	0.54	1.41
C16	1.99	3.27	5.27	1.72	0.59	1.13	1.04	2.76	1.19	0.73	0.46	0.54	1.73
C17				2.64	1.78	0.85	1.05	3.68	1.33	0.98	0.35	0.54	1.86

Table D.57 Insert time of 5 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.59	3.31	4.91	1.53	0.54	1.00	1.59	3.12	1.38	1.03	0.35	0.53	1.91
C3	1.54	3.35	4.89	1.75	0.75	1.00	1.58	3.33	1.59	1.23	0.35	0.53	2.11
C4	1.59	3.30	4.88	1.49	0.60	0.89	1.58	3.08	0.90	0.55	0.35	0.53	1.42
C5	1.49	3.29	4.78	1.46	0.57	0.88	1.58	3.04	0.88	0.54	0.35	0.53	1.41
C6	1.73	3.31	5.04	1.48	0.60	0.88	1.58	3.06	1.04	0.69	0.35	0.53	1.56
C7	1.70	3.32	5.02	1.59	0.70	0.89	1.60	3.19	0.98	0.64	0.34	0.53	1.51
C8	1.64	3.27	4.91	1.53	0.65	0.89	1.59	3.12	0.90	0.57	0.33	0.53	1.43
C9	1.57	3.27	4.84	1.45	0.56	0.89	1.58	3.03	0.92	0.59	0.34	0.53	1.45
C10	1.54	3.27	4.81	1.53	0.65	0.88	1.58	3.11	1.00	0.68	0.32	0.52	1.53
C12	1.55	3.27	4.82	1.47	0.59	0.88	1.58	3.05	0.91	0.59	0.32	0.52	1.44
C13_1	1.56	3.27	4.82	1.47	0.60	0.87	1.57	3.04	1.25	0.68	0.58	0.53	1.78
C13_2	1.56	3.31	4.87	1.47	0.60	0.87	1.59	3.05	1.25	0.67	0.58	0.53	1.78
C14	5.39	3.29	8.68	1.50	0.63	0.86	1.58	3.08	0.89	0.56	0.34	0.52	1.42
C15	1.49	3.30	4.79	1.46	0.58	0.88	1.58	3.03	0.88	0.54	0.34	0.52	1.40
C16	2.13	3.29	5.41	1.74	0.60	1.14	1.57	3.31	1.18	0.73	0.46	0.53	1.71
C17				2.68	1.79	0.89	1.58	4.26	1.32	0.97	0.34	0.53	1.84

Table D.58 Insert time of 5 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.61	3.28	4.88	1.55	0.55	1.00	1.57	3.12	1.38	1.03	0.35	0.48	1.86
C3	1.54	3.30	4.84	1.77	0.77	1.01	1.57	3.34	1.58	1.23	0.35	0.49	2.07
C4	1.60	3.26	4.85	1.50	0.61	0.89	1.59	3.08	0.89	0.55	0.35	0.49	1.38
C5	1.50	3.25	4.75	1.47	0.58	0.89	1.55	3.02	0.88	0.53	0.34	0.48	1.36
C6	1.74	3.30	5.04	1.50	0.61	0.88	1.55	3.04	1.00	0.65	0.35	0.48	1.49
C7	1.71	3.25	4.96	1.59	0.70	0.89	1.58	3.16	0.97	0.63	0.34	0.49	1.46
C8	1.65	3.27	4.91	1.54	0.64	0.89	1.58	3.12	0.89	0.57	0.32	0.48	1.37
C9	1.59	3.25	4.84	1.46	0.57	0.90	1.58	3.04	0.91	0.58	0.34	0.49	1.40
C10	1.56	3.31	4.87	1.54	0.67	0.88	1.58	3.12	1.00	0.68	0.33	0.49	1.49
C12	1.56	3.29	4.85	1.47	0.59	0.88	1.58	3.04	0.91	0.59	0.32	0.49	1.40
C13_1	1.58	3.27	4.84	1.48	0.61	0.87	1.58	3.06	1.25	0.67	0.57	0.49	1.73
C13_2	1.57	3.29	4.86	1.49	0.61	0.88	1.59	3.08	1.24	0.67	0.57	0.49	1.73
C14	5.39	3.29	8.69	1.53	0.65	0.87	1.56	3.08	0.89	0.55	0.33	0.48	1.37
C15	1.50	3.29	4.79	1.48	0.59	0.89	1.57	3.05	0.86	0.53	0.33	0.48	1.35
C16	2.18	3.30	5.48	1.78	0.61	1.16	1.55	3.33	1.17	0.72	0.45	0.48	1.65
C17				2.68	1.81	0.88	1.58	4.27	1.31	0.97	0.34	0.49	1.80

III) Detail-Table for data size 5 MB. in hot cache

Table D.59 Replace time of 5 MB. data size, 10 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.55	3.25	4.79	1.74	0.59	1.15	1.55	3.30	0.74	0.44	0.30	0.56	1.30
C3	1.93	3.29	5.22	1.82	0.67	1.15	1.55	3.37	0.93	0.62	0.31	0.55	1.49
C4	3.25	3.27	6.51	1.44	0.55	0.89	1.56	3.01	0.87	0.55	0.31	0.55	1.42
C5	1.42	3.26	4.68	1.45	0.56	0.89	1.55	3.01	1.01	0.63	0.38	0.56	1.57
C6	1.72	3.30	5.02	1.44	0.56	0.89	1.55	3.00	0.95	0.63	0.32	0.56	1.51
C7	3.42	3.29	6.72	1.49	0.59	0.90	1.56	3.05	0.94	0.64	0.30	0.56	1.49
C8	1.63	3.30	4.93	1.45	0.56	0.89	1.56	3.01	0.93	0.63	0.30	0.55	1.48
C9	1.52	3.29	4.82	1.45	0.56	0.89	1.56	3.01	0.88	0.55	0.33	0.55	1.44
C10	1.98	3.27	5.25	1.44	0.55	0.88	1.56	2.99	0.93	0.64	0.30	0.56	1.49
C12	1.52	3.32	4.85	1.44	0.56	0.89	1.56	3.01	0.87	0.56	0.31	0.55	1.42
C14	2.11	3.34	5.44	1.43	0.56	0.88	1.57	3.00	0.85	0.55	0.30	0.55	1.40
C15	2.82	3.34	6.16	1.44	0.56	0.89	1.57	3.01	0.89	0.59	0.30	0.56	1.45
C16	2.03	3.29	5.32	1.62	0.55	1.06	1.56	3.18	1.11	0.65	0.46	0.55	1.66
C17				2.10	1.22	0.88	1.56	3.66	0.95	0.58	0.37	0.56	1.51

Table D.60 Replace time of 5 MB. data size, 20 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.57	3.26	4.83	1.75	0.59	1.16	1.56	3.31	0.74	0.44	0.30	0.53	1.27
C3	1.98	3.33	5.32	2.34	1.19	1.15	1.55	3.89	0.92	0.62	0.30	0.54	1.46
C4	3.42	3.32	6.75	1.56	0.68	0.89	1.55	3.11	0.85	0.55	0.30	0.54	1.39
C5	1.49	3.36	4.84	1.46	0.56	0.90	1.56	3.03	1.00	0.63	0.37	0.54	1.54
C6	1.77	3.32	5.09	1.45	0.56	0.89	1.55	3.01	0.94	0.63	0.31	0.54	1.48
C7	3.50	3.31	6.82	1.46	0.57	0.90	1.56	3.03	0.93	0.63	0.30	0.54	1.47
C8	1.66	3.30	4.95	1.46	0.56	0.90	1.54	3.00	0.93	0.63	0.30	0.54	1.46
C9	1.58	3.33	4.90	1.46	0.57	0.89	1.54	3.00	0.88	0.55	0.33	0.54	1.41
C10	1.99	3.38	5.36	1.45	0.56	0.89	1.58	3.03	0.92	0.63	0.29	0.53	1.46
C12	1.59	3.29	4.88	1.46	0.56	0.90	1.55	3.01	0.86	0.55	0.31	0.53	1.39
C14	3.25	3.31	6.55	1.45	0.56	0.88	1.55	3.00	0.85	0.55	0.30	0.54	1.38
C15	2.93	3.34	6.27	1.46	0.56	0.90	1.58	3.04	0.89	0.59	0.30	0.54	1.42
C16	2.10	3.33	5.43	1.63	0.56	1.07	1.58	3.21	1.10	0.64	0.45	0.53	1.63
C17				2.11	1.22	0.89	1.58	3.68	0.94	0.57	0.37	0.53	1.47

Table D.61 Replace time of 5 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.62	3.39	5.01	1.76	0.60	1.16	1.58	3.34	0.73	0.44	0.30	0.53	1.26
C3	2.04	3.38	5.42	3.53	2.38	1.16	1.56	5.09	0.91	0.61	0.30	0.52	1.44
C4	3.54	3.31	6.85	2.08	1.19	0.90	1.58	3.66	0.85	0.55	0.30	0.53	1.37
C5	1.49	3.28	4.76	1.58	0.68	0.90	1.56	3.13	0.99	0.63	0.37	0.53	1.52
C6	1.80	3.26	5.06	1.46	0.56	0.90	1.56	3.03	0.93	0.62	0.31	0.53	1.45
C7	3.52	3.23	6.75	1.47	0.57	0.89	1.57	3.04	0.92	0.63	0.29	0.53	1.45
C8	1.68	3.26	4.93	1.46	0.57	0.90	1.57	3.04	0.92	0.62	0.30	0.53	1.46
C9	1.59	3.31	4.91	1.47	0.57	0.90	1.58	3.05	0.86	0.54	0.32	0.53	1.39
C10	2.02	3.26	5.27	1.46	0.56	0.89	1.58	3.04	0.92	0.63	0.29	0.53	1.45
C12	1.60	3.28	4.87	1.46	0.57	0.89	1.56	3.02	0.85	0.55	0.30	0.53	1.37
C14	5.47	3.35	8.82	1.47	0.57	0.90	1.55	3.02	0.84	0.55	0.30	0.53	1.37
C15	3.06	3.32	6.38	1.47	0.57	0.91	1.54	3.01	0.88	0.59	0.29	0.53	1.41
C16	2.29	3.36	5.66	1.64	0.56	1.07	1.55	3.19	1.09	0.64	0.45	0.53	1.62
C17				2.12	1.23	0.89	1.56	3.68	0.93	0.57	0.36	0.52	1.46

Table D.62 Replace time of 5 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.64	3.40	5.04	1.73	0.56	1.16	1.58	3.31	0.73	0.43	0.29	0.49	1.22
C3	2.05	3.37	5.42	6.01	4.84	1.16	1.57	7.58	0.91	0.61	0.30	0.49	1.40
C4	3.58	3.34	6.92	2.10	1.19	0.91	1.56	3.66	0.84	0.54	0.30	0.48	1.32
C5	1.51	3.30	4.82	1.59	0.68	0.90	1.57	3.16	0.98	0.63	0.36	0.49	1.47
C6	1.78	3.24	5.02	1.46	0.57	0.89	1.56	3.02	0.93	0.62	0.31	0.50	1.42
C7	3.58	3.26	6.84	1.49	0.58	0.91	1.58	3.07	0.92	0.62	0.29	0.48	1.40
C8	1.69	3.28	4.98	1.47	0.57	0.90	1.58	3.05	0.92	0.62	0.30	0.49	1.41
C9	1.58	3.27	4.84	1.48	0.58	0.90	1.57	3.05	0.86	0.54	0.32	0.49	1.35
C10	2.12	3.26	5.38	1.46	0.57	0.89	1.57	3.03	0.92	0.63	0.29	0.49	1.40
C12	1.58	3.30	4.88	1.47	0.57	0.90	1.57	3.04	0.85	0.55	0.30	0.49	1.34
C14	9.66	3.35	13.01	1.47	0.57	0.90	1.56	3.03	0.83	0.54	0.29	0.49	1.32
C15	3.28	3.34	6.63	1.47	0.57	0.90	1.58	3.05	0.87	0.58	0.29	0.49	1.37
C16	2.32	3.31	5.62	1.64	0.57	1.08	1.58	3.22	1.08	0.64	0.45	0.49	1.57
C17				2.20	1.30	0.89	1.55	3.75	0.93	0.57	0.36	0.48	1.41

Table D.63 Delete time of 5 MB. data size, 10 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.54	3.33	4.87	1.75	0.59	1.16	1.57	3.32	0.73	0.45	0.29	0.56	1.29
C3	1.48	3.34	4.82	1.81	0.66	1.15	1.56	3.37	0.94	0.63	0.31	0.56	1.50
C4	1.51	3.32	4.84	1.57	0.68	0.89	1.58	3.15	0.87	0.56	0.31	0.56	1.42
C5	1.42	3.28	4.70	1.45	0.56	0.89	1.57	3.01	0.99	0.63	0.37	0.55	1.55
C6	1.69	3.37	5.06	1.45	0.56	0.88	1.56	3.00	0.94	0.62	0.32	0.55	1.49
C7	1.58	3.30	4.88	1.46	0.58	0.89	1.56	3.03	0.95	0.64	0.31	0.56	1.51
C8	1.57	3.35	4.92	1.44	0.55	0.89	1.58	3.02	0.93	0.63	0.30	0.56	1.49
C9	1.52	3.35	4.87	1.45	0.56	0.90	1.57	3.02	0.86	0.56	0.30	0.56	1.42
C10	1.49	3.32	4.80	1.44	0.56	0.88	1.55	2.99	0.94	0.64	0.30	0.56	1.49
C12	1.49	3.31	4.80	1.45	0.56	0.89	1.56	3.01	0.87	0.55	0.32	0.55	1.43
C14	2.11	3.33	5.44	1.43	0.55	0.88	1.56	2.99	0.85	0.55	0.30	0.56	1.41
C15	1.46	3.31	4.77	1.44	0.56	0.88	1.58	3.02	0.89	0.58	0.31	0.56	1.45
C16	1.97	3.31	5.28	1.62	0.56	1.06	1.56	3.18	1.10	0.65	0.45	0.56	1.66
C17				1.74	0.85	0.89	1.58	3.32	0.94	0.57	0.37	0.56	1.49

Table D.64 Delete time of 5 MB. data size, 20 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.56	3.34	4.90	1.75	0.59	1.16	1.57	3.32	0.74	0.44	0.30	0.54	1.28
C3	1.49	3.31	4.80	2.35	1.19	1.16	1.57	3.92	0.94	0.63	0.31	0.54	1.47
C4	1.53	3.33	4.86	1.57	0.68	0.90	1.58	3.15	0.87	0.56	0.31	0.54	1.40
C5	1.41	3.28	4.69	1.46	0.56	0.90	1.57	3.03	0.98	0.62	0.36	0.54	1.52
C6	1.68	3.34	5.02	1.44	0.56	0.88	1.56	3.00	0.94	0.62	0.32	0.54	1.48
C7	1.65	3.37	5.02	1.47	0.59	0.88	1.54	3.01	0.94	0.63	0.30	0.53	1.47
C8	1.59	3.31	4.90	1.46	0.56	0.90	1.56	3.01	0.93	0.63	0.30	0.54	1.46
C9	1.54	3.27	4.81	1.46	0.56	0.90	1.58	3.04	0.85	0.55	0.30	0.54	1.39
C10	1.52	3.33	4.85	1.45	0.56	0.89	1.57	3.01	0.93	0.63	0.29	0.54	1.46
C12	1.52	3.35	4.87	1.47	0.57	0.91	1.56	3.03	0.86	0.55	0.32	0.54	1.40
C14	3.26	3.35	6.61	1.44	0.56	0.88	1.54	2.98	0.84	0.54	0.30	0.53	1.38
C15	1.45	3.37	4.81	1.45	0.56	0.89	1.58	3.03	0.90	0.59	0.30	0.53	1.43
C16	1.99	3.30	5.28	1.63	0.56	1.06	1.57	3.19	1.09	0.65	0.45	0.54	1.63
C17				2.10	1.21	0.89	1.58	3.68	0.94	0.57	0.37	0.54	1.47

Table D.65 Delete time of 5 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.58	3.34	4.93	1.76	0.60	1.16	1.55	3.31	0.74	0.44	0.30	0.53	1.26
C3	1.54	3.35	4.90	3.53	2.37	1.16	1.55	5.08	0.93	0.62	0.31	0.53	1.46
C4	1.56	3.35	4.91	1.58	0.68	0.90	1.55	3.13	0.85	0.54	0.31	0.53	1.38
C5	1.48	3.27	4.75	1.46	0.57	0.90	1.56	3.03	0.98	0.62	0.36	0.53	1.50
C6	1.73	3.26	4.99	1.46	0.57	0.89	1.56	3.02	0.91	0.62	0.29	0.53	1.44
C7	1.66	3.21	4.88	1.45	0.56	0.89	1.53	2.98	0.92	0.63	0.29	0.52	1.45
C8	1.63	3.34	4.98	1.46	0.56	0.90	1.54	3.00	0.92	0.62	0.30	0.53	1.45
C9	1.55	3.28	4.83	1.47	0.56	0.91	1.58	3.04	0.84	0.55	0.29	0.53	1.37
C10	1.52	3.32	4.84	1.45	0.56	0.88	1.54	2.98	0.93	0.63	0.30	0.52	1.46
C12	1.52	3.28	4.81	1.48	0.57	0.91	1.53	3.01	0.85	0.54	0.30	0.52	1.37
C14	5.39	3.29	8.68	1.45	0.56	0.89	1.53	2.98	0.83	0.54	0.30	0.53	1.36
C15	1.48	3.30	4.77	1.47	0.57	0.90	1.58	3.05	0.88	0.59	0.29	0.52	1.40
C16	2.12	3.33	5.44	1.63	0.56	1.07	1.56	3.19	1.10	0.65	0.45	0.53	1.62
C17				2.11	1.22	0.89	1.55	3.66	0.93	0.57	0.36	0.52	1.45

Table D.66 Delete time of 5 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.61	3.25	4.86	1.77	0.60	1.17	1.56	3.32	0.73	0.43	0.30	0.49	1.22
C3	1.53	3.27	4.81	6.03	4.86	1.17	1.56	7.59	0.93	0.62	0.31	0.49	1.41
C4	1.62	3.36	4.98	1.59	0.68	0.91	1.56	3.14	0.85	0.54	0.31	0.48	1.33
C5	1.53	3.37	4.89	1.48	0.57	0.91	1.57	3.05	0.97	0.62	0.35	0.48	1.45
C6	1.74	3.32	5.07	1.46	0.57	0.89	1.55	3.01	0.91	0.62	0.29	0.48	1.39
C7	1.69	3.30	4.99	1.47	0.57	0.90	1.55	3.01	0.93	0.63	0.30	0.49	1.42
C8	1.66	3.37	5.02	1.47	0.56	0.91	1.57	3.04	0.91	0.62	0.29	0.48	1.39
C9	1.58	3.34	4.92	1.48	0.57	0.91	1.58	3.06	0.84	0.55	0.29	0.49	1.33
C10	1.54	3.27	4.81	1.45	0.56	0.89	1.56	3.01	0.92	0.63	0.29	0.49	1.40
C12	1.55	3.29	4.84	1.49	0.57	0.92	1.58	3.07	0.85	0.54	0.31	0.49	1.33
C14	5.38	3.29	8.67	1.45	0.57	0.88	1.56	3.00	0.83	0.53	0.30	0.49	1.32
C15	1.50	3.29	4.79	1.47	0.57	0.89	1.57	3.03	0.88	0.58	0.30	0.49	1.37
C16	2.17	3.32	5.49	1.64	0.57	1.07	1.56	3.20	1.08	0.64	0.44	0.48	1.56
C17				2.11	1.22	0.89	1.56	3.67	0.92	0.56	0.36	0.49	1.41

Table D.67 Insert time of 5 MB. data size, 10 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.55	3.28	4.84	1.76	0.60	1.16	1.57	3.33	0.77	0.44	0.33	0.55	1.33
C3	1.48	3.29	4.76	1.73	0.58	1.15	1.56	3.30	0.93	0.61	0.32	0.56	1.49
C4	1.53	3.36	4.89	1.44	0.56	0.88	1.56	3.00	0.88	0.56	0.32	0.55	1.43
C5	1.43	3.32	4.75	1.45	0.56	0.89	1.57	3.02	1.00	0.64	0.36	0.56	1.55
C6	1.69	3.25	4.94	1.44	0.56	0.88	1.57	3.01	0.93	0.63	0.31	0.56	1.49
C7	1.59	3.37	4.97	1.49	0.59	0.89	1.54	3.03	0.95	0.64	0.30	0.55	1.50
C8	1.59	3.34	4.92	1.45	0.55	0.89	1.57	3.02	0.93	0.63	0.30	0.56	1.49
C9	1.53	3.38	4.91	1.46	0.57	0.89	1.57	3.03	0.89	0.56	0.33	0.56	1.44
C10	1.50	3.35	4.85	1.44	0.56	0.88	1.57	3.01	0.93	0.63	0.30	0.56	1.49
C12	1.51	3.28	4.79	1.44	0.55	0.89	1.56	3.01	0.88	0.55	0.33	0.55	1.43
C13_1	1.58	3.25	4.82	1.46	0.57	0.89	1.56	3.02	1.01	0.54	0.46	0.56	1.56
C13_2	1.59	3.29	4.87	1.47	0.57	0.90	1.59	3.06	1.01	0.54	0.46	0.56	1.57
C14	2.11	3.28	5.39	1.43	0.55	0.88	1.56	2.99	0.85	0.55	0.30	0.56	1.41
C15	1.46	3.27	4.73	1.44	0.56	0.89	1.56	3.01	0.86	0.56	0.30	0.56	1.42
C16	1.99	3.28	5.27	1.62	0.55	1.07	1.56	3.19	1.11	0.65	0.46	0.56	1.67
C17				2.10	1.22	0.88	1.57	3.67	0.94	0.57	0.37	0.55	1.49

Table D.68 Insert time of 5 MB. data size, 20 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.55	3.38	4.93	1.77	0.60	1.17	1.56	3.33	0.76	0.44	0.32	0.54	1.30
C3	1.50	3.34	4.83	1.74	0.59	1.15	1.57	3.31	0.93	0.61	0.32	0.54	1.47
C4	1.56	3.32	4.87	1.45	0.56	0.89	1.57	3.02	0.87	0.55	0.32	0.54	1.41
C5	1.42	3.26	4.68	1.45	0.57	0.89	1.56	3.02	0.99	0.63	0.36	0.54	1.53
C6	1.70	3.33	5.03	1.44	0.57	0.88	1.57	3.02	0.93	0.62	0.31	0.53	1.46
C7	1.65	3.30	4.95	1.48	0.59	0.89	1.53	3.01	0.94	0.64	0.30	0.54	1.48
C8	1.61	3.30	4.92	1.46	0.56	0.90	1.55	3.01	0.92	0.62	0.30	0.54	1.46
C9	1.53	3.35	4.88	1.47	0.57	0.90	1.57	3.04	0.88	0.56	0.33	0.54	1.42
C10	1.51	3.34	4.85	1.45	0.56	0.89	1.54	2.99	0.93	0.63	0.30	0.54	1.46
C12	1.51	3.30	4.81	1.45	0.56	0.90	1.56	3.01	0.87	0.55	0.32	0.53	1.40
C13_1	1.59	3.29	4.88	1.47	0.57	0.90	1.53	3.00	1.01	0.54	0.47	0.54	1.55
C13_2	1.59	3.35	4.94	1.47	0.57	0.90	1.58	3.06	1.00	0.54	0.46	0.54	1.53
C14	3.26	3.36	6.62	1.43	0.56	0.88	1.55	2.99	0.84	0.55	0.29	0.53	1.37
C15	1.51	3.30	4.81	1.45	0.56	0.89	1.56	3.01	0.86	0.56	0.30	0.53	1.39
C16	1.90	3.31	5.21	1.62	0.56	1.07	1.55	3.17	1.10	0.65	0.45	0.53	1.63
C17				2.11	1.22		1.57	3.68	0.94	0.57	0.37	0.54	1.47

Table D.69 Insert time of 5 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.59	3.29	4.88	1.77	0.60	1.17	1.58	3.35	0.76	0.44	0.32	0.53	1.28
C3	1.53	3.31	4.84	1.74	0.59	1.16	1.56	3.30	0.93	0.61	0.32	0.53	1.45
C4	1.59	3.29	4.87	1.46	0.57	0.89	1.55	3.01	0.86	0.55	0.31	0.53	1.39
C5	1.50	3.34	4.83	1.46	0.57	0.89	1.58	3.04	0.98	0.63	0.35	0.53	1.51
C6	1.74	3.31	5.05	1.46	0.57	0.89	1.56	3.02	0.92	0.62	0.30	0.53	1.45
C7	1.69	3.21	4.91	1.48	0.60	0.88	1.57	3.04	0.93	0.63	0.29	0.53	1.45
C8	1.64	3.29	4.92	1.47	0.56	0.90	1.56	3.03	0.92	0.62	0.30	0.53	1.45
C9	1.57	3.30	4.87	1.47	0.57	0.90	1.58	3.06	0.87	0.55	0.32	0.52	1.40
C10	1.55	3.34	4.89	1.47	0.57	0.90	1.57	3.04	0.92	0.62	0.30	0.53	1.45
C12	1.55	3.34	4.89	1.46	0.56	0.90	1.58	3.04	0.86	0.54	0.32	0.53	1.39
C13_1	1.56	3.27	4.82	1.47	0.57	0.90	1.58	3.05	1.00	0.54	0.46	0.52	1.52
C13_2	1.55	3.29	4.84	1.49	0.57	0.92	1.57	3.06	1.00	0.54	0.46	0.53	1.52
C14	5.38	3.31	8.69	1.44	0.56	0.88	1.58	3.02	0.84	0.55	0.29	0.52	1.36
C15	1.49	3.33	4.82	1.46	0.56	0.89	1.57	3.03	0.85	0.55	0.29	0.53	1.37
C16	2.11	3.34	5.45	1.63	0.56	1.08	1.57	3.21	1.09	0.65	0.45	0.53	1.62
C17				2.12	1.23	0.89	1.58	3.69	0.93	0.57	0.37	0.53	1.46

Table D.70 Insert time of 5 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.60	3.36	4.96	1.78	0.61	1.18	1.58	3.36	0.75	0.43	0.32	0.48	1.24
C3	1.54	3.31	4.85	1.72	0.56	1.16	1.57	3.29	0.92	0.60	0.31	0.49	1.41
C4	1.60	3.35	4.95	1.46	0.57	0.89	1.57	3.03	0.86	0.55	0.31	0.49	1.35
C5	1.49	3.35	4.84	1.47	0.58	0.89	1.58	3.05	0.98	0.63	0.35	0.49	1.47
C6	1.75	3.34	5.09	1.47	0.57	0.89	1.55	3.02	0.92	0.62	0.30	0.49	1.41
C7	1.71	3.32	5.04	1.50	0.60	0.90	1.53	3.03	0.92	0.63	0.29	0.49	1.41
C8	1.66	3.30	4.96	1.47	0.57	0.91	1.54	3.01	0.91	0.62	0.30	0.50	1.41
C9	1.59	3.28	4.87	1.48	0.58	0.91	1.55	3.03	0.86	0.55	0.32	0.49	1.35
C10	1.56	3.33	4.89	1.46	0.57	0.89	1.56	3.03	0.91	0.62	0.29	0.49	1.40
C12	1.57	3.26	4.83	1.47	0.57	0.90	1.55	3.02	0.86	0.54	0.32	0.49	1.35
C13_1	1.55	3.36	4.91	1.48	0.58	0.90	1.55	3.03	0.99	0.53	0.46	0.49	1.48
C13_2	1.56	3.29	4.84	1.50	0.58	0.92	1.58	3.08	0.99	0.53	0.45	0.48	1.47
C14	5.40	3.30	8.70	1.44	0.56	0.88	1.56	3.01	0.83	0.54	0.29	0.49	1.32
C15	1.50	3.31	4.81	1.47	0.57	0.90	1.58	3.04	0.84	0.55	0.29	0.49	1.32
C16	2.18	3.31	5.49	1.65	0.56	1.09	1.56	3.21	1.09	0.64	0.44	0.50	1.58
C17				2.13	1.23	0.89	1.58	3.70	0.93	0.56	0.36	0.49	1.41

IV) Detail-Table for 10 MB. data size in cold cache

Table D.71 Replace time of 10 MB. data size, 20 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.58	6.69	9.27	4.38	3.05	1.33	3.20	7.58	4.89	4.48	0.41	1.10	5.99
C3	3.37	6.63	10.00	5.98	4.55	1.43	3.19	9.17	5.04	4.62	0.42	1.10	6.14
C4	5.61	6.71	12.31	4.04	2.78	1.26	3.19	7.23	3.32	2.91	0.41	1.13	4.45
C5	2.35	6.85	9.20	4.22	2.93	1.28	3.11	7.33	3.84	3.38	0.46	0.93	4.77
C6	2.83	6.70	9.52	4.13	2.87	1.26	3.17	7.30	5.03	4.57	0.46	0.93	5.95
C7	6.38	6.65	13.03	5.08	3.79	1.29	3.21	8.29	4.64	4.23	0.41	1.13	5.77
C8	2.63	6.65	9.28	4.22	2.94	1.27	3.16	7.38	4.46	4.07	0.40	1.10	5.56
C9	2.70	6.69	9.38	3.98	2.72	1.26	3.13	7.11	3.23	2.82	0.41	0.94	4.17
C10	3.35	6.58	9.93	4.25	3.00	1.25	3.19	7.44	4.60	4.19	0.41	1.10	5.70
C12	2.59	6.76	9.35	3.99	2.72	1.27	3.19	7.18	3.27	2.86	0.40	1.11	4.37
C14	7.83	6.68	14.52	3.91	2.68	1.23	3.17	7.08	3.24	2.84	0.40	0.93	4.18
C15	5.21	6.81	12.02	4.13	2.86	1.26	3.23	7.36	3.39	2.98	0.41	1.13	4.53
C16	3.82	6.63	10.45	4.88	2.49	2.39	3.16	8.03	5.02	4.31	0.70	1.94	6.96
C17				5.61	4.49	1.12	3.17	8.78	3.72	3.26	0.46	0.92	4.65

Table D.72 Replace time of 10 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.60	6.84	9.44	4.62	3.19	1.43	3.19	7.81	4.14	3.75	0.39	1.07	5.21
C3	3.39	6.76	10.15	6.93	5.40	1.53	3.19	10.12	4.51	4.12	0.39	1.07	5.59
C4	5.68	6.82	12.49	4.14	2.87	1.27	3.17	7.31	3.14	2.75	0.39	1.10	4.24
C5	2.34	6.83	9.18	4.34	3.06	1.28	3.11	7.45	3.68	3.23	0.45	0.91	4.59
C6	2.83	6.78	9.61	4.21	2.92	1.29	3.13	7.34	4.95	4.50	0.45	0.89	5.84
C7	6.47	6.72	13.19	5.33	4.00	1.33	3.18	8.50	4.31	3.92	0.39	1.09	5.39
C8	2.64	6.81	9.45	4.38	3.08	1.31	3.19	7.58	4.05	3.65	0.40	1.06	5.11
C9	2.72	6.77	9.49	4.10	2.76	1.33	3.11	7.20	3.20	2.81	0.40	0.89	4.10
C10	3.39	6.76	10.14	4.31	3.05	1.26	3.18	7.49	4.36	3.96	0.40	1.07	5.44
C12	2.61	6.86	9.47	4.11	2.82	1.29	3.15	7.26	3.20	2.80	0.39	1.07	4.27
C14	10.37	6.72	17.09	4.05	2.77	1.28	3.11	7.17	3.19	2.78	0.40	0.89	4.08
C15	5.28	6.78	12.06	4.27	2.98	1.29	3.17	7.44	3.33	2.94	0.39	1.09	4.42
C16	3.84	6.73	10.57	4.93	2.63	2.30	3.26	8.20	4.66	3.99	0.67	1.89	6.55
C17				5.92	4.74	1.19	3.12	9.04	3.55	3.11	0.44	0.89	4.44

Table D.73 Replace time of 10 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.62	6.80	9.42	4.76	3.26	1.50	3.21	7.97	4.19	3.81	0.38	0.97	5.16
C3	3.44	6.80	10.24	8.55	6.96	1.59	3.21	11.77	4.37	3.98	0.39	0.98	5.34
C4	5.78	6.77	12.55	4.25	2.91	1.34	3.17	7.42	3.10	2.72	0.38	1.00	4.10
C5	2.36	6.80	9.16	4.40	3.07	1.33	3.11	7.52	3.58	3.16	0.42	0.83	4.41
C6	2.85	6.77	9.61	4.27	2.94	1.33	3.11	7.38	4.57	4.14	0.42	0.82	5.39
C7	6.56	6.80	13.36	5.34	3.97	1.37	3.17	8.51	4.17	3.78	0.38	1.00	5.17
C8	2.66	6.84	9.50	4.54	3.17	1.37	3.19	7.73	4.05	3.67	0.38	0.97	5.03
C9	2.75	6.82	9.56	4.18	2.81	1.37	3.13	7.31	3.04	2.66	0.38	0.82	3.87
C10	3.43	6.81	10.23	4.60	3.15	1.45	3.22	7.81	4.22	3.84	0.38	0.98	5.20
C12	2.64	6.87	9.50	4.25	2.88	1.36	3.16	7.41	3.15	2.78	0.38	0.97	4.13
C14	15.49	6.77	22.26	4.21	2.84	1.37	3.11	7.33	3.13	2.75	0.38	0.82	3.96
C15	5.40	6.81	12.21	4.46	3.06	1.40	3.16	7.63	3.21	2.83	0.38	1.00	4.21
C16	3.92	6.77	10.69	5.38	2.88	2.50	3.19	8.57	4.47	3.86	0.61	1.71	6.19
C17				6.07	4.85	1.22	3.14	9.20	3.47	3.09	0.38	0.82	4.29

Table D.74 Replace time of 10 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.64	6.89	9.53	4.88	3.34	1.54	3.23	8.11	4.02	3.67	0.35	0.81	4.83
C3	3.48	6.82	10.29	15.01	13.38	1.62	3.13	18.14	4.20	3.85	0.35	0.81	5.01
C4	5.81	6.80	12.62	4.34	2.97	1.37	3.19	7.53	3.08	2.73	0.35	0.84	3.92
C5	2.39	6.87	9.26	4.61	3.22	1.39	3.14	7.76	3.45	3.10	0.35	0.69	4.14
C6	2.86	6.84	9.70	4.44	3.07	1.37	3.14	7.58	4.23	3.87	0.35	0.70	4.92
C7	6.62	6.82	13.43	5.70	4.30	1.39	3.18	8.88	4.01	3.65	0.36	0.84	4.85
C8	2.68	6.85	9.53	4.91	3.53	1.38	3.25	8.16	3.81	3.46	0.35	0.82	4.63
C9	2.74	6.83	9.58	4.26	2.86	1.40	3.12	7.38	2.91	2.56	0.35	0.69	3.60
C10	3.46	6.86	10.31	4.99	3.49	1.50	3.27	8.26	3.96	3.61	0.35	0.81	4.77
C12	2.64	7.01	9.66	4.32	2.93	1.39	3.16	7.48	3.12	2.76	0.35	0.82	3.94
C14	25.29	6.88	32.17	4.30	2.90	1.40	3.15	7.45	3.16	2.81	0.34	0.69	3.85
C15	5.44	6.86	12.29	4.71	3.28	1.43	3.19	7.90	3.20	2.85	0.35	0.84	4.04
C16	3.93	6.83	10.76	6.06	3.22	2.85	3.22	9.28	4.21	3.67	0.54	1.44	5.65
C17				6.37	5.08	1.29	3.14	9.51	3.39	3.04	0.34	0.69	4.08

Table D.75 Delete time of 10 MB. data size, 20 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.57	6.66	9.23	4.47	3.03	1.43	3.14	7.61	4.90	4.45	0.45	1.10	6.01
C3	2.52	6.78	9.30	5.99	4.57	1.42	3.14	9.14	5.37	4.92	0.45	1.10	6.47
C4	2.48	6.71	9.19	4.02	2.76	1.26	3.14	7.17	3.49	3.10	0.39	1.15	4.64
C5	2.34	6.82	9.16	4.65	3.37	1.29	3.11	7.77	4.04	3.57	0.47	0.93	4.96
C6	2.81	6.65	9.47	4.34	3.10	1.24	3.12	7.46	5.18	4.73	0.45	0.93	6.12
C7	2.61	6.72	9.33	5.35	4.09	1.26	3.14	8.50	4.75	4.30	0.45	1.13	5.87
C8	2.61	6.70	9.31	4.25	3.00	1.25	3.14	7.39	4.61	4.18	0.43	1.10	5.71
C9	2.70	6.80	9.49	3.93	2.66	1.26	3.12	7.05	3.30	2.83	0.47	0.93	4.23
C10	2.53	6.73	9.26	4.24	3.00	1.25	3.17	7.42	4.67	4.22	0.45	1.09	5.76
C12	2.56	6.72	9.28	4.00	2.73	1.27	3.14	7.14	3.28	2.88	0.40	1.10	4.39
C14	7.82	6.73	14.55	3.92	2.67	1.26	3.10	7.03	3.30	2.83	0.47	0.92	4.22
C15	2.41	6.69	9.10	4.20	2.94	1.26	3.13	7.34	3.46	3.01	0.44	1.13	4.58
C16	3.81	6.66	10.46	4.70	2.49	2.20	3.12	7.81	5.01	4.32	0.69	1.94	6.95
C17				5.67	4.54	1.13	3.12	8.79	3.72	3.26	0.46	0.92	4.64

Table D.76 Delete time of 10 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.59	6.82	9.41	4.61	3.18	1.43	3.18	7.78	4.25	3.85	0.40	1.06	5.31
C3	2.55	6.84	9.39	6.92	5.39	1.53	3.16	10.08	4.53	4.12	0.40	1.06	5.59
C4	2.48	6.75	9.23	4.12	2.85	1.27	3.16	7.28	3.21	2.81	0.40	1.09	4.30
C5	2.34	6.83	9.17	4.30	3.04	1.26	3.11	7.41	3.77	3.32	0.45	0.90	4.67
C6	2.83	6.83	9.66	4.23	2.93	1.30	3.13	7.36	4.77	4.32	0.45	0.89	5.67
C7	2.63	6.84	9.47	5.51	4.16	1.36	3.14	8.65	4.29	3.89	0.40	1.09	5.39
C8	2.62	6.86	9.48	4.40	3.04	1.36	3.15	7.55	3.99	3.60	0.39	1.06	5.05
C9	2.71	6.88	9.59	4.04	2.76	1.28	3.10	7.14	3.25	2.79	0.46	0.89	4.14
C10	2.53	6.77	9.30	4.27	3.01	1.26	3.17	7.44	4.51	4.11	0.40	1.06	5.57
C12	2.58	6.88	9.46	4.02	2.74	1.28	3.15	7.17	3.21	2.83	0.38	1.07	4.28
C14	10.39	6.72	17.11	4.03	2.75	1.28	3.11	7.14	3.29	2.84	0.45	0.89	4.19
C15	2.42	6.79	9.21	4.37	3.11	1.26	3.13	7.50	3.35	2.95	0.40	1.10	4.45
C16	3.83	6.72	10.55	4.96	2.67	2.29	3.17	8.13	4.72	4.05	0.67	1.86	6.58
C17				5.90	4.71	1.18	3.11	9.00	3.60	3.15	0.45	0.89	4.49

Table D.77 Delete time of 10 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.60	6.80	9.40	4.70	3.25	1.45	3.16	7.87	4.22	3.84	0.38	0.97	5.19
C3	2.55	6.82	9.37	15.17	13.61	1.56	3.16	18.33	4.48	4.10	0.38	0.97	5.45
C4	2.49	6.76	9.26	4.24	2.92	1.32	3.15	7.40	3.11	2.73	0.37	1.00	4.11
C5	2.36	6.76	9.13	4.44	3.09	1.35	3.11	7.56	3.53	3.13	0.39	0.82	4.35
C6	2.84	6.79	9.64	4.32	2.99	1.33	3.11	7.44	4.60	4.21	0.40	0.82	5.42
C7	2.65	6.88	9.53	5.50	4.12	1.38	3.16	8.66	4.32	3.94	0.38	1.01	5.32
C8	2.63	6.88	9.51	4.44	3.08	1.36	3.15	7.59	4.10	3.71	0.38	0.97	5.07
C9	2.72	6.84	9.57	4.23	2.87	1.37	3.11	7.35	3.09	2.71	0.37	0.82	3.91
C10	2.55	6.84	9.39	4.57	3.09	1.48	3.15	7.72	4.35	3.98	0.37	0.98	5.33
C12	2.60	6.83	9.43	4.21	2.85	1.36	3.16	7.37	3.14	2.77	0.37	0.98	4.12
C14	15.59	6.80	22.39	4.23	2.85	1.38	3.12	7.34	3.09	2.71	0.37	0.84	3.93
C15	2.44	6.79	9.22	4.42	3.07	1.35	3.16	7.58	3.19	2.82	0.38	1.00	4.20
C16	3.88	6.78	10.67	5.39	2.88	2.51	3.16	8.55	4.53	3.91	0.62	1.72	6.25
C17				6.08	4.84	1.24	3.13	9.22	3.43	3.06	0.37	0.83	4.26

Table D.78 Delete time of 10 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.61	6.85	9.46	4.96	3.43	1.53	3.16	8.12	4.29	3.95	0.34	0.82	5.10
C3	2.56	6.83	9.39	18.22	16.62	1.60	3.14	21.36	4.34	3.99	0.35	0.81	5.15
C4	2.50	6.89	9.39	4.32	2.95	1.37	3.17	7.50	3.11	2.75	0.36	0.84	3.95
C5	2.38	6.94	9.32	4.68	3.31	1.37	3.15	7.83	3.41	3.07	0.34	0.69	4.10
C6	2.85	6.96	9.81	4.45	3.05	1.39	3.15	7.60	4.23	3.89	0.35	0.70	4.93
C7	2.66	6.92	9.57	5.79	4.40	1.38	3.23	9.02	3.76	3.42	0.34	0.84	4.60
C8	2.64	6.91	9.55	4.95	3.55	1.40	3.16	8.11	3.69	3.35	0.34	0.82	4.52
C9	2.73	6.93	9.66	4.30	2.90	1.39	3.15	7.44	3.07	2.72	0.35	0.69	3.76
C10	2.56	6.88	9.44	4.96	3.49	1.47	3.14	8.10	3.81	3.48	0.33	0.82	4.63
C12	2.61	6.97	9.58	4.27	2.90	1.37	3.17	7.45	3.04	2.70	0.35	0.81	3.86
C14	25.26	6.92	32.18	4.24	2.88	1.36	3.13	7.37	3.02	2.67	0.35	0.69	3.71
C15	2.45	6.86	9.31	4.70	3.31	1.39	3.23	7.92	3.18	2.84	0.34	0.84	4.02
C16	3.91	6.84	10.76	6.08	3.23	2.86	3.15	9.23	4.16	3.62	0.54	1.43	5.59
C17				6.53	5.26	1.27	3.13	9.66	3.36	3.01	0.34	0.69	4.05

Table D.79 Insert time of 10 MB. data size, 20 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.59	6.68	9.27	4.63	3.38	1.26	3.15	7.79	4.87	4.46	0.40	1.10	5.97
C3	2.55	6.70	9.25	4.43	3.15	1.27	3.18	7.60	5.07	4.66	0.41	1.10	6.17
C4	2.51	6.67	9.18	4.02	2.77	1.24	3.16	7.18	3.46	3.05	0.41	1.14	4.60
C5	2.35	6.74	9.10	4.21	2.97	1.24	3.11	7.32	4.07	3.62	0.46	0.92	4.99
C6	2.83	6.75	9.58	4.13	2.88	1.25	3.12	7.24	5.29	4.84	0.46	0.93	6.22
C7	2.65	6.69	9.34	4.84	3.57	1.26	3.17	8.00	4.57	4.16	0.41	1.12	5.69
C8	2.63	6.69	9.32	4.24	2.99	1.25	3.15	7.39	4.44	4.04	0.40	1.10	5.53
C9	2.71	6.72	9.43	4.00	2.74	1.26	3.12	7.11	3.28	2.87	0.41	0.93	4.22
C10	2.54	6.67	9.22	4.22	2.99	1.23	3.19	7.41	4.62	4.22	0.39	1.09	5.71
C12	2.59	6.75	9.34	4.03	2.76	1.27	3.14	7.17	3.28	2.88	0.40	1.11	4.39
C13_1	2.64	6.73	9.37	4.39	3.17	1.23	3.20	7.59	3.69	3.01	0.68	1.15	4.84
C13_2	2.64	6.70	9.34	4.38	3.16	1.22	3.18	7.56	3.73	3.05	0.68	1.15	4.88
C14	8.07	6.69	14.76	4.00	2.76	1.24	3.11	7.11	3.28	2.83	0.46	0.93	4.22
C15	2.45	6.74	9.19	4.20	2.93	1.27	3.16	7.36	3.44	3.01	0.42	1.13	4.57
C16	3.82	6.66	10.48	4.71	2.43	2.27	3.14	7.85	5.01	4.31	0.69	1.94	6.95
C17				5.69	4.56	1.13	3.13	8.82	3.76	3.30	0.46	0.93	4.68

Table D.80 Insert time of 10 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.61	6.78	9.39	4.62	3.29	1.32	3.19	7.80	4.24	3.84	0.40	1.06	5.30
C3	2.56	6.84	9.40	4.64	3.30	1.33	3.19	7.82	4.46	4.06	0.40	1.06	5.52
C4	2.54	6.79	9.33	4.12	2.87	1.25	3.17	7.30	4.37	2.80	1.57	1.09	5.47
C5	2.36	6.81	9.17	4.29	3.05	1.24	3.10	7.40	3.67	3.23	0.45	0.90	4.57
C6	2.85	6.74	9.59	4.21	2.95	1.26	3.12	7.33	4.85	4.40	0.45	0.89	5.74
C7	2.68	6.80	9.48	5.08	3.80	1.28	3.18	8.26	4.40	4.00	0.39	1.08	5.48
C8	2.64	6.83	9.48	4.35	3.06	1.29	3.18	7.53	4.00	3.61	0.39	1.07	5.06
C9	2.72	6.74	9.46	4.14	2.81	1.32	3.11	7.24	3.16	2.77	0.39	0.90	4.06
C10	2.56	6.84	9.40	4.56	3.30	1.26	3.18	7.75	4.47	4.08	0.38	1.06	5.53
C12	2.62	6.80	9.41	4.11	2.83	1.28	3.15	7.26	3.22	2.83	0.38	1.07	4.29
C13_1	2.66	6.76	9.41	4.46	3.19	1.27	2.94	7.40	3.66	3.01	0.65	1.11	4.77
C13_2	2.66	6.73	9.39	4.40	3.13	1.27	2.93	7.33	3.68	3.03	0.65	1.11	4.79
C14	10.64	6.69	17.33	4.07	2.81	1.26	3.10	7.17	3.15	2.76	0.39	0.90	4.05
C15	2.48	6.77	9.25	4.32	3.06	1.26	3.17	7.48	3.32	2.92	0.39	1.09	4.40
C16	3.86	6.80	10.66	4.92	2.62	2.30	3.18	8.10	4.87	4.21	0.66	1.87	6.74
C17				5.83	4.67	1.17	3.11	8.94	3.58	3.13	0.44	0.89	4.47

Table D.81 Insert time of 10 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.62	6.83	9.45	4.80	3.39	1.41	3.19	7.98	4.13	3.76	0.37	0.98	5.11
C3	2.60	6.82	9.42	4.85	3.43	1.42	3.20	8.05	4.26	3.89	0.37	0.98	5.24
C4	2.59	6.80	9.39	4.23	2.96	1.26	3.15	7.38	3.09	2.72	0.37	1.00	4.09
C5	2.38	6.82	9.20	4.43	3.10	1.34	3.11	7.55	3.53	3.13	0.40	0.82	4.36
C6	2.87	6.79	9.66	4.27	2.98	1.29	3.11	7.38	4.47	4.07	0.40	0.82	5.29
C7	2.72	6.80	9.53	5.60	4.27	1.32	3.17	8.76	4.25	3.87	0.37	1.00	5.25
C8	2.66	6.78	9.45	4.53	3.16	1.37	3.20	7.72	3.81	3.44	0.37	0.98	4.79
C9	2.74	6.83	9.57	4.20	2.88	1.32	3.13	7.33	3.10	2.73	0.37	0.82	3.93
C10	2.60	6.85	9.46	4.85	3.38	1.47	3.20	8.05	4.31	3.94	0.37	0.97	5.28
C12	2.63	6.80	9.43	4.30	2.95	1.35	3.17	7.47	3.09	2.72	0.37	0.98	4.07
C13_1	2.71	6.79	9.50	4.65	3.33	1.32	3.12	7.77	3.56	2.92	0.63	1.07	4.63
C13_2	2.70	6.78	9.48	4.66	3.34	1.32	3.12	7.78	3.57	2.93	0.63	1.06	4.63
C14	15.71	6.88	22.59	4.20	2.83	1.36	3.11	7.30	3.12	2.76	0.36	0.82	3.95
C15	2.52	6.79	9.31	4.38	3.06	1.33	3.16	7.55	3.19	2.82	0.37	1.00	4.19
C16	3.92	6.79	10.71	5.19	2.81	2.38	3.18	8.38	4.78	4.16	0.62	1.71	6.49
C17				6.11	4.89	1.23	3.12	9.24	3.45	3.07	0.37	0.83	4.27

Table D.82 Insert time of 10 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.64	6.85	9.49	4.86	3.40	1.46	3.23	8.09	4.06	3.72	0.34	0.82	4.88
C3	2.63	6.84	9.47	5.09	3.60	1.49	3.11	8.20	4.08	3.74	0.34	0.82	4.90
C4	2.65	6.82	9.47	4.31	2.98	1.33	3.17	7.48	3.08	2.74	0.34	0.84	3.93
C5	2.39	6.79	9.18	4.64	3.31	1.34	3.15	7.79	3.46	3.11	0.34	0.69	4.15
C6	2.87	6.91	9.78	4.43	3.07	1.37	3.14	7.58	4.27	3.92	0.35	0.69	4.96
C7	2.79	6.81	9.59	5.73	4.38	1.34	3.18	8.91	3.86	3.52	0.34	0.84	4.70
C8	2.68	6.92	9.60	4.86	3.51	1.35	3.24	8.11	3.68	3.34	0.34	0.82	4.50
C9	2.76	6.88	9.63	4.24	2.88	1.36	3.11	7.36	3.07	2.73	0.34	0.69	3.76
C10	2.62	6.84	9.46	4.97	3.53	1.44	3.26	8.23	3.99	3.65	0.34	0.82	4.81
C12	2.65	6.86	9.51	4.29	2.93	1.36	3.15	7.44	2.99	2.65	0.34	0.82	3.81
C13_1	2.74	6.84	9.58	4.71	3.35	1.35	2.98	7.69	3.24	2.76	0.48	1.08	4.33
C13_2	2.74	6.84	9.58	4.72	3.35	1.36	2.98	7.70	3.24	2.76	0.49	1.08	4.33
C14	25.46	6.92	32.38	4.24	2.86	1.38	3.18	7.42	3.05	2.71	0.34	0.69	3.75
C15	2.59	6.83	9.42	4.68	3.31	1.37	3.19	7.87	3.08	2.74	0.34	0.84	3.92
C16	3.93	6.87	10.79	5.87	3.10	2.77	3.23	9.11	4.11	3.57	0.54	1.43	5.54
C17				6.42	5.14	1.28	3.14	9.56	3.32	2.97	0.34	0.69	4.01

V) Detail-Table for data size 10 MB. in warm cache

Table D.83 Replace time of 10 MB. data size, 20 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.96	6.60	8.56	2.05	0.62	1.42	3.20	5.25	1.69	1.28	0.41	1.10	2.79
C3	2.69	6.58	9.26	3.46	2.04	1.42	3.19	6.66	2.08	1.68	0.40	1.10	3.18
C4	4.94	6.65	11.59	1.81	0.57	1.24	3.19	5.00	0.96	0.56	0.40	1.14	2.10
C5	1.72	6.73	8.45	1.94	0.66	1.28	3.13	5.07	1.49	1.03	0.46	0.93	2.42
C6	2.21	6.62	8.83	1.82	0.57	1.25	3.13	4.96	1.30	0.84	0.46	0.94	2.23
C7	5.71	6.60	12.31	1.94	0.65	1.29	3.23	5.17	1.12	0.71	0.40	1.14	2.25
C8	2.01	6.58	8.59	1.91	0.65	1.26	3.15	5.07	1.05	0.63	0.41	1.10	2.15
C9	2.08	6.64	8.72	1.86	0.58	1.28	3.14	4.99	1.02	0.60	0.41	0.93	1.95
C10	2.71	6.56	9.27	1.88	0.62	1.26	3.19	5.08	1.16	0.75	0.41	1.12	2.28
C12	1.97	6.66	8.63	1.84	0.56	1.28	3.16	5.00	1.12	0.70	0.41	1.11	2.22
C14	7.14	6.64	13.78	1.81	0.57	1.23	3.18	4.99	0.98	0.58	0.40	0.93	1.91
C15	4.54	6.73	11.27	1.83	0.57	1.26	3.23	5.06	0.97	0.55	0.42	1.13	2.10
C16	3.14	6.60	9.74	2.72	0.58	2.14	3.16	5.87	1.55	0.84	0.71	1.95	3.50
C17				3.06	1.92	1.14	3.12	6.19	1.41	0.96	0.45	0.93	2.34

Table D.84 Replace time of 10 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.97	6.72	8.69	2.07	0.63	1.43	3.19	5.26	1.48	1.09	0.39	1.07	2.54
C3	2.73	6.71	9.43	5.00	3.48	1.53	3.19	8.20	1.76	1.37	0.39	1.07	2.83
C4	5.01	6.81	11.82	1.83	0.57	1.26	3.18	5.01	0.95	0.56	0.39	1.10	2.05
C5	1.73	6.76	8.49	1.97	0.67	1.30	3.11	5.08	1.42	0.98	0.44	0.90	2.32
C6	2.21	6.68	8.89	1.87	0.57	1.30	3.13	5.00	1.24	0.79	0.45	0.89	2.13
C7	5.80	6.68	12.47	1.99	0.66	1.33	3.18	5.17	1.09	0.69	0.40	1.09	2.18
C8	2.01	6.69	8.70	1.95	0.65	1.30	3.19	5.14	1.04	0.63	0.41	1.06	2.11
C9	2.11	6.70	8.81	1.89	0.58	1.31	3.11	5.00	0.97	0.56	0.41	0.89	1.87
C10	2.72	6.70	9.43	1.94	0.65	1.29	3.19	8.03	1.12	0.72	0.40	1.07	2.19
C12	1.98	6.83	8.81	1.87	0.57	1.30	3.17	5.04	1.04	0.64	0.40	1.07	2.11
C14	9.63	6.69	16.32	1.86	0.57	1.29	3.11	4.98	0.95	0.56	0.39	0.91	1.86
C15	4.61	6.79	11.39	1.87	0.58	1.29	3.18	5.06	0.96	0.56	0.40	1.10	2.06
C16	3.18	6.69	9.87	2.90	0.60	2.31	3.28	6.18	1.49	0.80	0.68	1.89	3.37
C17				3.13	1.96	1.17	3.14	6.27	1.27	0.84	0.43	0.89	2.16

Table D.85 Replace time of 10 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.00	6.69	8.70	2.14	0.65	1.49	3.22	5.35	1.34	0.95	0.38	0.98	2.32
C3	2.77	6.75	9.51	8.36	6.79	1.56	3.22	11.58	1.66	1.28	0.38	0.98	2.64
C4	5.13	6.72	11.85	1.92	0.58	1.33	3.20	5.11	0.94	0.56	0.38	1.00	1.94
C5	1.75	6.72	8.48	2.04	0.69	1.35	3.11	5.15	1.33	0.95	0.38	0.83	2.16
C6	2.23	6.68	8.91	1.92	0.60	1.32	3.12	5.04	1.11	0.73	0.38	0.83	1.94
C7	5.92	6.70	12.62	2.05	0.68	1.37	3.20	5.25	1.06	0.68	0.38	1.00	2.06
C8	2.04	6.75	8.79	2.05	0.67	1.39	3.22	5.27	1.00	0.62	0.38	0.99	1.99
C9	2.12	6.66	8.78	1.97	0.60	1.37	3.13	5.10	0.94	0.57	0.37	0.83	1.77
C10	2.77	6.70	9.47	2.17	0.69	1.48	3.22	11.51	1.07	0.69	0.38	0.98	2.05
C12	2.01	6.76	8.77	1.95	0.58	1.37	3.20	5.15	1.01	0.63	0.38	0.98	1.99
C14	14.64	6.70	21.34	1.95	0.58	1.36	3.13	5.08	0.94	0.56	0.38	0.83	1.77
C15	4.75	6.80	11.54	2.00	0.59	1.40	3.19	5.19	0.93	0.55	0.38	1.01	1.94
C16	3.25	6.67	9.92	3.11	0.63	2.48	3.18	6.30	1.44	0.80	0.64	1.72	3.16
C17				3.24	2.02	1.22	3.15	6.39	1.16	0.80	0.36	0.82	1.98

Table D.86 Replace time of 10 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.02	6.71	8.73	2.23	0.69	1.53	3.24	5.46	1.33	0.99	0.34	0.82	2.15
C3	2.81	6.72	9.53	15.79	14.19	1.60	3.12	18.91	1.86	1.52	0.34	0.82	2.68
C4	5.21	6.75	11.96	1.97	0.62	1.35	3.21	5.18	0.89	0.55	0.34	0.84	1.74
C5	1.77	6.80	8.57	2.10	0.73	1.37	3.15	5.26	1.35	1.00	0.35	0.69	2.05
C6	2.24	6.69	8.93	1.99	0.63	1.36	3.15	5.14	1.04	0.68	0.35	0.69	1.73
C7	5.99	6.80	12.79	2.10	0.71	1.39	3.19	5.30	1.00	0.66	0.35	0.83	1.84
C8	2.06	6.73	8.79	2.10	0.70	1.41	3.25	5.36	0.97	0.61	0.36	0.82	1.79
C9	2.12	6.73	8.85	2.01	0.62	1.39	3.12	5.13	0.90	0.56	0.34	0.70	1.60
C10	2.80	6.76	9.56	2.24	0.75	1.49	3.27	18.61	1.03	0.67	0.36	0.83	1.87
C12	2.03	6.93	8.96	2.00	0.60	1.39	3.16	5.16	1.00	0.66	0.34	0.83	1.83
C14	24.32	6.80	31.12	2.00	0.61	1.40	3.15	5.15	0.92	0.57	0.35	0.69	1.61
C15	4.81	6.78	11.59	2.08	0.64	1.44	3.20	5.28	0.91	0.56	0.34	0.84	1.75
C16	3.28	6.72	10.01	3.64	0.66	2.98	3.22	6.87	1.33	0.79	0.54	1.44	2.77
C17				3.46	2.17	1.29	3.13	6.60	1.30	0.96	0.34	0.69	1.99

Table D.87 Delete time of 10 MB. data size, 20 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.95	6.58	8.54	2.03	0.62	1.41	3.15	5.18	1.81	1.37	0.44	1.10	2.91
C3	1.91	6.63	8.54	3.44	2.01	1.43	3.15	6.59	2.03	1.57	0.45	1.10	3.12
C4	1.86	6.62	8.48	1.82	0.57	1.25	3.15	4.97	1.00	0.60	0.40	1.13	2.13
C5	1.72	6.74	8.46	1.94	0.66	1.28	3.12	5.07	1.47	1.00	0.47	0.92	2.39
C6	2.19	6.59	8.78	1.82	0.57	1.25	3.12	4.95	1.24	0.84	0.40	0.93	2.18
C7	1.99	6.66	8.65	1.93	0.65	1.28	3.15	5.08	1.20	0.75	0.45	1.12	2.32
C8	1.99	6.62	8.61	1.88	0.65	1.23	3.14	5.02	1.13	0.69	0.44	1.10	2.23
C9	2.08	6.71	8.79	1.83	0.57	1.26	3.12	4.95	1.09	0.62	0.46	0.93	2.01
C10	1.90	6.64	8.55	1.87	0.65	1.23	3.17	6.42	1.20	0.75	0.45	1.10	2.30
C12	1.95	6.64	8.58	1.84	0.57	1.27	3.15	4.99	1.10	0.70	0.40	1.10	2.20
C14	7.17	6.65	13.81	1.82	0.57	1.25	3.13	4.95	1.04	0.58	0.46	0.92	1.96
C15	1.79	6.61	8.40	1.82	0.58	1.24	3.14	4.96	1.03	0.59	0.44	1.12	2.15
C16	3.15	6.63	9.77	2.75	0.56	2.20	3.14	5.89	1.54	0.84	0.70	1.94	3.48
C17				3.08	1.93	1.15	3.12	6.20	1.47	1.01	0.46	0.92	2.39

Table D.88 Delete time of 10 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.96	6.73	8.69	2.05	0.63	1.42	3.17	5.23	1.47	1.07	0.40	1.06	2.53
C3	1.93	6.69	8.62	4.94	3.46	1.47	3.17	8.10	1.85	1.45	0.40	1.06	2.91
C4	1.86	6.68	8.53	1.83	0.57	1.25	3.17	4.99	0.96	0.57	0.39	1.09	2.05
C5	1.72	6.77	8.49	1.96	0.67	1.29	3.11	5.07	1.38	0.93	0.45	0.89	2.27
C6	2.21	6.69	8.90	1.86	0.58	1.28	3.13	4.99	1.17	0.78	0.39	0.89	2.07
C7	2.01	6.76	8.77	1.98	0.66	1.32	3.14	5.12	1.10	0.71	0.39	1.09	2.19
C8	2.00	6.74	8.73	1.98	0.65	1.32	3.16	5.14	1.05	0.65	0.40	1.07	2.11
C9	2.10	6.74	8.83	1.86	0.58	1.28	3.10	4.97	1.03	0.58	0.44	0.89	1.92
C10	1.91	6.71	8.62	1.95	0.67	1.28	3.16	7.93	1.11	0.71	0.39	1.06	2.17
C12	1.96	6.78	8.74	1.85	0.57	1.28	3.15	5.01	1.05	0.66	0.39	1.06	2.11
C14	9.72	6.75	16.47	1.86	0.57	1.28	3.12	4.98	1.01	0.56	0.45	0.89	1.90
C15	1.80	6.69	8.49	1.84	0.58	1.26	3.14	4.99	0.97	0.57	0.39	1.09	2.06
C16	3.16	6.76	9.91	2.90	0.60	2.30	3.17	6.07	1.45	0.78	0.66	1.87	3.32
C17				3.14	1.96	1.18	3.11	6.26	1.30	0.86	0.44	0.89	2.19

Table D.89 Delete time of 10 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.98	6.71	8.69	2.08	0.65	1.42	3.18	5.26	1.39	1.01	0.38	0.98	2.37
C3	1.93	6.73	8.65	8.11	6.59	1.52	3.17	11.28	1.73	1.35	0.38	0.97	2.71
C4	1.87	6.69	8.56	1.87	0.59	1.28	3.18	5.05	0.94	0.57	0.37	1.00	1.94
C5	1.75	6.72	8.46	2.01	0.69	1.32	3.12	5.13	1.39	1.01	0.38	0.82	2.21
C6	2.22	6.71	8.93	1.94	0.60	1.33	3.13	5.06	1.10	0.73	0.37	0.82	1.92
C7	2.02	6.80	8.82	2.04	0.67	1.36	3.16	5.20	1.08	0.70	0.38	1.00	2.08
C8	2.01	6.79	8.81	2.04	0.67	1.37	3.15	5.19	1.01	0.64	0.38	0.98	2.00
C9	2.11	6.78	8.89	1.94	0.59	1.35	3.12	5.06	0.95	0.57	0.37	0.82	1.77
C10	1.92	6.74	8.66	2.18	0.72	1.47	3.16	11.24	1.08	0.70	0.38	0.97	2.05
C12	1.97	6.80	8.77	1.94	0.58	1.36	3.17	5.11	1.02	0.65	0.37	0.97	1.99
C14	14.95	6.76	21.71	1.92	0.58	1.34	3.12	5.04	0.94	0.56	0.38	0.82	1.77
C15	1.81	6.72	8.54	1.97	0.60	1.36	3.15	5.12	0.94	0.56	0.38	1.00	1.95
C16	3.23	6.76	9.99	3.19	0.64	2.55	3.16	6.35	1.39	0.76	0.62	1.72	3.11
C17				3.27	2.04	1.23	3.13	6.39	1.19	0.81	0.38	0.82	2.02

Table D.90 Delete time of 10 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.99	6.77	8.75	2.20	0.69	1.51	3.17	5.37	1.30	0.95	0.35	0.81	2.11
C3	1.94	6.76	8.69	14.81	13.22	1.59	3.13	17.94	1.71	1.36	0.35	0.81	2.52
C4	1.89	6.81	8.70	1.98	0.61	1.37	3.18	5.16	0.92	0.56	0.36	0.83	1.76
C5	1.75	6.90	8.66	2.08	0.73	1.35	3.16	5.24	1.36	1.01	0.35	0.69	2.05
C6	2.23	6.88	9.11	2.00	0.63	1.37	3.14	5.15	1.03	0.69	0.34	0.69	1.72
C7	2.04	6.82	8.86	2.08	0.70	1.38	3.19	5.27	1.03	0.68	0.35	0.83	1.86
C8	2.03	6.84	8.88	2.10	0.70	1.40	3.15	5.24	0.96	0.61	0.35	0.82	1.78
C9	2.13	6.91	9.04	2.01	0.62	1.39	3.16	5.17	0.94	0.60	0.35	0.69	1.63
C10	1.93	6.74	8.68	2.23	0.75	1.48	3.13	17.85	1.03	0.68	0.35	0.82	1.84
C12	1.99	6.95	8.93	1.98	0.60	1.38	3.17	5.15	0.99	0.63	0.35	0.81	1.80
C14	24.60	6.84	31.44	1.97	0.60	1.37	3.13	5.11	0.87	0.53	0.34	0.69	1.57
C15	1.83	6.83	8.66	2.03	0.64	1.39	3.19	5.22	0.88	0.53	0.34	0.83	1.71
C16	3.25	6.87	10.12	3.42	0.66	2.76	3.16	6.57	1.29	0.75	0.54	1.44	2.73
C17				3.47	2.19	1.28	3.14	6.61	1.14	0.79	0.35	0.69	1.83

Table D.91 Insert time of 10 MB. data size, 20 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.97	6.58	8.55	1.94	0.67	1.28	3.15	5.10	1.72	1.32	0.40	1.10	2.82
C3	1.91	6.63	8.54	2.21	0.93	1.28	3.14	5.35	1.84	1.44	0.40	1.10	2.94
C4	1.88	6.59	8.48	1.79	0.57	1.22	3.30	5.09	0.98	0.58	0.40	1.12	2.10
C5	1.72	6.66	8.38	1.93	0.70	1.23	3.12	5.05	1.48	1.02	0.46	0.92	2.40
C6	2.20	6.68	8.88	1.84	0.59	1.25	3.12	4.96	1.25	0.85	0.40	0.93	2.18
C7	2.02	6.61	8.63	2.08	0.83	1.25	3.17	5.25	1.17	0.77	0.40	1.13	2.29
C8	2.01	6.59	8.60	1.98	0.74	1.24	3.35	5.33	1.08	0.69	0.38	1.10	2.18
C9	2.09	6.63	8.72	1.93	0.68	1.24	3.15	5.07	1.05	0.63	0.42	0.93	1.98
C10	1.92	6.63	8.55	1.94	0.69	1.25	3.22	5.16	1.22	0.82	0.40	1.10	2.33
C12	1.96	6.69	8.66	1.84	0.58	1.26	3.30	5.15	1.09	0.70	0.40	1.10	2.20
C13_1	2.02	6.61	8.63	1.85	0.60	1.24	3.16	5.01	1.45	0.77	0.68	1.14	2.59
C13_2	2.01	6.59	8.60	1.86	0.61	1.25	3.16	5.02	1.46	0.78	0.68	1.14	2.60
C14	7.37	6.66	14.03	1.91	0.67	1.24	3.11	5.02	1.00	0.59	0.41	0.92	1.93
C15	1.82	6.65	8.47	1.87	0.61	1.26	3.16	5.04	0.98	0.58	0.40	1.12	2.10
C16	3.14	6.60	9.74	2.92	0.64	2.28	3.16	6.08	1.52	0.82	0.70	1.94	3.47
C17				3.46	2.33	1.13	3.11	6.57	1.48	1.02	0.46	0.92	2.40

Table D.92 Insert time of 10 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.97	6.70	8.67	2.01	0.68	1.33	3.19	5.20	1.39	1.01	0.38	1.07	2.46
C3	1.93	6.76	8.69	2.28	0.94	1.34	3.21	5.49	1.60	1.22	0.38	1.06	2.66
C4	1.91	6.67	8.58	1.90	0.58	1.33	3.18	5.08	0.95	0.57	0.38	1.09	2.04
C5	1.74	6.72	8.46	1.98	0.73	1.25	3.11	5.09	1.39	0.94	0.45	0.89	2.28
C6	2.21	6.71	8.93	1.87	0.61	1.26	3.12	4.99	1.15	0.79	0.37	0.89	2.05
C7	2.06	6.73	8.79	2.12	0.85	1.27	3.18	5.30	1.10	0.73	0.37	1.08	2.19
C8	2.02	6.71	8.73	2.05	0.74	1.30	3.18	5.23	1.04	0.68	0.36	1.06	2.10
C9	2.10	6.69	8.79	2.02	0.70	1.32	3.11	5.13	0.98	0.58	0.40	0.89	1.87
C10	1.93	6.77	8.71	1.98	0.71	1.27	3.21	5.18	1.16	0.79	0.36	1.06	2.22
C12	1.98	6.72	8.69	1.86	0.59	1.27	3.17	5.04	1.04	0.65	0.39	1.06	2.11
C13_1	2.02	6.69	8.71	1.88	0.62	1.26	3.00	4.88	1.31	0.65	0.66	1.11	2.42
C13_2	2.02	6.66	8.69	1.88	0.62	1.26	2.99	4.87	1.32	0.66	0.65	1.11	2.43
C14	9.95	6.63	16.57	1.94	0.69	1.25	3.10	5.04	0.95	0.56	0.39	0.89	1.85
C15	1.86	6.71	8.57	1.90	0.64	1.26	3.17	5.08	1.17	0.57	0.60	1.08	2.25
C16	3.19	6.68	9.87	2.96	0.66	2.30	3.18	6.15	1.45	0.78	0.67	1.87	3.32
C17				3.56	2.40	1.16	3.10	6.67	1.28	0.83	0.44	0.89	2.17

Table D.93 Insert time of 10 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.00	6.72	8.72	2.20	0.79	1.41	3.19	5.39	1.32	0.95	0.37	0.98	2.30
C3	1.97	6.77	8.74	2.40	0.97	1.42	3.20	5.60	1.51	1.14	0.37	0.98	2.49
C4	1.96	6.73	8.69	1.89	0.59	1.30	3.18	5.06	0.94	0.57	0.37	1.00	1.94
C5	1.76	6.75	8.51	2.09	0.74	1.34	3.12	5.20	1.31	0.89	0.42	0.82	2.13
C6	2.23	6.73	8.96	1.95	0.66	1.29	3.11	5.06	1.09	0.73	0.36	0.81	1.91
C7	2.10	6.71	8.81	2.18	0.86	1.32	3.17	5.36	1.09	0.72	0.37	1.00	2.09
C8	2.04	6.81	8.85	2.16	0.80	1.36	3.20	5.36	0.99	0.63	0.36	0.98	1.97
C9	2.11	6.75	8.87	2.05	0.72	1.33	3.12	5.17	0.92	0.56	0.36	0.82	1.75
C10	1.96	6.78	8.74	2.23	0.75	1.48	3.19	5.41	1.12	0.75	0.36	0.97	2.09
C12	2.00	6.71	8.71	1.95	0.61	1.34	3.17	5.12	1.01	0.64	0.37	0.97	1.98
C13_1	2.07	6.72	8.79	1.98	0.66	1.33	3.14	5.12	1.29	0.64	0.64	1.06	2.35
C13_2	2.08	6.69	8.77	1.95	0.65	1.29	3.13	5.07	1.29	0.65	0.64	1.07	2.36
C14	14.88	6.85	21.74	2.08	0.71	1.36	3.11	5.19	0.93	0.55	0.37	0.82	1.75
C15	1.89	6.70	8.59	2.02	0.68	1.34	3.18	5.20	0.94	0.55	0.39	0.99	1.93
C16	3.26	6.73	9.99	3.10	0.70	2.40	3.18	6.28	1.37	0.76	0.60	1.71	3.08
C17				3.75	2.52	1.23	3.12	6.86	1.20	0.82	0.38	0.83	2.03

Table D.94 Insert time of 10 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.02	6.74	8.77	2.28	0.83	1.45	3.26	5.53	1.29	0.95	0.35	0.82	2.11
C3	2.00	6.78	8.78	2.52	1.05	1.46	3.12	5.63	1.46	1.12	0.34	0.82	2.29
C4	2.01	6.74	8.75	1.93	0.61	1.32	3.18	5.12	0.88	0.54	0.34	0.84	1.72
C5	1.77	6.73	8.50	2.19	0.85	1.34	3.15	5.34	1.22	0.87	0.35	0.69	1.91
C6	2.25	6.81	9.06	2.07	0.71	1.36	3.15	5.22	1.02	0.68	0.34	0.69	1.72
C7	2.17	6.73	8.89	2.31	0.98	1.34	3.20	5.52	1.03	0.69	0.34	0.84	1.87
C8	2.06	6.81	8.87	2.22	0.84	1.37	3.24	5.46	0.97	0.63	0.34	0.82	1.78
C9	2.12	6.78	8.90	2.11	0.73	1.37	3.13	5.24	0.88	0.54	0.34	0.69	1.58
C10	1.99	6.76	8.75	2.28	0.84	1.44	3.27	5.55	1.08	0.73	0.35	0.81	1.89
C12	2.02	6.77	8.79	1.97	0.61	1.35	3.16	5.13	0.97	0.62	0.34	0.81	1.78
C13_1	2.10	6.74	8.85	2.01	0.67	1.34	2.99	5.00	1.12	0.64	0.49	1.08	2.20
C13_2	2.10	6.75	8.84	2.02	0.66	1.36	2.98	5.00	1.13	0.63	0.50	1.07	2.21
C14	24.75	6.83	31.58	2.11	0.72	1.39	3.19	5.30	0.87	0.52	0.34	0.69	1.56
C15	1.96	6.76	8.72	2.12	0.75	1.36	3.20	5.32	0.95	0.60	0.34	0.83	1.78
C16	3.27	6.80	10.07	3.44	0.73	2.71	3.24	6.68	1.30	0.75	0.54	1.43	2.72
C17				4.06	2.80	1.26	3.14	7.20	1.14	0.79	0.35	0.69	1.83

VI) Detail-Table for data size 10 MB. in hot cache

Table D.95 Replace time of 10 MB. data size, 20 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.97	6.64	8.60	1.96	0.64	1.32	3.15	5.11	1.05	0.64	0.41	1.10	2.15
C3	2.69	6.64	9.33	4.85	3.43	1.42	3.16	8.01	1.11	0.71	0.40	1.11	2.21
C4	4.92	6.68	11.60	1.82	0.57	1.25	3.30	5.12	0.97	0.56	0.41	1.13	2.09
C5	1.73	6.75	8.48	1.94	0.66	1.28	3.12	5.06	1.16	0.70	0.46	0.92	2.08
C6	2.20	6.67	8.87	1.85	0.59	1.26	3.12	4.98	1.05	0.63	0.41	0.93	1.98
C7	5.71	6.63	12.34	2.10	0.81	1.29	3.17	5.27	1.10	0.69	0.40	1.13	2.23
C8	2.00	6.58	8.58	2.00	0.72	1.28	3.22	5.21	1.04	0.64	0.40	1.10	2.13
C9	2.08	6.63	8.71	1.93	0.67	1.26	3.19	5.13	0.99	0.57	0.42	0.93	1.92
C10	2.68	6.59	9.27	2.02	0.76	1.25	3.20	5.22	1.12	0.71	0.41	1.09	2.21
C12	1.97	6.67	8.64	1.84	0.56	1.27	3.16	4.99	0.97	0.56	0.40	1.10	2.07
C14	7.18	6.63	13.81	1.89	0.67	1.22	3.14	5.03	0.96	0.56	0.40	0.92	1.88
C15	4.53	6.78	11.32	1.99	0.72	1.26	3.22	5.21	0.97	0.56	0.40	1.13	2.09
C16	3.15	6.61	9.77	2.81	0.66	2.15	3.24	6.05	1.50	0.79	0.71	1.95	3.45
C17				3.52	2.38	1.14	3.16	6.68	1.05	0.58	0.47	0.92	1.97

Table D.96 Replace time of 10 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.98	6.73	8.71	2.12	0.65	1.47	3.17	5.29	1.04	0.63	0.40	1.07	2.11
C3	2.73	6.74	9.47	6.08	4.53	1.55	3.19	9.27	1.11	0.70	0.40	1.06	2.17
C4	5.02	6.83	11.85	1.86	0.58	1.28	3.15	5.01	0.96	0.56	0.40	1.09	2.05
C5	1.73	6.79	8.52	2.03	0.73	1.30	3.12	5.15	1.14	0.69	0.45	0.90	2.04
C6	2.22	6.73	8.94	1.90	0.60	1.29	3.12	5.02	1.04	0.63	0.41	0.89	1.93
C7	5.80	6.70	12.51	2.16	0.83	1.33	3.19	5.35	1.08	0.68	0.40	1.09	2.17
C8	2.01	6.73	8.74	2.13	0.82	1.31	3.16	5.29	1.04	0.64	0.40	1.07	2.10
C9	2.11	6.76	8.87	1.99	0.68	1.31	3.12	5.11	0.97	0.57	0.40	0.90	1.86
C10	2.73	6.74	9.47	2.10	0.81	1.29	3.19	5.30	1.10	0.70	0.40	1.06	2.16
C12	1.99	6.84	8.83	1.86	0.56	1.30	3.16	5.02	0.96	0.56	0.40	1.07	2.02
C14	9.71	6.70	16.41	1.98	0.69	1.29	3.14	5.12	0.96	0.56	0.40	0.89	1.85
C15	4.61	6.82	11.42	1.93	0.64	1.29	3.19	5.12	0.96	0.55	0.40	1.09	2.05
C16	3.20	6.74	9.93	3.00	0.69	2.32	3.18	6.19	1.45	0.78	0.67	1.89	3.35
C17				3.64	2.46	1.18	3.13	6.77	1.01	0.57	0.44	0.90	1.91

Table D.97 Replace time of 10 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.01	6.73	8.74	2.16	0.67	1.49	3.25	5.41	1.01	0.63	0.38	0.98	1.99
C3	2.78	6.75	9.53	8.32	6.74	1.58	3.24	11.56	1.08	0.70	0.38	0.97	2.06
C4	5.10	6.77	11.87	1.90	0.58	1.32	3.17	5.07	0.94	0.56	0.38	1.00	1.94
C5	1.76	6.77	8.53	2.14	0.80	1.34	3.13	5.27	1.09	0.69	0.39	0.82	1.91
C6	2.23	6.74	8.97	2.02	0.68	1.34	3.14	5.16	1.02	0.63	0.39	0.82	1.84
C7	5.90	6.74	12.64	2.31	0.92	1.39	3.17	5.49	1.06	0.68	0.38	1.00	2.07
C8	2.04	6.79	8.82	2.24	0.87	1.37	3.21	5.45	1.01	0.63	0.38	0.99	1.99
C9	2.12	6.71	8.83	2.08	0.73	1.35	3.13	5.21	0.95	0.57	0.38	0.83	1.78
C10	2.78	6.79	9.56	2.29	0.86	1.43	3.23	5.51	1.08	0.69	0.38	0.98	2.05
C12	2.03	6.80	8.83	2.06	0.69	1.37	3.24	5.31	0.96	0.56	0.39	0.98	1.93
C14	14.74	6.76	21.50	2.07	0.71	1.36	3.17	5.24	0.94	0.56	0.38	0.83	1.77
C15	4.73	6.81	11.54	2.08	0.67	1.40	3.18	5.26	0.93	0.55	0.38	1.00	1.93
C16	3.26	6.73	9.99	3.19	0.71	2.48	3.19	6.39	1.39	0.77	0.62	1.71	3.11
C17				4.24	3.02	1.22	3.16	7.40	0.97	0.57	0.39	0.82	1.79

Table D.98 Replace time of 10 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.11	6.82	8.93	2.37	0.84	1.53	3.24	5.61	0.98	0.63	0.35	0.81	1.79
C3	2.79	6.83	9.62	12.07	10.46	1.61	3.13	15.20	1.03	0.68	0.36	0.81	1.85
C4	5.17	6.82	11.99	1.98	0.62	1.36	3.21	5.19	0.91	0.56	0.35	0.84	1.75
C5	1.83	6.81	8.64	2.19	0.82	1.37	3.14	5.33	1.02	0.67	0.35	0.69	1.71
C6	2.42	6.80	9.22	2.14	0.76	1.38	3.14	5.28	0.98	0.63	0.35	0.69	1.67
C7	5.93	6.80	12.73	2.41	1.00	1.41	3.21	5.61	1.01	0.66	0.35	0.83	1.85
C8	2.15	6.81	8.96	2.30	0.89	1.41	3.27	5.57	0.97	0.63	0.34	0.82	1.79
C9	2.22	6.81	9.03	2.13	0.73	1.40	3.16	5.29	0.92	0.58	0.34	0.69	1.62
C10	2.79	6.83	9.63	2.37	0.90	1.47	3.28	5.65	1.04	0.69	0.35	0.82	1.86
C12	2.09	6.85	8.94	2.01	0.61	1.40	3.19	5.20	0.92	0.56	0.36	0.82	1.74
C14	14.76	6.82	21.58	2.13	0.74	1.39	3.15	5.27	0.92	0.56	0.36	0.69	1.61
C15	4.79	6.82	11.61	2.16	0.73	1.42	3.21	5.37	0.90	0.55	0.35	0.84	1.75
C16	3.56	6.81	10.37	3.80	0.81	2.98	3.24	7.04	1.29	0.75	0.54	1.44	2.73
C17				4.63	3.33	1.30	3.14	7.77	0.92	0.57	0.35	0.69	1.62

Table D.99 Delete time of 10 MB. data size, 20 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.95	6.63	8.58	2.09	0.68	1.41	3.14	5.22	1.08	0.63	0.45	1.10	2.18
C3	1.90	6.64	8.54	4.89	3.46	1.43	3.14	8.04	1.15	0.70	0.45	1.10	2.25
C4	1.85	6.67	8.52	1.81	0.57	1.24	3.14	4.95	0.97	0.56	0.40	1.13	2.09
C5	1.71	6.72	8.43	2.02	0.74	1.28	3.11	5.13	1.16	0.70	0.46	0.92	2.08
C6	2.19	6.62	8.82	1.83	0.59	1.24	3.13	4.96	1.04	0.64	0.41	0.93	1.97
C7	1.99	6.68	8.67	2.12	0.85	1.27	3.14	5.26	1.14	0.69	0.44	1.12	2.26
C8	1.98	6.64	8.62	1.99	0.75	1.24	3.14	5.13	1.08	0.64	0.44	1.10	2.18
C9	2.07	6.75	8.82	1.93	0.67	1.26	3.15	5.08	1.03	0.57	0.46	0.93	1.97
C10	1.90	6.65	8.55	1.98	0.75	1.23	3.13	5.11	1.16	0.71	0.44	1.11	2.27
C12	1.95	6.69	8.64	1.84	0.58	1.27	3.14	4.99	0.95	0.56	0.40	1.10	2.06
C14	7.18	6.68	13.86	1.91	0.67	1.24	3.11	5.02	1.02	0.56	0.46	0.92	1.94
C15	1.79	6.64	8.43	1.89	0.62	1.27	3.13	5.02	1.00	0.55	0.45	1.12	2.13
C16	3.14	6.68	9.83	2.83	0.66	2.17	3.14	5.98	1.48	0.78	0.70	1.95	3.43
C17				3.50	2.37	1.13	3.11	6.61	1.04	0.57	0.46	0.92	1.96

Table D.100 Delete time of 10 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.96	6.79	8.74	2.10	0.68	1.42	3.18	5.28	1.03	0.63	0.40	1.07	2.10
C3	1.92	6.77	8.70	6.07	4.54	1.53	3.16	9.22	1.10	0.70	0.40	1.06	2.16
C4	1.88	6.74	8.62	1.83	0.57	1.26	3.16	4.99	0.95	0.56	0.39	1.09	2.04
C5	1.72	6.79	8.51	2.01	0.73	1.28	3.12	5.13	1.13	0.69	0.44	0.89	2.02
C6	2.20	6.76	8.96	1.90	0.61	1.29	3.13	5.03	1.03	0.63	0.40	0.89	1.92
C7	2.01	6.76	8.77	2.18	0.84	1.33	3.14	5.32	1.08	0.68	0.40	1.09	2.16
C8	1.99	6.81	8.80	2.14	0.81	1.33	3.16	5.29	1.02	0.63	0.39	1.06	2.09
C9	2.08	6.76	8.84	1.96	0.68	1.29	3.12	5.08	1.01	0.56	0.45	0.89	1.90
C10	1.91	6.71	8.62	2.09	0.81	1.28	3.16	5.25	1.10	0.70	0.40	1.06	2.17
C12	1.96	6.80	8.76	1.88	0.59	1.29	3.16	5.03	0.95	0.56	0.39	1.06	2.00
C14	9.78	6.74	16.53	1.97	0.70	1.27	3.12	5.09	1.01	0.56	0.45	0.89	1.90
C15	1.79	6.77	8.56	1.90	0.64	1.26	3.13	5.04	0.95	0.55	0.40	1.09	2.04
C16	3.16	6.83	9.99	2.98	0.68	2.30	3.18	6.15	1.45	0.77	0.67	1.87	3.31
C17				3.61	2.42	1.19	3.11	6.72	1.01	0.57	0.44	0.89	1.90

Table D.101 Delete time of 10 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.98	6.76	8.73	2.20	0.74	1.46	3.17	5.36	1.00	0.62	0.38	0.98	1.98
C3	1.93	6.75	8.68	8.36	6.80	1.57	3.15	11.52	1.06	0.69	0.37	0.98	2.04
C4	1.87	6.75	8.63	1.91	0.59	1.32	3.15	5.06	0.94	0.56	0.37	1.00	1.94
C5	1.74	6.78	8.52	2.11	0.78	1.33	3.11	5.22	1.06	0.67	0.38	0.82	1.87
C6	2.22	6.77	9.00	1.99	0.67	1.32	3.12	5.10	1.00	0.62	0.38	0.82	1.82
C7	2.03	6.85	8.88	2.25	0.89	1.36	3.15	5.40	1.05	0.67	0.37	1.00	2.05
C8	2.01	6.76	8.77	2.25	0.87	1.38	3.14	5.38	1.01	0.63	0.38	0.98	1.99
C9	2.10	6.83	8.93	2.07	0.72	1.35	3.12	5.19	0.95	0.56	0.38	0.83	1.78
C10	1.92	6.79	8.70	2.32	0.86	1.46	3.15	5.47	1.07	0.70	0.38	0.98	2.05
C12	1.97	6.83	8.81	1.99	0.62	1.37	3.16	5.15	0.93	0.56	0.37	0.97	1.90
C14	14.98	6.84	21.82	2.11	0.74	1.37	3.11	5.23	0.94	0.56	0.37	0.83	1.77
C15	1.80	6.78	8.58	2.05	0.69	1.36	3.15	5.20	0.93	0.56	0.38	1.00	1.94
C16	3.24	6.81	10.05	3.20	0.72	2.49	3.16	6.36	1.38	0.76	0.62	1.71	3.10
C17				3.78	2.55	1.23	3.12	6.91	0.95	0.57	0.38	0.82	1.77

Table D.102 Delete time of 10 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.04	6.83	8.87	2.47	0.93	1.53	3.16	5.63	0.95	0.61	0.34	0.81	1.76
C3	1.98	6.83	8.81	12.12	10.54	1.59	3.14	15.27	1.01	0.66	0.34	0.81	1.82
C4	1.92	6.81	8.73	1.97	0.61	1.36	3.19	5.17	0.91	0.56	0.35	0.84	1.75
C5	1.77	6.83	8.61	2.21	0.83	1.37	3.16	5.37	1.01	0.66	0.35	0.68	1.69
C6	2.51	6.82	9.33	2.08	0.70	1.38	3.14	5.23	0.96	0.61	0.35	0.69	1.65
C7	2.16	6.84	8.99	2.32	0.95	1.37	3.20	5.52	0.99	0.65	0.34	0.83	1.83
C8	2.12	6.83	8.94	2.30	0.90	1.39	3.15	5.45	0.95	0.61	0.33	0.81	1.76
C9	2.24	6.84	9.07	2.14	0.74	1.40	3.15	5.29	0.90	0.56	0.34	0.68	1.59
C10	1.96	6.82	8.79	2.37	0.90	1.47	3.14	5.51	1.02	0.68	0.34	0.81	1.83
C12	2.06	6.84	8.90	2.01	0.63	1.38	3.19	5.20	0.91	0.56	0.35	0.81	1.72
C14	15.03	6.82	21.85	2.13	0.74	1.39	3.15	5.27	0.90	0.56	0.34	0.69	1.59
C15	1.83	6.81	8.64	2.15	0.74	1.41	3.21	5.36	0.90	0.55	0.34	0.83	1.73
C16	3.48	6.81	10.29	3.67	0.81	2.86	3.17	6.84	1.26	0.73	0.53	1.43	2.69
C17				4.22	2.94	1.28	3.14	7.36	0.91	0.57	0.34	0.69	1.60

Table D.103 Insert time of 10 MB. data size, 20 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.97	6.61	8.58	1.91	0.62	1.29	3.15	5.06	1.04	0.63	0.40	1.10	2.13
C3	1.92	6.65	8.57	1.89	0.62	1.27	3.26	5.15	1.04	0.64	0.40	1.10	2.14
C4	1.89	6.61	8.50	1.80	0.56	1.24	3.18	4.98	0.96	0.56	0.40	1.12	2.08
C5	1.73	6.62	8.35	1.91	0.66	1.25	3.13	5.04	1.16	0.70	0.46	0.92	2.09
C6	2.19	6.69	8.89	1.82	0.57	1.25	3.13	4.95	1.04	0.63	0.41	0.92	1.96
C7	2.02	6.62	8.64	1.90	0.65	1.25	3.17	5.08	1.08	0.68	0.40	1.13	2.21
C8	2.00	6.63	8.63	1.89	0.65	1.25	3.15	5.05	1.04	0.64	0.40	1.10	2.14
C9	2.08	6.63	8.71	1.83	0.57	1.26	3.13	4.96	0.99	0.57	0.42	0.94	1.92
C10	1.91	6.65	8.57	1.86	0.62	1.24	3.27	5.13	1.11	0.70	0.40	1.11	2.22
C12	1.96	6.74	8.71	1.84	0.57	1.27	3.15	4.98	0.96	0.56	0.40	1.10	2.06
C13_1	2.01	6.65	8.66	1.80	0.56	1.24	3.21	5.01	1.24	0.56	0.67	1.13	2.36
C13_2	2.01	6.62	8.63	1.79	0.56	1.23	3.20	4.99	1.24	0.56	0.68	1.13	2.37
C14	7.37	6.71	14.08	1.81	0.56	1.25	3.12	4.93	0.97	0.56	0.41	0.92	1.90
C15	1.82	6.67	8.49	1.83	0.57	1.26	3.18	5.01	0.96	0.56	0.40	1.13	2.08
C16	3.12	6.64	9.76	2.85	0.55	2.30	3.16	6.00	1.48	0.78	0.70	1.95	3.43
C17				3.08	1.93	1.15	3.12	6.20	1.03	0.57	0.46	0.92	1.96

Table D.104 Insert time of 10 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.97	6.74	8.71	1.94	0.63	1.31	3.19	5.13	1.02	0.63	0.39	1.07	2.09
C3	1.93	6.80	8.73	1.96	0.63	1.34	3.21	5.18	1.03	0.64	0.39	1.06	2.09
C4	1.94	6.72	8.66	1.80	0.56	1.24	3.18	4.99	0.95	0.56	0.39	1.08	2.03
C5	1.74	6.75	8.49	1.92	0.67	1.25	3.11	5.03	1.14	0.69	0.44	0.89	2.03
C6	2.22	6.70	8.92	1.86	0.58	1.28	3.12	4.98	1.04	0.63	0.41	0.89	1.93
C7	2.06	6.78	8.84	1.95	0.65	1.29	3.18	5.13	1.07	0.68	0.39	1.08	2.15
C8	2.02	6.75	8.77	1.97	0.65	1.32	3.18	5.16	1.02	0.63	0.39	1.06	2.08
C9	2.09	6.80	8.89	1.84	0.58	1.26	3.13	4.97	0.96	0.56	0.40	0.89	1.85
C10	1.93	6.79	8.72	1.91	0.63	1.28	3.21	5.11	1.08	0.69	0.39	1.06	2.14
C12	1.98	6.78	8.76	1.82	0.56	1.26	3.17	5.00	0.95	0.56	0.38	1.06	2.01
C13_1	2.04	6.74	8.77	1.83	0.57	1.26	3.01	4.83	1.22	0.56	0.66	1.10	2.31
C13_2	2.04	6.75	8.79	1.83	0.57	1.26	3.01	4.84	1.22	0.56	0.66	1.09	2.31
C14	10.00	6.69	16.69	1.82	0.57	1.25	3.12	4.94	0.95	0.55	0.40	0.89	1.84
C15	1.85	6.74	8.59	1.86	0.58	1.28	3.18	5.04	0.95	0.55	0.39	1.08	2.03
C16	3.17	6.72	9.89	2.78	0.57	2.21	3.19	5.97	1.44	0.77	0.67	1.87	3.31
C17				3.13	1.96	1.17	3.12	6.26	1.02	0.58	0.44	0.89	1.91

Table D.105 Insert time of 10 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.98	6.76	8.74	2.07	0.65	1.42	3.20	5.27	1.00	0.63	0.37	0.98	1.98
C3	1.96	6.84	8.81	2.06	0.63	1.42	3.20	5.26	1.00	0.63	0.37	0.98	1.98
C4	1.96	6.75	8.71	1.87	0.58	1.29	3.19	5.06	0.93	0.56	0.37	1.00	1.93
C5	1.75	6.80	8.55	1.98	0.68	1.30	3.12	5.10	1.05	0.68	0.36	0.82	1.87
C6	2.24	6.78	9.02	1.89	0.60	1.29	3.12	5.02	0.99	0.62	0.36	0.81	1.80
C7	2.10	6.81	8.91	1.99	0.67	1.32	3.19	5.18	1.04	0.67	0.37	1.00	2.04
C8	2.04	6.86	8.90	2.03	0.66	1.37	3.21	5.25	0.99	0.63	0.37	0.98	1.97
C9	2.10	6.84	8.94	1.95	0.59	1.36	3.13	5.08	1.13	0.56	0.57	0.82	1.95
C10	1.96	6.89	8.85	1.99	0.65	1.34	3.20	5.18	1.06	0.68	0.37	0.98	2.04
C12	2.01	6.72	8.73	1.87	0.58	1.29	3.18	5.05	0.92	0.55	0.37	0.97	1.90
C13_1	2.09	6.77	8.86	1.87	0.58	1.30	3.16	5.04	1.20	0.56	0.64	1.06	2.27
C13_2	2.09	6.77	8.86	1.86	0.57	1.29	3.15	5.02	1.20	0.55	0.64	1.06	2.25
C14	14.91	6.92	21.84	1.94	0.58	1.36	3.12	5.06	0.91	0.55	0.36	0.82	1.73
C15	1.89	6.81	8.70	1.94	0.59	1.35	3.19	5.13	0.93	0.55	0.37	1.00	1.93
C16	3.25	6.81	10.06	3.02	0.61	2.41	3.18	6.20	1.39	0.77	0.62	1.72	3.11
C17				3.26	2.04	1.22	3.12	6.38	0.94	0.57	0.37	0.83	1.77

Table D.106 Insert time of 10 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	1.99	6.81	8.80	2.15	0.70	1.45	3.24	5.38	0.96	0.62	0.34	0.81	1.77
C3	1.98	6.82	8.80	2.14	0.66	1.48	3.12	5.26	0.96	0.62	0.34	0.82	1.78
C4	2.11	6.81	8.91	1.93	0.61	1.32	3.19	5.12	0.90	0.56	0.34	0.83	1.73
C5	1.78	6.83	8.61	2.03	0.73	1.31	3.16	5.19	0.99	0.65	0.34	0.69	1.69
C6	2.35	6.82	9.17	1.97	0.63	1.34	3.15	5.12	0.95	0.61	0.34	0.69	1.64
C7	1.86	6.83	8.69	2.04	0.71	1.33	3.19	5.23	0.99	0.65	0.34	0.84	1.83
C8	2.32	6.86	9.18	2.07	0.70	1.38	3.25	5.33	0.95	0.61	0.34	0.81	1.76
C9	2.09	6.84	8.93	2.00	0.62	1.38	3.13	5.13	0.90	0.56	0.34	0.69	1.59
C10	2.19	6.83	9.02	2.09	0.68	1.41	3.27	5.36	1.01	0.67	0.34	0.81	1.83
C12	2.16	6.83	8.99	1.94	0.61	1.33	3.16	5.11	0.90	0.55	0.34	0.81	1.71
C13_1	2.16	6.86	9.02	1.95	0.60	1.34	3.00	4.95	1.04	0.55	0.49	1.07	2.11
C13_2	2.18	6.82	9.00	1.94	0.60	1.34	3.00	4.94	1.04	0.55	0.49	1.07	2.12
C14	15.06	6.84	21.91	1.96	0.60	1.36	3.18	5.14	0.89	0.55	0.34	0.69	1.58
C15	1.98	6.86	8.84	2.00	0.64	1.36	3.18	5.18	0.89	0.55	0.34	0.83	1.73
C16	3.33	6.83	10.16	3.32	0.64	2.68	3.23	6.55	1.27	0.74	0.53	1.43	2.70
C17				3.45	2.20	1.25	3.13	6.58	0.91	0.57	0.34	0.69	1.60

VII) Detail-Table for 20 MB. data size in cold cache

Table D.107 Replace time of 20 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.48	13.22	16.70	5.43	3.51	1.92	6.30	11.72	6.70	6.16	0.54	2.10	8.80
C3	4.96	13.35	18.30	9.73	7.29	2.43	6.27	16.00	7.57	7.03	0.54	2.09	9.66
C4	9.44	13.25	22.68	5.20	2.97	2.23	6.17	11.37	3.55	3.01	0.54	2.15	5.70
C5	3.02	13.22	16.24	5.56	3.34	2.22	6.09	11.65	4.31	3.76	0.55	1.76	6.07
C6	3.94	13.24	17.18	5.13	3.03	2.11	6.17	11.31	8.40	7.86	0.54	1.79	10.19
C7	10.98	13.28	24.26	7.49	5.36	2.12	6.23	13.72	7.10	6.56	0.54	2.15	9.26
C8	3.56	13.28	16.84	5.27	3.12	2.16	6.31	11.58	6.56	6.02	0.53	2.09	8.65
C9	3.71	13.06	16.76	5.02	2.86	2.16	6.18	11.20	3.49	2.94	0.55	1.74	5.23
C10	4.93	13.36	18.29	5.46	3.11	2.34	6.20	11.66	7.28	6.73	0.54	2.09	9.37
C12	3.51	13.29	16.80	5.04	2.91	2.13	6.24	11.28	3.59	2.97	0.62	2.09	5.68
C14	25.81	13.23	39.04	4.97	2.85	2.13	6.12	11.09	3.55	2.98	0.56	1.74	5.29
C15	8.60	13.36	21.96	5.44	3.24	2.20	6.19	11.63	3.70	3.14	0.55	2.16	5.85
C16	5.84	13.35	19.19	6.57	2.28	4.29	6.21	12.79	8.29	7.16	1.13	3.77	12.06
C17				8.41	6.23	2.18	6.14	14.55	3.95	3.40	0.55	1.77	5.72

Table D.108 Replace time of 20 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.49	13.27	16.75	5.78	3.53	2.25	6.21	11.99	5.58	4.96	0.62	2.02	7.60
C3	4.93	13.24	18.17	14.45	11.81	2.64	6.26	20.71	6.36	5.75	0.61	2.01	8.37
C4	9.48	13.18	22.66	5.44	3.08	2.37	6.28	11.72	3.52	2.89	0.62	2.07	5.59
C5	3.03	13.18	16.21	5.55	3.19	2.37	6.12	11.67	4.42	3.86	0.56	1.72	6.13
C6	3.96	13.20	17.16	5.69	3.39	2.30	6.17	11.85	7.53	6.97	0.56	1.68	9.22
C7	11.04	13.33	24.37	8.03	5.83	2.20	6.31	14.33	6.28	5.39	0.89	2.06	8.34
C8	3.57	13.24	16.81	5.93	3.46	2.47	6.30	12.23	5.46	4.85	0.61	2.01	7.47
C9	3.71	13.17	16.88	5.48	3.06	2.42	6.15	11.63	3.43	2.86	0.56	1.68	5.10
C10	4.93	13.19	18.12	5.95	3.42	2.52	6.27	12.22	6.17	5.57	0.60	2.01	8.19
C12	3.53	13.39	16.91	5.52	3.29	2.23	6.24	11.76	3.81	2.90	0.90	2.02	5.83
C14	42.78	13.17	55.95	5.20	3.02	2.18	6.10	11.30	3.48	2.94	0.54	1.68	5.16
C15	8.67	13.37	22.03	5.54	3.28	2.26	6.28	11.82	4.01	3.11	0.90	2.07	6.08
C16	5.89	13.24	19.13	6.96	2.48	4.48	6.23	13.18	7.28	6.23	1.05	3.63	10.91
C17				8.66	6.51	2.15	6.13	14.79	3.86	3.22	0.64	1.70	5.56

Table D.109 Replace time of 20 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.52	13.12	16.64	6.05	3.69	2.37	6.25	12.30	5.38	4.88	0.50	1.87	7.25
C3	4.99	13.23	18.22	22.85	20.17	2.68	6.33	29.17	6.32	5.82	0.50	1.87	8.19
C4	9.64	13.20	22.84	5.58	3.11	2.46	6.29	11.87	3.30	2.80	0.51	1.92	5.22
C5	3.05	13.13	16.18	6.04	3.52	2.52	6.11	12.15	4.24	3.67	0.57	1.55	5.79
C6	3.96	13.30	17.27	5.71	3.26	2.45	6.12	11.82	7.28	6.70	0.58	1.55	8.84
C7	11.12	13.17	24.29	7.87	5.60	2.27	6.32	14.19	5.95	5.45	0.50	1.91	7.86
C8	3.61	13.15	16.76	6.12	3.60	2.53	6.27	12.39	5.21	4.72	0.49	1.87	7.08
C9	3.73	13.17	16.89	5.67	3.13	2.54	6.12	11.79	3.44	2.85	0.58	1.56	5.00
C10	4.99	13.26	18.25	6.21	3.56	2.65	6.33	12.54	6.06	5.55	0.50	1.90	7.96
C12	3.54	13.42	16.96	5.33	3.07	2.27	6.21	11.54	3.37	2.86	0.51	1.88	5.24
C14	76.87	13.24	90.11	5.38	3.10	2.28	6.09	11.47	3.39	2.82	0.58	1.55	4.95
C15	8.86	13.18	22.04	5.83	3.39	2.44	6.21	12.03	3.39	2.88	0.50	1.92	5.30
C16	5.97	13.15	19.12	7.52	2.64	4.87	6.31	13.83	6.90	5.90	1.00	3.34	10.24
C17				9.60	7.15	2.45	6.13	15.72	3.70	3.12	0.58	1.56	5.26

Table D.110 Replace time of 20 MB. data size, 320 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.62	13.42	17.04	6.48	3.91	2.58	6.64	13.12	5.71	4.96	0.75	1.55	7.26
C3	5.24	13.40	18.64	41.55	38.55	3.00	6.60	48.15	6.21	5.47	0.74	1.55	7.76
C4	10.01	13.43	23.45	5.80	3.20	2.60	6.25	12.05	3.32	2.57	0.75	1.60	4.92
C5	3.14	13.39	16.53	6.44	3.84	2.60	6.15	12.58	3.96	3.50	0.46	1.31	5.27
C6	4.08	13.48	17.55	5.90	3.32	2.58	6.33	12.23	7.00	6.55	0.45	1.31	8.32
C7	11.56	13.60	25.16	8.80	6.41	2.39	6.52	15.32	5.47	4.71	0.76	1.60	7.07
C8	3.69	13.62	17.31	6.49	3.77	2.72	6.65	13.13	5.32	4.56	0.75	1.56	6.88
C9	3.81	13.40	17.22	5.95	3.22	2.73	6.43	12.38	3.10	2.64	0.46	1.35	4.45
C10	5.19	13.41	18.59	6.55	3.74	2.81	6.17	12.72	5.64	4.89	0.75	1.56	7.20
C12	3.67	13.82	17.49	5.75	3.25	2.50	6.52	12.28	3.31	2.84	0.47	1.57	4.88
C14	146.15	13.35	159.49	5.52	3.05	2.47	6.40	11.92	3.14	2.67	0.47	1.31	4.45
C15	9.22	13.56	22.78	6.38	3.80	2.58	6.25	12.63	3.63	2.86	0.76	1.60	5.22
C16	6.22	13.37	19.59	8.47	2.88	5.59	6.34	14.81	6.34	5.48	0.86	2.79	9.13
C17				10.34	7.79	2.56	6.23	16.57	3.54	3.08	0.45	1.30	4.84

Table D.111 Delete time of 20 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.44	13.22	16.66	5.48	3.53	1.96	6.21	11.69	7.04	6.40	0.64	2.09	9.13
C3	3.34	13.18	16.51	9.73	7.31	2.42	6.17	15.90	7.31	6.77	0.54	2.09	9.40
C4	3.25	13.23	16.48	5.08	2.96	2.12	6.18	11.26	4.00	3.06	0.93	2.15	6.14
C5	3.00	13.53	16.53	5.57	3.40	2.17	6.10	11.67	4.37	3.82	0.55	1.76	6.12
C6	3.90	13.29	17.18	5.21	3.08	2.12	6.12	11.33	8.50	7.95	0.55	1.76	10.26
C7	3.52	13.27	16.78	7.77	5.76	2.01	6.19	13.96	7.53	6.61	0.93	2.14	9.68
C8	3.50	13.24	16.74	5.29	3.14	2.15	6.19	11.49	6.69	6.06	0.63	2.09	8.78
C9	3.66	13.10	16.77	4.95	2.83	2.13	6.09	11.04	3.52	2.97	0.55	1.76	5.28
C10	3.34	13.19	16.52	5.30	3.07	2.24	6.20	11.51	7.74	7.10	0.64	2.08	9.83
C12	3.47	13.20	16.67	4.93	2.91	2.02	6.23	11.15	3.55	2.93	0.62	2.09	5.64
C14	25.98	13.21	39.19	5.00	2.86	2.14	6.06	11.06	3.54	2.98	0.55	1.75	5.28
C15	3.12	13.17	16.29	5.54	3.31	2.24	6.16	11.70	3.66	3.12	0.54	2.14	5.80
C16	5.81	13.31	19.12	6.55	2.26	4.29	6.21	12.76	8.26	7.13	1.13	3.76	12.02
C17				8.37	6.20	2.18	6.08	14.45	3.93	3.39	0.55	1.75	5.69

Table D.112 Delete time of 20 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.46	13.19	16.65	5.80	3.57	2.24	6.25	12.05	5.93	5.04	0.89	2.02	7.94
C3	3.34	13.14	16.48	14.30	11.83	2.47	6.55	20.85	6.81	5.92	0.89	2.02	8.83
C4	3.27	13.29	16.55	5.43	3.08	2.36	6.50	11.93	3.93	3.03	0.90	2.07	6.00
C5	3.01	13.37	16.38	5.89	3.54	2.35	6.12	12.02	4.30	3.75	0.55	1.69	6.00
C6	3.92	13.32	17.24	5.52	3.18	2.34	6.11	11.64	7.36	6.81	0.55	1.69	9.05
C7	3.57	13.31	16.87	8.11	5.89	2.23	6.43	14.54	6.73	5.84	0.89	2.06	8.79
C8	3.52	13.27	16.79	5.65	3.28	2.36	6.34	11.98	5.73	4.84	0.89	2.02	7.75
C9	3.68	13.15	16.83	5.30	2.99	2.30	6.13	11.42	3.47	2.92	0.55	1.69	5.16
C10	3.34	13.12	16.46	5.68	3.27	2.40	6.24	11.91	7.02	6.14	0.88	2.02	9.04
C12	3.47	13.39	16.86	5.18	3.06	2.12	6.23	11.41	3.78	2.88	0.90	2.01	5.79
C14	42.94	13.34	56.27	5.16	2.99	2.17	6.12	11.28	3.41	2.88	0.52	1.69	5.09
C15	3.12	13.27	16.39	5.58	3.33	2.24	6.23	11.81	3.60	3.06	0.54	2.07	5.66
C16	5.88	13.20	19.08	6.87	2.43	4.44	6.16	13.03	7.39	6.35	1.04	3.63	11.02
C17				8.92	6.64	2.28	6.26	15.18	3.95	3.31	0.64	1.69	5.63

Table D.113 Delete time of 20 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.50	13.11	16.60	6.09	3.72	2.37	6.20	12.29	5.34	4.84	0.50	1.88	7.22
C3	3.38	13.10	16.47	22.84	20.17	2.66	6.17	29.01	6.13	5.63	0.50	1.87	8.00
C4	3.28	13.16	16.43	5.56	3.15	2.41	6.28	11.84	3.41	2.89	0.52	1.92	5.33
C5	3.04	13.26	16.30	6.01	3.57	2.44	6.13	12.14	4.15	3.57	0.58	1.55	5.70
C6	3.94	13.32	17.25	5.68	3.25	2.43	6.15	11.83	7.19	6.61	0.57	1.55	8.74
C7	3.59	13.14	16.72	8.26	5.99	2.28	6.22	14.48	6.16	5.65	0.50	1.92	8.07
C8	3.54	13.12	16.65	6.09	3.56	2.53	6.18	12.27	5.19	4.70	0.50	1.87	7.07
C9	3.69	13.16	16.84	5.92	3.35	2.57	6.17	12.08	3.44	2.86	0.58	1.55	4.99
C10	3.35	13.06	16.41	6.14	3.51	2.62	6.22	12.36	5.84	5.34	0.50	1.87	7.71
C12	3.51	13.51	17.01	5.63	3.36	2.27	6.26	11.89	3.34	2.83	0.51	1.87	5.21
C14	76.83	13.18	90.01	5.56	3.31	2.26	6.13	11.69	3.43	2.85	0.58	1.56	4.99
C15	3.14	13.19	16.32	6.15	3.68	2.47	6.21	12.36	3.48	2.97	0.51	1.91	5.39
C16	5.92	13.18	19.10	7.29	2.63	4.66	6.23	13.52	7.08	6.09	0.98	3.35	10.42
C17				9.84	7.43	2.41	6.11	15.96	3.72	3.13	0.59	1.55	5.27

Table D.114 Delete time of 20 MB. data size, 320 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.56	13.44	16.99	6.54	4.01	2.53	6.45	12.99	5.17	4.69	0.49	1.55	6.73
C3	3.45	13.49	16.94	41.82	38.88	2.94	6.19	48.01	5.89	5.40	0.49	1.55	7.43
C4	3.34	13.68	17.02	6.28	3.71	2.57	6.25	12.53	3.31	2.85	0.46	1.60	4.91
C5	3.12	13.56	16.68	6.51	3.89	2.62	6.16	12.67	3.99	3.53	0.46	1.30	5.29
C6	4.06	13.61	17.67	5.90	3.31	2.58	6.15	12.04	6.85	6.39	0.46	1.30	8.15
C7	3.66	13.67	17.33	9.10	6.71	2.38	6.25	15.34	5.04	4.59	0.45	1.58	6.62
C8	3.63	13.55	17.18	6.44	3.76	2.68	6.22	12.66	5.01	4.56	0.44	1.56	6.56
C9	3.78	13.52	17.30	5.87	3.18	2.69	6.16	12.03	3.30	2.84	0.46	1.31	4.61
C10	3.43	13.42	16.85	6.52	3.74	2.78	6.21	12.74	5.36	4.84	0.51	1.55	6.91
C12	3.61	13.63	17.23	6.01	3.54	2.47	6.27	12.28	3.27	2.82	0.45	1.56	4.82
C14	146.71	13.56	160.27	5.68	3.20	2.48	6.15	11.83	3.23	2.77	0.46	1.31	4.54
C15	3.21	13.58	16.79	6.62	4.07	2.55	6.23	12.85	3.41	2.91	0.50	1.60	5.01
C16	6.19	13.57	19.76	8.39	2.87	5.52	6.17	14.55	6.22	5.38	0.84	2.78	9.00
C17				9.79	7.23	2.56	6.31	16.10	3.47	3.01	0.46	1.31	4.78

Table D.115 Insert time of 20 MB. data size, 40 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.45	13.15	16.60	5.46	3.54	1.92	6.21	11.67	6.77	6.22	0.54	2.10	8.86
C3	3.39	13.16	16.54	6.13	3.69	2.44	6.20	12.32	6.66	6.12	0.54	2.09	8.75
C4	3.33	13.09	16.42	5.12	2.98	2.14	6.18	11.30	3.57	3.02	0.54	2.15	5.72
C5	3.02	13.15	16.17	5.56	3.40	2.16	6.10	11.66	4.42	3.87	0.54	1.76	6.18
C6	3.93	13.22	17.15	5.20	3.07	2.13	6.12	11.32	8.32	7.77	0.55	1.76	10.07
C7	3.63	13.04	16.67	7.94	5.76	2.19	6.20	14.14	7.13	6.60	0.53	2.15	9.28
C8	3.56	13.09	16.65	5.29	3.17	2.12	6.21	11.50	6.55	6.03	0.52	2.09	8.64
C9	3.69	13.12	16.81	5.01	2.83	2.18	6.10	11.11	3.54	3.00	0.54	1.74	5.28
C10	3.37	13.26	16.63	5.47	3.22	2.25	6.21	11.68	7.27	6.73	0.53	2.09	9.36
C12	3.42	13.20	16.61	4.99	2.97	2.02	6.24	11.22	3.59	2.98	0.61	2.10	5.69
C13_1	3.57	13.21	16.78	5.68	3.31	2.37	6.74	12.42	4.01	3.16	0.85	2.12	6.14
C13_2	3.58	13.15	16.73	5.80	3.32	2.48	6.71	12.51	4.03	3.17	0.85	2.13	6.15
C14	26.22	13.04	39.27	5.01	2.82	2.19	6.08	11.09	3.51	2.96	0.54	1.75	5.25
C15	3.18	13.18	16.36	5.55	3.32	2.22	6.17	11.72	3.69	3.15	0.54	2.14	5.84
C16	5.90	13.14	19.04	6.43	2.15	4.29	6.22	12.65	8.39	7.27	1.12	3.76	12.15
C17				8.36	6.14	2.22	6.06	14.42	3.96	3.43	0.54	1.75	5.72

Table D.116 Insert time of 20 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.48	13.16	16.64	5.76	3.61	2.15	6.21	11.97	5.37	4.85	0.52	2.02	7.38
C3	3.40	13.22	16.62	6.24	3.74	2.50	6.55	12.79	5.74	5.22	0.52	2.01	7.76
C4	3.34	13.13	16.46	5.43	3.06	2.37	6.57	12.00	3.41	2.89	0.52	2.06	5.48
C5	3.03	13.23	16.26	5.82	3.43	2.39	6.12	11.93	4.35	3.82	0.53	1.69	6.04
C6	3.94	13.26	17.19	5.49	3.16	2.33	6.10	11.60	7.32	6.79	0.53	1.69	9.01
C7	3.66	13.11	16.76	8.01	5.81	2.20	6.48	14.49	6.52	5.99	0.52	2.07	8.58
C8	3.57	13.11	16.68	5.84	3.38	2.46	6.36	12.20	5.27	4.77	0.50	2.01	7.28
C9	3.69	13.33	17.01	5.31	2.95	2.36	6.13	11.44	3.52	2.96	0.56	1.69	5.21
C10	3.38	13.33	16.71	5.82	3.40	2.43	6.24	12.06	6.79	6.18	0.61	2.01	8.80
C12	3.49	13.23	16.72	5.34	3.10	2.24	6.23	11.57	3.47	2.91	0.56	2.01	5.49
C13_1	3.60	13.24	16.83	5.84	3.32	2.52	6.95	12.79	3.98	3.14	0.84	2.06	6.04
C13_2	3.61	13.25	16.85	5.80	3.32	2.48	6.89	12.69	4.00	3.15	0.85	2.05	6.06
C14	43.11	13.21	56.32	5.20	3.00	2.20	6.11	11.31	3.48	2.95	0.53	1.68	5.16
C15	3.20	13.27	16.47	5.62	3.38	2.24	6.27	11.89	3.59	3.07	0.52	2.07	5.66
C16	5.94	13.16	19.10	6.82	2.39	4.43	6.21	13.02	7.18	6.14	1.04	3.63	10.81
C17				9.13	6.85	2.28	6.25	15.38	3.75	3.23	0.52	1.69	5.43

Table D.117 Insert time of 20 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.50	13.17	16.67	6.09	3.72	2.38	6.25	12.34	5.25	4.75	0.50	1.87	7.13
C3	3.43	13.15	16.58	6.66	3.93	2.73	6.31	12.97	5.62	5.12	0.49	1.87	7.49
C4	3.43	13.12	16.55	5.61	3.14	2.47	6.28	11.89	3.32	2.82	0.50	1.91	5.24
C5	3.06	13.09	16.15	5.96	3.52	2.44	6.14	12.10	4.22	3.73	0.49	1.56	5.78
C6	3.97	13.21	17.18	5.60	3.20	2.40	6.13	11.73	7.06	6.56	0.50	1.56	8.62
C7	3.72	13.07	16.79	8.25	6.01	2.25	6.30	14.55	5.94	5.44	0.50	1.92	7.86
C8	3.59	13.06	16.64	6.05	3.50	2.55	6.27	12.32	5.16	4.66	0.49	1.87	7.02
C9	3.71	13.27	16.97	5.56	2.96	2.59	6.16	11.71	3.40	2.87	0.53	1.56	4.95
C10	3.42	13.17	16.59	6.07	3.49	2.58	6.36	12.43	6.18	5.68	0.50	1.86	8.05
C12	3.54	13.17	16.70	5.39	3.13	2.27	6.24	11.63	3.37	2.87	0.50	1.87	5.24
C13_1	3.66	13.09	16.75	6.39	3.75	2.64	6.71	13.10	3.88	3.04	0.84	1.98	5.86
C13_2	3.66	13.12	16.78	6.43	3.76	2.67	6.69	13.12	3.90	3.06	0.84	1.98	5.88
C14	77.03	13.24	90.27	5.42	3.16	2.26	6.13	11.55	3.33	2.81	0.52	1.55	4.88
C15	3.29	13.25	16.54	6.13	3.70	2.43	6.18	12.31	3.42	2.91	0.50	1.92	5.34
C16	5.96	13.31	19.26	7.21	2.56	4.65	6.32	13.53	7.01	6.01	0.99	3.34	10.34
C17				9.55	7.11	2.44	6.10	15.65	3.62	3.13	0.49	1.56	5.18

Table D.118 Insert time of 20 MB. data size, 320 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	3.59	13.56	17.15	6.60	4.08	2.52	6.57	13.17	5.14	4.68	0.46	1.55	6.69
C3	3.59	13.41	17.00	7.23	4.24	2.99	6.36	13.59	5.46	5.01	0.45	1.55	7.01
C4	3.59	13.46	17.05	5.79	3.19	2.60	6.26	12.04	3.16	2.68	0.48	1.59	4.75
C5	3.16	13.44	16.61	6.42	3.82	2.60	6.16	12.58	4.07	3.61	0.46	1.31	5.38
C6	4.11	13.42	17.53	5.91	3.29	2.62	6.15	12.06	6.93	6.47	0.46	1.30	8.24
C7	3.93	13.44	17.36	9.09	6.47	2.62	6.26	15.35	5.30	4.83	0.47	1.59	6.89
C8	3.69	13.41	17.10	6.90	4.24	2.66	6.39	13.29	5.01	4.56	0.45	1.55	6.56
C9	3.80	13.55	17.35	5.70	3.02	2.68	6.15	11.85	3.29	2.83	0.46	1.31	4.60
C10	3.55	13.46	17.00	7.04	4.25	2.79	6.17	13.21	5.41	4.95	0.45	1.55	6.96
C12	3.65	13.61	17.25	5.70	3.20	2.50	6.28	11.98	3.29	2.84	0.46	1.56	4.85
C13_1	3.84	13.40	17.24	6.61	3.82	2.79	6.67	13.28	3.77	2.98	0.79	1.71	5.47
C13_2	3.84	13.46	17.30	6.53	3.76	2.78	6.64	13.17	3.77	2.98	0.79	1.71	5.48
C14	146.62	13.53	160.15	5.72	3.22	2.50	6.12	11.84	3.30	2.85	0.45	1.31	4.61
C15	3.46	13.75	17.20	6.43	3.88	2.55	6.25	12.68	3.27	2.78	0.48	1.59	4.85
C16	6.26	13.43	19.68	8.34	2.80	5.55	6.33	14.67	6.44	5.59	0.84	2.78	9.22
C17				10.38	7.86	2.52	6.31	16.69	3.47	3.01	0.46	1.30	4.77

VIII) Detail-Table for data size 20 MB. in warm cache

Table D.119 Replace time of 20 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.78	13.22	16.00	2.66	0.74	1.92	6.23	8.89	2.02	1.48	0.54	2.10	4.12
C3	4.25	13.41	17.66	8.81	6.38	2.43	6.24	15.05	2.65	2.11	0.54	2.10	4.74
C4	8.78	13.36	22.14	2.75	0.58	2.17	6.21	8.96	1.11	0.57	0.54	2.16	3.27
C5	2.36	13.48	15.84	3.00	0.80	2.19	6.09	9.09	2.25	1.70	0.56	1.83	4.08
C6	3.26	13.39	16.64	2.74	0.62	2.12	6.11	8.85	1.53	0.98	0.55	1.82	3.35
C7	10.35	13.50	23.85	3.11	1.07	2.03	6.22	9.33	1.39	0.84	0.54	2.15	3.54
C8	2.88	13.31	16.19	3.06	0.95	2.12	6.25	9.32	1.27	0.73	0.54	2.10	3.37
C9	3.03	13.16	16.19	2.94	0.76	2.18	6.12	9.06	1.16	0.61	0.54	1.75	2.91
C10	4.25	13.36	17.61	3.21	0.90	2.31	6.24	9.46	1.41	0.87	0.54	2.10	3.51
C12	2.82	13.51	16.33	2.66	0.57	2.09	6.23	8.89	1.43	0.81	0.61	2.10	3.53
C14	25.50	13.17	38.68	2.78	0.76	2.02	6.13	8.91	1.16	0.60	0.55	1.75	2.91
C15	8.02	13.57	21.59	2.86	0.63	2.23	6.21	9.07	1.10	0.56	0.54	2.16	3.27
C16	5.25	13.35	18.61	5.12	0.82	4.29	6.24	11.35	2.12	0.99	1.13	3.92	6.04
C17				6.04	3.88	2.16	6.12	12.15	1.59	1.04	0.55	1.81	3.40

Table D.120 Replace time of 20 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.85	13.28	16.13	2.95	0.75	2.20	6.31	9.26	1.92	1.40	0.52	2.02	3.94
C3	4.31	13.25	17.56	12.72	10.16	2.56	6.31	19.03	2.43	1.91	0.52	2.01	4.45
C4	8.79	13.18	21.97	2.97	0.60	2.37	6.23	9.21	1.08	0.56	0.52	2.07	3.16
C5	2.40	13.23	15.64	3.20	0.83	2.37	6.21	9.41	2.13	1.57	0.55	1.92	4.05
C6	3.29	13.26	16.54	2.93	0.64	2.29	6.21	9.14	1.47	0.93	0.54	1.91	3.38
C7	10.36	13.34	23.70	3.33	1.15	2.17	6.31	9.64	1.35	0.82	0.53	2.07	3.42
C8	2.93	13.24	16.17	3.33	0.94	2.39	6.35	9.68	1.23	0.71	0.52	2.02	3.25
C9	3.05	13.18	16.23	3.24	0.84	2.40	6.22	9.46	1.12	0.58	0.53	1.69	2.81
C10	4.33	13.16	17.49	3.45	0.95	2.49	6.26	9.70	1.37	0.84	0.52	2.01	3.38
C12	2.88	13.38	16.25	2.77	0.59	2.17	6.31	9.07	1.28	0.73	0.55	2.03	3.31
C14	42.32	13.16	55.48	3.04	0.85	2.19	6.21	9.25	1.10	0.57	0.53	1.75	2.84
C15	8.02	13.29	21.31	2.95	0.65	2.29	6.30	9.25	1.09	0.56	0.53	2.09	3.19
C16	5.24	13.29	18.53	5.35	0.91	4.44	6.26	11.62	2.04	0.98	1.06	3.64	5.68
C17				6.49	4.19	2.30	6.16	12.65	1.40	0.88	0.53	1.68	3.09

Table D.121 Replace time of 20 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.87	13.09	15.96	3.23	0.86	2.37	6.26	9.48	1.73	1.22	0.50	1.89	3.61
C3	4.38	13.12	17.51	20.42	17.73	2.69	6.32	26.74	2.41	1.91	0.50	1.88	4.30
C4	9.04	13.22	22.26	3.10	0.63	2.47	6.30	9.40	1.07	0.57	0.50	1.95	3.03
C5	2.42	13.08	15.50	3.33	0.86	2.47	6.13	9.46	1.91	1.39	0.52	1.56	3.47
C6	3.33	13.21	16.53	3.16	0.72	2.44	6.52	9.68	1.38	0.86	0.52	1.55	2.94
C7	10.52	13.17	23.69	3.50	1.23	2.27	6.30	9.79	1.30	0.79	0.50	1.92	3.22
C8	2.96	13.11	16.07	3.56	1.03	2.53	6.27	9.83	1.20	0.70	0.50	1.87	3.07
C9	3.07	13.10	16.16	3.42	0.86	2.56	6.12	9.54	1.08	0.58	0.49	1.56	2.64
C10	4.37	13.16	17.53	3.67	1.03	2.64	6.34	10.01	1.31	0.81	0.50	1.87	3.19
C12	2.90	13.31	16.21	2.94	0.64	2.30	6.21	9.15	1.22	0.72	0.50	1.88	3.10
C14	76.54	13.12	89.66	3.17	0.86	2.31	6.50	9.67	1.09	0.57	0.52	1.55	2.64
C15	8.21	13.20	21.41	3.18	0.75	2.42	6.50	9.68	1.07	0.56	0.51	1.94	3.00
C16	5.36	13.18	18.55	5.83	1.12	4.70	6.60	12.42	1.95	0.96	0.99	3.35	5.30
C17				6.77	4.34	2.44	6.30	13.07	1.33	0.84	0.49	1.55	2.88

Table D.122 Replace time of 20 MB. data size, 320 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.98	13.43	16.41	3.47	0.92	2.55	6.49	9.96	1.84	1.38	0.46	1.55	3.39
C3	4.47	13.66	18.13	36.57	33.58	2.98	6.47	43.04	2.27	1.82	0.45	1.55	3.83
C4	9.34	13.58	22.92	3.29	0.69	2.60	6.28	9.57	1.01	0.56	0.45	1.60	2.61
C5	2.48	13.44	15.92	3.78	1.15	2.64	6.17	9.96	1.71	1.25	0.46	1.31	3.02
C6	3.43	13.55	16.98	3.36	0.76	2.60	6.20	9.56	1.25	0.78	0.47	1.31	2.56
C7	10.95	13.43	24.38	4.02	1.63	2.39	6.26	10.28	1.22	0.75	0.46	1.59	2.81
C8	3.02	13.47	16.50	3.84	1.13	2.71	6.40	10.24	1.13	0.67	0.46	1.55	2.68
C9	3.16	13.38	16.54	3.82	1.09	2.74	6.17	9.99	1.04	0.57	0.47	1.31	2.35
C10	4.47	13.46	17.93	3.96	1.13	2.83	6.17	10.13	1.24	0.79	0.45	1.56	2.79
C12	3.00	13.53	16.53	3.15	0.65	2.50	6.30	9.45	1.17	0.72	0.44	1.56	2.73
C14	145.95	13.38	159.33	3.61	1.13	2.48	6.45	10.06	1.02	0.56	0.46	1.30	2.32
C15	8.61	13.87	22.48	3.87	1.27	2.60	6.54	10.40	1.01	0.56	0.45	1.59	2.60
C16	5.58	13.39	18.97	6.75	1.24	5.52	6.63	13.39	1.75	0.89	0.86	2.79	4.53
C17				7.36	4.77	2.59	6.15	13.51	1.28	0.82	0.46	1.30	2.58

Table D.123 Delete time of 20 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.85	13.39	16.23	2.79	0.82	1.97	6.23	9.02	2.27	1.63	0.64	2.10	4.36
C3	2.71	13.18	15.89	8.70	6.29	2.41	6.20	14.90	2.64	2.10	0.54	2.10	4.74
C4	2.63	13.46	16.09	2.62	0.62	2.00	6.23	8.85	1.23	0.60	0.63	2.17	3.40
C5	2.38	13.43	15.81	2.94	0.81	2.13	6.14	9.09	2.22	1.67	0.55	1.79	4.01
C6	3.27	13.42	16.69	2.62	0.61	2.01	6.15	8.77	1.52	0.97	0.55	1.77	3.29
C7	2.93	13.31	16.24	3.01	1.04	1.97	6.20	9.21	1.53	0.91	0.63	2.15	3.69
C8	2.90	13.32	16.22	3.07	0.98	2.09	6.22	9.29	1.40	0.76	0.64	2.10	3.51
C9	3.03	13.39	16.42	2.88	0.80	2.08	6.13	9.00	1.19	0.64	0.55	1.77	2.96
C10	2.71	13.24	15.95	3.23	0.99	2.24	6.25	9.48	1.60	0.96	0.64	2.10	3.69
C12	2.85	13.31	16.16	2.55	0.59	1.96	6.25	8.80	1.42	0.80	0.62	2.10	3.52
C14	25.48	13.20	38.68	2.79	0.76	2.03	6.14	8.93	1.14	0.59	0.55	1.75	2.89
C15	2.49	13.44	15.93	2.83	0.66	2.17	6.23	9.05	1.23	0.60	0.63	2.15	3.38
C16	5.22	13.59	18.81	5.14	0.85	4.29	6.24	11.38	2.18	1.05	1.13	3.77	5.95
C17				5.88	3.84	2.04	6.14	12.02	1.60	1.05	0.55	1.76	3.37

Table D.124 Delete time of 20 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.85	13.30	16.15	3.05	0.86	2.18	6.33	9.37	2.01	1.42	0.59	2.02	4.02
C3	2.73	13.17	15.89	12.69	10.20	2.50	6.29	18.98	2.36	1.84	0.52	2.02	4.38
C4	2.64	13.35	15.98	2.97	0.62	2.35	6.21	9.18	1.17	0.60	0.57	2.07	3.24
C5	2.39	13.45	15.85	3.20	0.82	2.38	6.13	9.33	2.15	1.61	0.54	1.70	3.86
C6	3.29	13.34	16.63	2.95	0.64	2.31	6.15	9.10	1.46	0.92	0.54	1.69	3.15
C7	2.93	13.36	16.29	3.26	1.11	2.15	6.24	9.50	1.44	0.84	0.59	2.07	3.51
C8	2.91	13.39	16.30	3.54	1.08	2.46	6.24	9.78	1.34	0.75	0.59	2.02	3.37
C9	3.05	13.26	16.31	3.18	0.83	2.35	6.14	9.32	1.14	0.61	0.53	1.70	2.84
C10	2.86	13.19	16.05	3.59	1.15	2.43	6.26	9.84	1.47	0.88	0.59	2.02	3.49
C12	2.85	13.47	16.32	2.73	0.59	2.14	6.27	9.00	1.32	0.73	0.59	2.01	3.33
C14	42.43	13.36	55.80	3.03	0.81	2.21	6.13	9.16	1.10	0.57	0.53	1.70	2.79
C15	2.50	13.29	15.78	2.91	0.67	2.24	6.22	9.13	1.17	0.59	0.58	2.08	3.26
C16	5.26	13.31	18.56	5.31	0.89	4.42	6.18	11.49	2.05	1.01	1.04	3.63	5.69
C17				6.40	4.11	2.29	6.13	12.53	1.44	0.92	0.52	1.69	3.13

Table D.125 Delete time of 20 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.87	13.06	15.93	3.26	0.89	2.37	6.18	9.44	1.88	1.38	0.50	1.88	3.75
C3	2.73	13.03	15.76	19.91	17.44	2.48	6.16	26.07	2.16	1.67	0.50	1.87	4.04
C4	2.65	13.12	15.78	3.07	0.64	2.43	6.28	9.35	1.12	0.60	0.52	1.93	3.05
C5	2.41	13.23	15.63	3.30	0.87	2.43	6.12	9.42	1.91	1.39	0.53	1.56	3.48
C6	3.30	13.22	16.52	3.14	0.73	2.41	6.11	9.26	1.38	0.87	0.51	1.56	2.94
C7	2.97	13.05	16.02	3.40	1.22	2.17	6.22	9.61	1.32	0.82	0.49	1.92	3.24
C8	2.93	13.07	16.01	3.70	1.16	2.54	6.22	9.92	1.25	0.75	0.49	1.89	3.13
C9	3.06	13.07	16.13	3.43	0.90	2.53	6.15	9.58	1.13	0.62	0.51	1.56	2.70
C10	2.91	13.04	15.95	3.82	1.21	2.61	6.17	9.98	1.34	0.84	0.50	1.87	3.21
C12	2.86	13.45	16.31	2.91	0.65	2.26	6.21	9.12	1.22	0.71	0.51	1.87	3.09
C14	76.62	13.08	89.70	3.09	0.85	2.25	6.11	9.21	1.04	0.56	0.49	1.57	2.62
C15	2.50	13.14	15.64	3.20	0.75	2.44	6.27	9.47	1.07	0.57	0.50	1.92	2.99
C16	5.28	13.10	18.37	5.63	0.99	4.64	6.22	11.85	2.01	1.03	0.99	3.35	5.36
C17				7.00	4.58	2.42	6.09	13.09	1.36	0.87	0.49	1.55	2.91

Table D.126 Delete time of 20 MB. data size, 320 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.92	13.35	16.27	3.52	0.99	2.53	6.25	9.77	1.79	1.33	0.46	1.56	3.35
C3	2.81	13.39	16.20	36.62	33.66	2.96	6.24	42.86	2.01	1.56	0.46	1.56	3.58
C4	2.70	13.83	16.52	3.28	0.69	2.59	6.24	9.52	1.06	0.60	0.46	1.60	2.67
C5	2.48	13.83	16.32	3.56	0.95	2.61	6.21	9.77	1.78	1.31	0.46	1.31	3.08
C6	3.40	13.64	17.04	3.39	0.79	2.60	6.16	9.54	1.23	0.78	0.45	1.32	2.55
C7	3.02	14.03	17.05	3.84	1.45	2.39	6.25	10.09	1.25	0.79	0.46	1.59	2.84
C8	2.99	13.52	16.50	3.98	1.30	2.67	6.24	10.21	1.22	0.76	0.46	1.55	2.77
C9	3.13	13.59	16.71	3.57	0.89	2.68	6.18	9.75	1.07	0.61	0.45	1.31	2.38
C10	2.95	13.36	16.31	4.06	1.28	2.77	6.22	10.27	1.28	0.82	0.46	1.55	2.83
C12	2.95	13.86	16.81	3.13	0.65	2.48	6.27	9.40	1.19	0.73	0.46	1.56	2.75
C14	146.56	13.38	159.94	3.37	0.90	2.48	6.18	9.55	0.99	0.54	0.46	1.32	2.31
C15	2.57	13.85	16.42	3.56	1.00	2.56	6.25	9.80	1.00	0.55	0.45	1.59	2.59
C16	5.52	13.59	19.12	6.68	1.18	5.50	6.17	12.85	1.83	0.99	0.84	2.78	4.62
C17				7.41	4.86	2.55	6.13	13.54	1.31	0.86	0.45	1.31	2.62

Table D.127 Insert time of 20 MB. data size, 40 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.83	13.49	16.32	2.67	0.74	1.93	6.26	8.93	2.07	1.53	0.54	2.11	4.18
C3	2.74	13.26	16.00	3.50	1.06	2.44	6.25	9.75	2.40	1.86	0.54	2.11	4.50
C4	2.69	13.49	16.18	2.76	0.62	2.13	6.23	8.98	1.15	0.60	0.54	2.17	3.31
C5	2.38	13.11	15.49	2.95	0.81	2.15	6.14	9.10	2.25	1.70	0.55	1.79	4.04
C6	3.29	13.29	16.58	2.68	0.62	2.06	6.16	8.84	1.53	0.98	0.55	1.77	3.30
C7	2.99	13.32	16.32	3.15	1.10	2.05	6.21	9.36	1.44	0.89	0.55	2.15	3.59
C8	2.92	13.42	16.34	2.97	0.92	2.05	6.24	9.21	1.34	0.81	0.53	2.11	3.45
C9	3.04	13.33	16.36	2.90	0.78	2.11	6.13	9.03	1.19	0.64	0.55	1.76	2.95
C10	2.72	13.35	16.07	2.99	0.76	2.23	6.31	9.29	1.46	0.93	0.53	2.10	3.56
C12	2.86	13.40	16.26	2.59	0.59	2.00	6.26	8.85	1.44	0.82	0.62	2.10	3.54
C13_1	2.94	13.31	16.25	2.95	0.62	2.33	6.83	9.78	1.68	0.82	0.86	2.16	3.83
C13_2	2.94	13.30	16.23	3.05	0.62	2.42	6.81	9.86	1.70	0.84	0.85	2.16	3.85
C14	25.77	13.08	38.85	2.80	0.77	2.02	6.15	8.95	1.14	0.59	0.55	1.75	2.89
C15	2.55	13.44	16.00	2.88	0.64	2.23	6.23	9.10	1.15	0.60	0.54	2.15	3.30
C16	5.21	13.39	18.60	5.02	0.82	4.20	6.25	11.27	2.21	1.08	1.13	3.78	5.99
C17				6.08	3.92	2.16	6.14	12.21	1.57	1.02	0.55	1.75	3.32

Table D.128 Insert time of 20 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.84	13.22	16.06	3.00	0.86	2.14	6.25	9.25	2.02	1.50	0.53	2.02	4.05
C3	2.75	13.31	16.06	3.62	1.15	2.47	6.27	9.88	2.20	1.68	0.52	2.02	4.21
C4	2.70	13.16	15.86	2.98	0.63	2.35	6.24	9.22	1.12	0.59	0.52	2.07	3.19
C5	2.40	13.30	15.70	3.19	0.85	2.34	6.12	9.30	2.03	1.49	0.54	1.70	3.73
C6	3.31	13.30	16.60	2.97	0.63	2.34	6.15	9.12	1.47	0.93	0.54	1.68	3.15
C7	3.01	13.17	16.17	3.32	1.15	2.17	6.25	9.57	1.36	0.84	0.52	2.07	3.44
C8	2.93	13.15	16.08	3.36	0.97	2.39	6.25	9.61	1.28	0.76	0.52	2.02	3.30
C9	3.05	13.38	16.42	3.19	0.84	2.35	6.14	9.33	1.12	0.61	0.51	1.69	2.81
C10	2.74	13.33	16.07	3.22	0.78	2.43	6.27	9.49	1.37	0.86	0.51	2.02	3.39
C12	2.86	13.26	16.12	2.82	0.58	2.24	6.27	9.09	1.32	0.72	0.59	2.02	3.34
C13_1	2.96	13.27	16.23	3.08	0.64	2.44	6.80	9.88	1.61	0.76	0.85	2.05	3.66
C13_2	2.96	13.24	16.20	3.08	0.63	2.45	6.79	9.87	1.60	0.75	0.85	2.05	3.65
C14	42.54	13.24	55.78	3.04	0.82	2.23	6.13	9.17	1.11	0.57	0.54	1.69	2.79
C15	2.57	13.29	15.85	2.81	0.66	2.15	6.20	9.01	1.11	0.59	0.52	2.08	3.19
C16	5.26	13.26	18.52	5.23	0.85	4.38	6.21	11.44	2.08	1.03	1.05	3.63	5.71
C17				6.39	4.11	2.28	6.13	12.52	1.42	0.90	0.52	1.69	3.11

Table D.129 Insert time of 20 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.86	13.11	15.98	3.34	0.98	2.36	6.24	9.58	1.98	1.48	0.50	1.87	3.85
C3	2.78	13.06	15.84	3.96	1.29	2.67	6.31	10.27	2.21	1.71	0.50	1.87	4.08
C4	2.79	13.10	15.89	3.08	0.64	2.43	6.29	9.36	1.09	0.59	0.50	1.93	3.02
C5	2.43	13.07	15.50	3.29	0.86	2.42	6.12	9.41	1.96	1.45	0.51	1.56	3.53
C6	3.33	13.12	16.45	3.15	0.73	2.42	6.12	9.27	1.39	0.86	0.52	1.55	2.94
C7	3.08	13.03	16.11	3.42	1.19	2.23	6.29	9.71	1.33	0.82	0.50	1.93	3.25
C8	2.94	13.00	15.95	3.63	1.08	2.55	6.31	9.94	1.23	0.73	0.49	1.88	3.10
C9	3.06	13.21	16.27	3.41	0.89	2.52	6.15	9.56	1.08	0.59	0.48	1.56	2.64
C10	2.77	13.08	15.85	3.46	0.88	2.57	6.31	9.77	1.34	0.84	0.50	1.87	3.21
C12	2.90	13.18	16.07	3.00	0.63	2.37	6.20	9.20	1.21	0.70	0.50	1.87	3.08
C13_1	2.99	13.04	16.03	3.34	0.69	2.64	6.73	10.07	1.57	0.74	0.83	1.98	3.55
C13_2	2.98	13.06	16.04	3.36	0.69	2.66	6.72	10.07	1.54	0.72	0.82	1.98	3.53
C14	76.56	13.14	89.70	3.08	0.84	2.24	6.12	9.20	1.09	0.56	0.52	1.55	2.64
C15	2.65	13.20	15.85	3.20	0.76	2.44	6.34	9.54	1.09	0.59	0.50	1.92	3.01
C16	5.34	13.28	18.62	5.54	0.90	4.64	6.30	11.83	2.00	1.00	1.00	3.34	5.34
C17				6.74	4.30	2.44	6.10	12.84	1.32	0.83	0.49	1.56	2.88

Table D.130 Insert time of 20 MB. data size, 320 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.94	13.46	16.40	3.61	1.07	2.54	6.41	10.02	1.84	1.38	0.46	1.56	3.40
C3	2.93	13.33	16.26	4.33	1.33	3.00	6.18	10.50	2.02	1.56	0.46	1.56	3.58
C4	2.94	13.47	16.41	3.28	0.69	2.59	6.26	9.54	1.04	0.59	0.44	1.59	2.62
C5	2.52	13.61	16.14	3.54	0.94	2.60	6.21	9.75	1.81	1.35	0.46	1.31	3.11
C6	3.45	13.62	17.06	3.37	0.78	2.59	6.16	9.53	1.23	0.76	0.47	1.31	2.53
C7	3.27	13.82	17.09	3.77	1.38	2.39	6.28	10.05	1.24	0.79	0.45	1.59	2.83
C8	3.03	13.47	16.49	3.83	1.17	2.66	6.41	10.24	1.17	0.73	0.44	1.55	2.72
C9	3.15	13.70	16.85	3.62	0.93	2.69	6.16	9.78	1.07	0.61	0.46	1.31	2.38
C10	2.89	13.61	16.50	3.86	1.06	2.79	6.16	10.01	1.26	0.81	0.45	1.55	2.81
C12	3.00	13.74	16.74	3.14	0.65	2.48	6.26	9.40	1.17	0.73	0.43	1.56	2.72
C13_1	3.14	13.66	16.79	3.48	0.70	2.78	6.68	10.16	1.53	0.74	0.79	1.71	3.24
C13_2	3.15	13.56	16.71	3.49	0.70	2.79	6.68	10.18	1.55	0.75	0.79	1.71	3.25
C14	146.63	13.44	160.07	3.37	0.89	2.48	6.15	9.52	1.02	0.57	0.45	1.30	2.32
C15	2.82	13.93	16.75	3.58	1.02	2.56	6.25	9.83	1.04	0.59	0.44	1.59	2.63
C16	5.61	13.62	19.23	6.49	0.98	5.51	6.31	12.81	1.80	0.96	0.84	2.80	4.61
C17				7.27	4.72	2.55	6.13	13.39	1.47	1.01	0.46	1.32	2.78

IX) Detail-Table for data size 20 MB. in hot cache

Table D.131 Replace time of 20 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.84	13.13	15.96	2.61	0.69	1.92	6.29	8.91	1.26	0.72	0.53	2.10	3.36
C3	4.31	13.35	17.66	6.40	3.95	2.45	6.30	12.70	1.40	0.86	0.54	2.10	3.50
C4	8.84	13.22	22.05	2.73	0.58	2.15	6.18	8.90	1.11	0.57	0.54	2.15	3.27
C5	2.40	13.24	15.64	3.00	0.77	2.22	6.09	9.09	1.41	0.85	0.56	1.76	3.17
C6	3.31	13.22	16.53	2.72	0.59	2.13	6.13	8.85	1.27	0.72	0.55	1.77	3.04
C7	10.40	13.34	23.74	2.87	0.74	2.13	6.18	9.05	1.37	0.83	0.54	2.15	3.53
C8	2.92	13.27	16.19	2.86	0.74	2.12	6.32	9.18	1.26	0.72	0.54	2.11	3.36
C9	3.07	13.06	16.13	2.74	0.59	2.15	6.18	8.92	1.14	0.58	0.55	1.75	2.89
C10	4.29	13.29	17.58	3.08	0.74	2.34	6.22	9.30	1.39	0.85	0.54	2.11	3.50
C12	2.87	13.23	16.10	2.69	0.57	2.11	6.23	8.92	1.19	0.56	0.63	2.10	3.30
C14	25.30	13.19	38.49	2.60	0.58	2.02	6.15	8.75	1.12	0.57	0.55	1.76	2.88
C15	8.03	13.32	21.35	2.76	0.59	2.17	6.17	8.93	1.10	0.56	0.54	2.16	3.27
C16	5.29	13.34	18.63	5.05	0.67	4.39	6.21	11.26	2.10	0.96	1.14	3.78	5.88
C17				5.46	3.30	2.17	6.11	11.57	1.13	0.58	0.54	1.75	2.88

Table D.132 Replace time of 20 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.85	13.26	16.11	2.87	0.71	2.16	6.22	9.09	1.35	0.72	0.63	2.03	3.38
C3	4.31	13.17	17.48	10.30	7.74	2.56	6.26	16.56	1.48	0.85	0.63	2.02	3.49
C4	8.85	13.21	22.06	2.96	0.60	2.36	6.24	9.20	1.19	0.57	0.62	2.07	3.26
C5	2.42	13.22	15.64	3.17	0.80	2.37	6.13	9.30	1.40	0.84	0.55	1.70	3.10
C6	3.29	13.26	16.55	2.86	0.60	2.26	6.13	8.99	1.27	0.71	0.56	1.69	2.96
C7	10.42	13.30	23.72	2.96	0.79	2.16	6.24	9.20	1.45	0.81	0.63	2.07	3.52
C8	2.93	13.23	16.16	3.14	0.76	2.38	6.30	9.44	1.32	0.72	0.60	2.02	3.34
C9	3.08	13.17	16.26	3.00	0.60	2.39	6.15	9.14	1.14	0.58	0.55	1.69	2.83
C10	4.33	13.17	17.49	3.28	0.79	2.49	6.28	9.56	1.44	0.82	0.62	2.02	3.47
C12	2.88	13.34	16.22	2.70	0.58	2.12	6.25	8.95	1.09	0.56	0.53	2.04	3.13
C14	42.26	13.15	55.41	2.76	0.59	2.17	6.12	8.87	1.10	0.57	0.52	1.69	2.79
C15	8.03	13.32	21.34	2.76	0.60	2.16	6.23	9.00	1.19	0.56	0.63	2.07	3.26
C16	5.31	13.25	18.56	5.15	0.70	4.45	6.22	11.37	2.01	0.95	1.06	3.62	5.63
C17				5.71	3.38	2.34	6.13	11.84	1.22	0.58	0.64	1.69	2.91

Table D.133 Replace time of 20 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.87	13.10	15.97	3.12	0.75	2.37	6.26	9.37	1.20	0.70	0.50	1.88	3.08
C3	4.39	13.18	17.57	17.87	15.29	2.58	6.32	24.19	1.33	0.83	0.50	1.88	3.20
C4	8.99	13.21	22.20	3.07	0.62	2.45	6.29	9.36	1.07	0.56	0.50	1.93	2.99
C5	2.41	13.15	15.56	3.30	0.83	2.47	6.12	9.42	1.40	0.82	0.57	1.57	2.96
C6	3.33	13.27	16.60	3.08	0.64	2.44	6.13	9.21	1.31	0.72	0.58	1.56	2.87
C7	10.47	13.16	23.63	3.07	0.81	2.26	6.33	9.40	1.30	0.79	0.50	1.92	3.22
C8	2.97	13.12	16.09	3.34	0.80	2.54	6.27	9.61	1.20	0.70	0.50	1.88	3.07
C9	3.09	13.08	16.18	3.19	0.63	2.56	6.12	9.31	1.17	0.58	0.59	1.56	2.73
C10	4.38	13.23	17.60	3.41	0.81	2.60	6.33	9.74	1.31	0.81	0.50	1.87	3.19
C12	2.92	13.39	16.31	2.90	0.61	2.29	6.22	9.12	1.08	0.56	0.51	1.90	2.98
C14	77.14	13.25	90.39	2.88	0.62	2.26	6.10	8.98	1.15	0.57	0.58	1.56	2.70
C15	8.20	13.17	21.36	3.08	0.65	2.43	6.30	9.38	1.06	0.56	0.50	1.93	2.99
C16	5.35	13.10	18.45	5.38	0.71	4.67	6.30	11.68	1.92	0.92	1.00	3.34	5.26
C17				5.97	3.55	2.42	6.14	12.11	1.16	0.58	0.58	1.56	2.72

Table D.134 Replace time of 20 MB. data size, 320 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.98	13.43	16.41	3.48	0.93	2.55	6.64	10.12	1.43	0.68	0.75	1.56	2.99
C3	4.59	13.39	17.98	32.32	29.37	2.95	6.68	39.00	1.54	0.78	0.75	1.55	3.09
C4	9.32	13.48	22.80	3.26	0.67	2.60	6.26	9.52	1.32	0.56	0.75	1.59	2.91
C5	2.50	13.36	15.86	3.51	0.90	2.61	6.15	9.67	1.23	0.77	0.45	1.31	2.54
C6	3.44	13.48	16.92	3.29	0.70	2.59	6.23	9.52	1.14	0.67	0.46	1.32	2.45
C7	10.91	13.58	24.49	3.31	0.92	2.38	6.52	9.83	1.52	0.76	0.76	1.60	3.12
C8	3.06	13.55	16.61	3.57	0.85	2.72	6.66	10.23	1.44	0.69	0.75	1.56	3.00
C9	3.17	13.38	16.55	3.41	0.67	2.74	6.43	9.84	1.04	0.58	0.47	1.34	2.39
C10	4.52	13.38	17.90	3.67	0.85	2.81	6.16	9.82	1.53	0.78	0.75	1.57	3.10
C12	3.03	13.82	16.85	3.15	0.65	2.50	6.53	9.68	1.01	0.56	0.45	1.57	2.58
C14	145.90	13.35	159.25	3.28	0.82	2.46	6.41	9.69	1.04	0.57	0.47	1.31	2.35
C15	8.54	13.59	22.12	3.34	0.75	2.59	6.51	9.85	1.32	0.56	0.75	1.60	2.91
C16	5.53	13.37	18.89	6.34	0.80	5.54	6.34	12.68	1.72	0.86	0.86	2.79	4.51
C17				6.73	4.16	2.57	6.40	13.13	1.04	0.58	0.46	1.31	2.35

Table D.135 Delete time of 20 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.82	13.12	15.95	2.61	0.69	1.92	6.16	8.78	1.36	0.72	0.64	2.10	3.46
C3	2.71	13.10	15.81	6.27	3.91	2.36	6.16	12.42	1.49	0.86	0.63	2.10	3.59
C4	2.62	13.28	15.90	2.60	0.59	2.01	6.17	8.77	1.50	0.58	0.92	2.15	3.66
C5	2.39	13.58	15.96	2.91	0.78	2.12	6.09	8.99	1.40	0.85	0.54	1.76	3.15
C6	3.30	13.24	16.54	2.74	0.59	2.15	6.10	8.83	1.26	0.71	0.55	1.76	3.02
C7	2.94	13.20	16.14	2.77	0.75	2.02	6.18	8.95	1.76	0.83	0.93	2.14	3.90
C8	2.89	13.25	16.14	2.91	0.74	2.17	6.20	9.11	1.36	0.72	0.63	2.09	3.45
C9	3.03	13.13	16.16	2.76	0.60	2.16	6.08	8.84	1.14	0.58	0.55	1.75	2.89
C10	2.70	13.13	15.83	3.03	0.75	2.28	6.18	9.21	1.48	0.85	0.63	2.09	3.57
C12	2.84	13.17	16.01	2.56	0.57	1.99	6.21	8.77	1.18	0.56	0.62	2.09	3.27
C14	25.32	13.14	38.46	2.50	0.58	1.92	6.08	8.59	1.12	0.57	0.55	1.75	2.87
C15	2.48	13.18	15.66	2.75	0.60	2.16	6.18	8.93	1.49	0.56	0.92	2.15	3.64
C16	5.25	13.28	18.53	4.94	0.65	4.28	6.20	11.13	2.09	0.96	1.13	3.77	5.87
C17				5.40	3.29	2.11	6.09	11.49	1.13	0.58	0.54	1.76	2.89

Table D.136 Delete time of 20 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.84	13.23	16.07	2.88	0.72	2.16	6.27	9.15	1.30	0.71	0.58	2.02	3.32
C3	2.72	13.11	15.82	10.21	7.73	2.47	6.26	16.47	1.44	0.85	0.59	2.02	3.47
C4	2.63	13.27	15.90	2.98	0.60	2.38	6.21	9.19	1.47	0.57	0.89	2.07	3.54
C5	2.34	13.49	15.83	3.15	0.80	2.35	6.14	9.29	1.39	0.84	0.55	1.69	3.08
C6	3.28	13.27	16.55	2.90	0.60	2.30	6.14	9.03	1.26	0.71	0.55	1.69	2.95
C7	2.88	13.35	16.23	2.92	0.77	2.15	6.23	9.15	1.71	0.81	0.89	2.06	3.77
C8	2.89	13.31	16.20	3.23	0.76	2.47	6.24	9.47	1.26	0.72	0.54	2.02	3.28
C9	3.03	13.22	16.25	2.98	0.61	2.36	6.13	9.11	1.12	0.58	0.53	1.69	2.81
C10	2.70	13.13	15.83	3.24	0.76	2.48	6.25	9.49	1.42	0.84	0.59	2.02	3.44
C12	2.85	13.38	16.23	2.84	0.59	2.25	6.25	9.09	1.12	0.56	0.56	2.01	3.13
C14	42.39	13.39	55.78	2.82	0.59	2.23	6.13	8.95	1.10	0.57	0.53	1.69	2.79
C15	2.49	13.32	15.82	2.78	0.61	2.17	6.23	9.01	1.45	0.56	0.88	2.07	3.52
C16	5.24	13.20	18.44	5.10	0.70	4.41	6.18	11.28	1.99	0.95	1.04	3.63	5.62
C17				5.65	3.37	2.28	6.12	11.78	1.12	0.58	0.54	1.68	2.80

Table D.137 Delete time of 20 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.86	13.08	15.94	3.13	0.75	2.37	6.19	9.32	1.20	0.70	0.50	1.87	3.07
C3	2.73	13.08	15.81	17.96	15.29	2.67	6.18	24.14	1.32	0.82	0.50	1.88	3.20
C4	2.64	13.17	15.81	3.05	0.63	2.42	6.29	9.34	1.09	0.57	0.51	1.92	3.01
C5	2.41	13.28	15.69	3.27	0.84	2.43	6.13	9.40	1.34	0.82	0.52	1.55	2.89
C6	3.30	13.36	16.66	3.07	0.64	2.44	6.13	9.20	1.23	0.71	0.52	1.55	2.78
C7	2.95	13.13	16.07	2.99	0.81	2.18	6.24	9.24	1.29	0.79	0.49	1.92	3.21
C8	2.90	13.11	16.01	3.39	0.80	2.59	6.19	9.58	1.20	0.70	0.50	1.87	3.07
C9	3.05	13.18	16.23	3.18	0.64	2.54	6.15	9.33	1.17	0.69	0.48	1.56	2.73
C10	2.72	13.09	15.81	3.43	0.80	2.62	6.18	9.60	1.31	0.81	0.50	1.87	3.18
C12	2.88	13.46	16.34	2.87	0.61	2.26	6.22	9.09	1.14	0.63	0.51	1.87	3.01
C14	76.87	13.30	90.17	2.87	0.61	2.26	6.12	8.99	1.19	0.70	0.49	1.55	2.74
C15	2.50	13.32	15.82	3.09	0.65	2.44	6.22	9.31	1.06	0.56	0.50	1.91	2.97
C16	5.32	13.23	18.55	5.34	0.71	4.63	6.21	11.55	1.91	0.92	0.99	3.34	5.25
C17				5.96	3.56	2.41	6.09	12.05	1.18	0.66	0.52	1.55	2.73

Table D.138 Delete time of 20 MB. data size, 320 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.94	13.64	16.58	3.34	0.83	2.51	6.23	9.57	1.14	0.67	0.46	1.55	2.69
C3	2.81	13.70	16.51	31.81	28.92	2.89	6.20	38.01	1.23	0.77	0.46	1.55	2.78
C4	2.72	13.82	16.55	3.26	0.67	2.59	6.25	9.51	1.03	0.56	0.47	1.59	2.62
C5	2.51	13.67	16.18	3.50	0.91	2.59	6.16	9.66	1.22	0.76	0.46	1.31	2.54
C6	3.42	13.85	17.28	3.27	0.70	2.56	6.15	9.41	1.13	0.67	0.45	1.30	2.43
C7	3.03	13.73	16.76	3.29	0.86	2.43	6.25	9.54	1.34	0.88	0.46	1.59	2.93
C8	2.99	13.70	16.69	3.50	0.86	2.64	6.22	9.71	1.20	0.74	0.46	1.55	2.75
C9	3.14	13.70	16.85	3.31	0.67	2.63	6.16	9.47	1.03	0.57	0.46	1.31	2.34
C10	2.79	13.59	16.39	3.63	0.86	2.77	6.21	9.84	1.25	0.79	0.46	1.55	2.80
C12	2.98	13.71	16.69	3.15	0.65	2.50	6.25	9.40	1.03	0.56	0.46	1.56	2.58
C14	146.81	13.32	160.13	3.15	0.65	2.49	6.16	9.31	1.03	0.56	0.47	1.31	2.34
C15	2.59	13.42	16.01	3.23	0.71	2.52	6.25	9.48	1.01	0.56	0.45	1.58	2.59
C16	5.56	13.43	18.99	6.19	0.81	5.39	6.18	12.37	1.71	0.86	0.85	2.79	4.50
C17				6.48	3.96	2.52	6.14	12.62	1.05	0.58	0.46	1.31	2.35

Table D.139 Insert time of 20 MB. data size, 40 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.82	13.23	16.05	2.62	0.69	1.93	6.20	8.82	1.26	0.72	0.54	2.10	3.37
C3	2.73	13.15	15.88	3.12	0.69	2.43	6.24	9.37	1.27	0.73	0.54	2.10	3.37
C4	2.69	13.28	15.97	2.71	0.58	2.13	6.17	8.88	1.11	0.57	0.53	2.15	3.26
C5	2.41	13.24	15.65	2.91	0.79	2.13	6.09	9.00	1.41	0.85	0.56	1.76	3.17
C6	3.29	13.18	16.48	2.75	0.59	2.16	6.09	8.85	1.27	0.71	0.55	1.76	3.03
C7	2.99	13.23	16.22	2.90	0.75	2.15	6.18	9.08	1.37	0.82	0.54	2.15	3.52
C8	2.92	13.23	16.15	2.83	0.74	2.08	6.22	9.05	1.25	0.72	0.53	2.10	3.35
C9	3.04	13.14	16.18	2.72	0.60	2.12	6.10	8.82	1.13	0.58	0.54	1.75	2.88
C10	2.72	13.14	15.86	2.91	0.68	2.23	6.23	9.14	1.39	0.85	0.54	2.09	3.49
C12	2.86	13.31	16.17	2.61	0.57	2.03	6.21	8.82	1.19	0.57	0.62	2.10	3.29
C13_1	2.90	13.16	16.07	2.84	0.58	2.26	6.80	9.64	1.42	0.57	0.85	2.13	3.55
C13_2	2.92	13.12	16.04	2.96	0.58	2.38	6.79	9.75	1.42	0.56	0.85	2.14	3.55
C14	25.55	13.04	38.58	2.69	0.58	2.11	6.10	8.79	1.11	0.57	0.54	1.75	2.85
C15	2.54	13.20	15.73	2.72	0.59	2.14	6.18	8.90	1.10	0.56	0.54	2.15	3.25
C16	5.26	13.28	18.54	4.75	0.64	4.11	6.20	10.95	2.10	0.96	1.14	3.78	5.87
C17				5.44	3.29	2.15	6.08	11.52	1.13	0.58	0.54	1.75	2.87

Table D.140 Insert time of 20 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.82	13.17	15.99	2.93	0.71	2.22	6.21	9.14	1.34	0.71	0.63	2.02	3.37
C3	2.74	13.24	15.98	3.15	0.70	2.45	6.25	9.40	1.35	0.73	0.62	2.01	3.36
C4	2.70	13.11	15.81	2.93	0.60	2.33	6.24	9.16	1.19	0.57	0.62	2.07	3.25
C5	2.41	13.24	15.65	3.13	0.79	2.33	6.13	9.26	1.40	0.84	0.56	1.69	3.09
C6	3.32	13.18	16.50	2.92	0.60	2.32	6.12	9.04	1.27	0.71	0.56	1.69	2.95
C7	2.97	13.14	16.11	2.93	0.76	2.17	6.23	9.16	1.44	0.82	0.62	2.07	3.51
C8	2.93	13.09	16.02	3.14	0.76	2.38	6.28	9.42	1.32	0.71	0.60	2.02	3.33
C9	3.05	13.39	16.44	2.95	0.61	2.33	6.14	9.08	1.13	0.58	0.55	1.69	2.82
C10	2.73	13.38	16.11	3.13	0.70	2.42	6.24	9.37	1.45	0.83	0.62	2.01	3.46
C12	2.86	13.22	16.08	2.75	0.59	2.16	6.25	9.00	1.08	0.56	0.51	2.03	3.11
C13_1	2.96	13.27	16.23	3.04	0.59	2.45	6.82	9.86	1.43	0.56	0.86	2.04	3.47
C13_2	2.97	13.22	16.19	3.03	0.58	2.45	6.82	9.85	1.43	0.56	0.87	2.06	3.49
C14	42.45	13.22	55.68	2.79	0.59	2.20	6.13	8.92	1.12	0.57	0.55	1.68	2.80
C15	2.56	13.24	15.80	2.78	0.61	2.17	6.23	9.01	1.18	0.56	0.62	2.07	3.25
C16	5.27	13.23	18.51	5.03	0.67	4.37	6.22	11.25	2.00	0.94	1.06	3.62	5.62
C17				5.63	3.36	2.26	6.13	11.75	1.22	0.58	0.64	1.69	2.91

Table D.141 Insert time of 20 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.83	13.19	16.02	3.11	0.75	2.36	6.25	9.36	1.20	0.70	0.50	1.87	3.08
C3	2.80	13.19	15.99	3.40	0.73	2.67	6.33	9.72	1.21	0.71	0.50	1.86	3.07
C4	2.77	13.15	15.92	3.05	0.62	2.42	6.32	9.37	1.07	0.57	0.50	1.92	2.98
C5	2.43	13.13	15.55	3.26	0.84	2.42	6.14	9.41	1.40	0.81	0.58	1.55	2.95
C6	3.32	13.21	16.53	3.05	0.64	2.41	6.13	9.18	1.31	0.71	0.59	1.56	2.86
C7	3.09	13.12	16.20	3.03	0.80	2.23	6.31	9.34	1.30	0.79	0.50	1.92	3.22
C8	2.94	13.07	16.01	3.33	0.80	2.54	6.28	9.61	1.20	0.70	0.49	1.87	3.06
C9	3.05	13.28	16.33	3.16	0.64	2.52	6.15	9.31	1.16	0.57	0.58	1.56	2.72
C10	2.77	13.18	15.95	3.32	0.74	2.58	6.32	9.64	1.31	0.81	0.49	1.87	3.18
C12	2.90	13.24	16.14	2.85	0.61	2.23	6.23	9.07	1.07	0.56	0.50	1.87	2.93
C13_1	3.03	13.04	16.07	3.28	0.61	2.67	6.77	10.05	1.41	0.56	0.85	1.97	3.38
C13_2	3.05	13.16	16.21	3.28	0.61	2.67	6.77	10.05	1.42	0.56	0.85	1.96	3.38
C14	76.73	13.39	90.12	2.83	0.61	2.21	6.12	8.95	1.15	0.56	0.59	1.55	2.70
C15	2.65	13.35	16.00	2.93	0.64	2.28	6.30	9.23	1.06	0.56	0.50	1.92	2.98
C16	5.38	13.41	18.79	5.32	0.70	4.61	6.30	11.61	1.93	0.92	1.01	3.33	5.26
C17				6.00	3.59	2.42	6.10	12.10	1.16	0.57	0.59	1.55	2.71

Table D.142 Insert time of 20 MB. data size, 320 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	2.95	13.59	16.54	3.35	0.83	2.52	6.36	9.72	1.43	0.68	0.75	1.55	2.98
C3	2.92	13.57	16.48	3.77	0.78	2.99	6.17	9.94	1.45	0.71	0.74	1.55	3.00
C4	2.96	13.65	16.61	3.23	0.67	2.56	6.27	9.50	1.30	0.56	0.74	1.59	2.89
C5	2.55	13.62	16.17	3.50	0.90	2.60	6.16	9.66	1.23	0.77	0.45	1.31	2.53
C6	3.48	13.51	16.98	3.29	0.70	2.59	6.15	9.44	1.13	0.67	0.46	1.30	2.43
C7	3.28	13.56	16.84	3.28	0.88	2.40	6.26	9.55	1.52	0.77	0.75	1.59	3.11
C8	3.03	13.64	16.67	3.47	0.85	2.62	6.39	9.86	1.44	0.69	0.74	1.55	2.99
C9	3.15	13.68	16.83	3.36	0.68	2.68	6.16	9.52	1.03	0.57	0.46	1.31	2.34
C10	2.92	13.59	16.51	3.56	0.80	2.75	6.14	9.70	1.53	0.79	0.74	1.55	3.07
C12	3.05	13.71	16.76	3.14	0.65	2.49	6.26	9.40	0.99	0.56	0.43	1.56	2.55
C13_1	3.15	13.34	16.48	3.43	0.64	2.79	6.68	10.11	1.35	0.56	0.79	1.70	3.05
C13_2	3.16	13.35	16.51	3.43	0.63	2.80	6.68	10.11	1.35	0.56	0.78	1.70	3.04
C14	146.70	13.46	160.16	3.13	0.65	2.48	6.14	9.27	1.01	0.56	0.45	1.31	2.32
C15	2.82	13.63	16.45	3.28	0.72	2.56	6.25	9.53	1.30	0.55	0.75	1.59	2.89
C16	5.64	13.27	18.92	6.28	0.76	5.52	6.33	12.61	1.71	0.86	0.84	2.79	4.50
C17				6.57	4.03	2.54	6.14	12.71	1.02	0.56	0.46	1.31	2.33

X) Detail-Table for 40 MB. data size in cold cache

Table D.143 Replace time of 40 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.15	26.35	31.51	9.39	4.81	4.59	12.34	21.74	11.10	10.24	0.86	4.10	15.19
C3	8.21	26.24	34.45	17.87	13.27	4.61	12.37	30.25	11.07	10.25	0.82	4.09	15.16
C4	17.18	26.59	43.78	7.95	3.37	4.58	12.30	20.25	4.07	3.21	0.86	4.21	8.28
C5	4.33	26.53	30.85	9.17	4.62	4.55	12.16	21.33	7.93	7.04	0.90	3.43	11.36
C6	6.21	26.05	32.26	8.11	3.37	4.73	12.15	20.25	14.09	13.20	0.89	3.42	17.51
C7	20.15	26.35	46.50	12.47	7.84	4.63	12.31	24.78	12.09	11.30	0.79	4.18	16.27
C8	5.33	26.58	31.91	8.35	3.72	4.63	12.41	20.76	10.05	9.22	0.83	4.08	14.13
C9	5.57	26.24	31.80	7.80	3.17	4.63	12.15	19.95	4.09	3.22	0.88	3.42	7.51
C10	8.15	26.33	34.48	8.36	3.72	4.64	12.35	20.71	12.37	11.54	0.82	4.08	16.44
C12	5.26	26.60	31.86	8.01	3.28	4.73	12.40	20.41	4.04	3.20	0.84	4.09	8.12
C14	85.54	26.33	111.88	7.76	3.13	4.63	12.17	19.94	4.03	3.21	0.82	3.41	7.43
C15	15.56	26.27	41.83	8.27	3.64	4.63	12.30	20.57	4.11	3.31	0.80	4.20	8.30
C16	10.03	26.20	36.23	11.98	3.53	8.45	12.39	24.37	13.99	12.02	1.96	7.42	21.41
C17				14.19	11.01	3.18	12.10	26.29	4.47	3.66	0.81	3.40	7.87

Table D.144 Replace time of 40 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.22	26.14	31.36	9.93	5.16	4.77	12.37	22.30	8.12	7.29	0.84	3.97	12.09
C3	8.22	25.94	34.16	30.44	25.58	4.87	12.51	42.95	8.67	7.87	0.80	3.97	12.64
C4	17.24	26.19	43.43	8.09	3.37	4.72	12.35	20.44	3.90	3.08	0.82	4.05	7.94
C5	4.37	26.35	30.72	9.62	4.87	4.75	12.11	21.73	6.59	5.77	0.82	3.28	9.87
C6	6.26	26.40	32.65	8.29	3.48	4.81	12.09	20.38	11.68	10.84	0.83	3.28	14.96
C7	20.28	26.26	46.54	12.82	8.05	4.78	12.30	25.12	9.87	9.07	0.79	4.05	13.91
C8	5.42	26.24	31.67	8.61	3.80	4.81	12.38	20.99	8.02	7.12	0.89	3.97	11.99
C9	5.66	26.33	31.99	8.02	3.17	4.84	12.19	20.20	3.92	3.13	0.79	3.27	7.19
C10	8.22	26.08	34.31	8.53	3.79	4.74	12.46	20.99	10.36	9.57	0.79	3.93	14.29
C12	5.33	27.13	32.46	8.20	3.37	4.83	12.35	20.55	3.87	3.09	0.78	3.93	7.80
C14	150.95	26.17	177.12	7.95	3.16	4.79	12.16	20.11	3.92	3.09	0.83	3.33	7.25
C15	15.63	26.29	41.92	8.45	3.71	4.75	12.38	20.83	4.00	3.21	0.79	4.04	8.04
C16	10.06	26.22	36.29	13.04	3.80	9.23	12.51	25.55	11.14	9.33	1.82	7.19	18.33
C17				13.98	10.45	3.53	12.11	26.09	4.22	3.43	0.79	3.28	7.50

Table D.145 Replace time of 40 MB. data size, 320 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.33	26.38	31.71	10.52	5.47	5.05	12.47	22.99	5.79	5.03	0.76	3.63	9.42
C3	8.34	26.27	34.61	53.29	48.30	4.99	12.62	65.91	8.36	7.60	0.76	3.62	11.98
C4	17.45	26.49	43.94	9.33	4.07	5.26	12.33	21.66	3.76	2.99	0.76	3.72	7.47
C5	4.43	26.37	30.80	10.51	5.46	5.06	12.13	22.65	6.04	5.26	0.78	3.01	9.05
C6	6.28	26.36	32.64	8.64	3.61	5.02	12.13	20.76	11.56	10.77	0.79	3.01	14.57
C7	20.53	26.08	46.61	13.32	8.27	5.05	12.34	25.66	9.53	8.77	0.76	3.72	13.25
C8	5.46	26.49	31.95	9.09	4.11	4.98	12.52	21.61	7.75	6.99	0.76	3.62	11.37
C9	5.71	26.48	32.19	8.37	3.29	5.08	12.13	20.50	3.72	2.94	0.77	3.03	6.75
C10	8.37	26.14	34.51	9.27	4.08	5.18	12.59	21.85	9.54	8.78	0.76	3.61	13.16
C12	5.42	27.04	32.46	8.52	3.45	5.08	12.39	20.92	3.66	2.95	0.71	3.65	7.30
C14	286.91	26.01	312.92	8.36	3.30	5.06	12.09	20.45	3.75	2.99	0.76	3.02	6.77
C15	15.94	26.26	42.20	9.14	3.95	5.19	12.34	21.48	3.75	2.99	0.76	3.73	7.47
C16	10.31	26.18	36.50	13.76	4.08	9.68	12.45	26.21	10.86	9.21	1.66	6.54	17.40
C17				15.20	11.52	3.68	12.17	27.37	4.06	3.30	0.76	3.01	7.07

Table D.146 Replace time of 40 MB. data size, 640 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.38	26.77	32.15	11.43	5.88	5.55	12.66	24.08	7.71	7.07	0.65	3.02	10.73
C3	8.52	26.33	34.85	102.89	97.26	5.63	12.88	115.77	7.80	7.15	0.65	3.00	10.80
C4	17.78	26.59	44.37	9.51	4.19	5.32	12.56	22.07	3.71	3.06	0.65	3.10	6.81
C5	4.44	26.66	31.10	11.68	5.95	5.72	12.17	23.84	5.03	4.37	0.67	2.54	7.57
C6	6.31	26.61	32.91	9.74	4.09	5.64	12.27	22.01	10.52	9.84	0.68	2.49	13.02
C7	20.76	26.67	47.43	14.56	9.14	5.43	12.36	26.93	8.83	8.16	0.66	3.10	11.93
C8	5.52	26.57	32.09	10.21	4.48	5.74	12.71	22.92	7.53	6.88	0.65	3.02	10.56
C9	5.78	26.82	32.60	9.44	3.82	5.62	12.23	21.67	3.52	2.85	0.67	2.50	6.03
C10	8.40	26.46	34.86	10.09	4.51	5.58	12.84	22.93	9.12	8.46	0.66	3.01	12.13
C12	5.46	27.20	32.67	9.88	4.40	5.48	12.48	22.36	3.65	3.00	0.65	3.02	6.67
C14	551.72	26.25	557.97	9.47	3.69	5.78	12.30	21.77	3.53	2.86	0.66	2.50	6.03
C15	16.23	26.46	42.68	9.99	4.55	5.44	12.37	22.36	3.52	2.87	0.65	3.11	6.63
C16	10.36	26.51	36.87	15.16	4.21	10.95	12.69	27.84	10.26	8.76	1.50	5.42	15.68
C17				17.94	14.01	3.93	12.31	30.25	3.81	3.14	0.67	2.49	6.30

Table D.147 Delete time of 40 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.24	26.25	31.50	9.70	5.12	4.58	12.27	21.97	9.88	8.99	0.89	4.09	13.96
C3	4.92	26.00	30.93	18.12	13.55	4.58	12.25	30.38	10.68	9.88	0.80	4.09	14.77
C4	4.81	26.52	31.34	7.93	3.36	4.56	12.30	20.22	4.11	3.27	0.84	4.20	8.31
C5	4.36	26.51	30.88	9.27	4.73	4.54	12.15	21.42	7.25	6.35	0.90	3.41	10.66
C6	6.20	26.25	32.45	8.14	3.43	4.72	12.13	20.27	13.90	13.01	0.89	3.42	17.32
C7	5.43	26.22	31.65	12.11	7.50	4.62	12.35	24.47	12.15	11.34	0.81	4.19	16.34
C8	5.34	26.10	31.44	8.26	3.65	4.61	12.33	20.59	9.67	8.87	0.80	4.09	13.76
C9	5.60	26.10	31.70	8.15	3.16	4.99	12.16	20.31	4.20	3.31	0.89	3.41	7.61
C10	4.94	26.17	31.11	8.31	3.67	4.64	12.27	20.58	12.33	11.53	0.80	4.09	16.42
C12	5.30	26.66	31.96	8.01	3.33	4.69	12.35	20.37	4.06	3.24	0.82	4.09	8.14
C14	85.70	26.20	111.90	7.79	3.16	4.63	12.05	19.83	4.02	3.20	0.82	3.42	7.44
C15	4.50	26.39	30.88	8.25	3.65	4.60	12.32	20.57	4.18	3.37	0.81	4.19	8.37
C16	10.07	26.21	36.29	11.92	3.52	8.40	12.31	24.22	13.67	11.71	1.96	7.43	21.10
C17				13.32	10.16	3.16	12.08	25.40	4.52	3.70	0.82	3.40	7.92

Table D.148 Delete time of 40 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.28	26.06	31.34	10.33	5.58	4.75	12.35	22.68	8.10	7.26	0.84	3.94	12.04
C3	3.46	26.20	29.66	31.06	26.21	4.84	12.30	43.36	8.64	7.83	0.81	3.94	12.58
C4	4.83	26.28	31.10	8.12	3.39	4.73	12.33	20.45	3.88	3.11	0.77	4.03	7.91
C5	4.39	25.95	30.33	9.52	4.79	4.73	12.17	21.68	6.65	5.81	0.84	3.28	9.93
C6	6.26	26.40	32.66	8.21	3.45	4.76	12.15	20.36	11.84	11.02	0.82	3.31	15.15
C7	5.45	26.04	31.49	12.38	7.64	4.74	12.36	24.74	9.73	8.88	0.85	4.04	13.77
C8	5.35	26.15	31.50	8.47	3.69	4.77	12.37	20.83	8.21	7.37	0.84	3.96	12.17
C9	5.59	26.21	31.80	8.02	3.17	4.85	12.16	20.18	3.99	3.16	0.83	3.29	7.28
C10	4.97	26.08	31.05	8.45	3.72	4.73	12.35	20.80	9.93	9.10	0.84	3.94	13.87
C12	5.34	26.49	31.84	8.15	3.36	4.80	12.40	20.55	3.83	3.06	0.77	3.92	7.75
C14	151.06	25.99	177.05	8.04	3.26	4.77	12.13	20.17	3.93	3.11	0.82	3.31	7.24
C15	4.51	26.11	30.61	8.53	3.80	4.73	12.34	20.86	4.17	3.34	0.83	4.02	8.19
C16	10.10	26.25	36.35	12.85	3.74	9.11	12.40	25.25	11.26	9.45	1.81	7.19	18.45
C17				13.97	10.47	3.50	12.11	26.08	4.30	3.47	0.83	3.28	7.58

Table D.149 Delete time of 40 MB. data size, 320 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.31	26.07	31.38	11.06	6.07	4.99	12.34	23.40	8.02	7.26	0.76	3.64	11.66
C3	5.02	25.91	30.93	54.47	49.52	4.95	12.29	66.76	8.28	7.52	0.75	3.63	11.90
C4	4.87	26.21	31.08	8.67	3.51	5.16	12.33	21.00	4.46	3.06	1.40	3.71	8.17
C5	4.41	26.03	30.44	10.97	5.87	5.10	12.13	23.09	5.99	5.21	0.77	3.02	9.01
C6	6.38	26.04	32.43	8.58	3.60	4.97	12.12	20.70	11.73	10.95	0.78	3.02	14.75
C7	5.49	26.01	31.50	13.12	8.12	5.00	12.43	25.55	9.55	8.79	0.76	3.72	13.26
C8	5.41	26.22	31.63	9.02	4.06	4.96	12.32	21.34	7.92	7.17	0.75	3.65	11.57
C9	5.69	26.30	31.99	8.29	3.25	5.04	12.14	20.42	3.77	2.99	0.78	3.01	6.78
C10	4.98	26.22	31.20	9.13	4.01	5.12	12.25	21.38	9.63	8.85	0.77	3.62	13.25
C12	5.36	26.32	31.68	8.41	3.45	4.96	12.40	20.81	3.68	2.93	0.75	3.63	7.31
C14	286.12	25.81	311.93	8.25	3.29	4.97	12.13	20.38	3.74	2.96	0.78	3.03	6.77
C15	4.54	26.10	30.64	8.95	3.88	5.06	12.35	21.29	3.81	3.06	0.75	3.72	7.53
C16	10.21	26.03	36.24	13.66	4.04	9.62	12.28	25.94	11.14	9.46	1.68	6.57	17.71
C17				14.32	10.69	3.63	12.15	26.47	4.04	3.27	0.77	3.04	7.08

Table D.150 Delete time of 40 MB. data size, 640 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.29	26.29	31.58	11.06	6.19	4.88	12.44	23.50	7.80	7.16	0.64	3.02	10.81
C3	5.11	26.53	31.65	104.30	98.66	5.64	12.44	116.74	7.80	7.15	0.66	3.02	10.82
C4	4.91	26.31	31.23	9.13	3.84	5.29	12.61	21.73	3.66	3.02	0.64	3.09	6.75
C5	4.48	26.66	31.13	11.62	5.92	5.70	12.31	23.92	5.22	4.55	0.67	2.53	7.75
C6	6.45	26.35	32.79	9.71	4.12	5.60	12.36	22.07	11.04	9.82	1.22	2.51	13.55
C7	5.55	26.36	31.91	14.99	9.60	5.39	12.59	27.58	8.97	8.32	0.65	3.09	12.06
C8	5.42	26.71	32.13	10.24	4.55	5.69	12.49	22.73	6.90	6.26	0.65	3.01	9.92
C9	5.71	26.40	32.11	9.43	3.88	5.56	12.31	21.74	3.50	2.84	0.67	2.52	6.02
C10	5.02	26.17	31.19	9.90	4.52	5.38	12.41	22.31	9.21	8.57	0.64	3.01	12.22
C12	5.39	26.73	32.12	9.19	3.77	5.42	12.51	21.70	3.39	2.75	0.64	3.00	6.39
C14	550.99	26.05	557.04	9.54	3.80	5.74	12.24	21.79	3.65	3.00	0.66	2.51	6.17
C15	4.62	26.32	30.94	9.83	4.45	5.39	12.54	22.37	3.71	3.06	0.65	3.09	6.80
C16	10.42	26.13	36.54	15.12	4.26	10.87	12.30	27.43	10.43	9.07	1.35	5.41	15.84
C17				15.71	11.80	3.91	12.27	27.99	3.82	3.15	0.67	2.51	6.33

Table D.151 Insert time of 40 MB. data size, 80 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.23	26.18	31.41	10.18	5.60	4.58	12.30	22.48	9.46	8.60	0.86	4.09	13.54
C3	4.98	26.14	31.12	9.53	4.96	4.57	12.35	21.88	9.92	9.10	0.83	4.08	14.00
C4	4.87	26.21	31.08	7.91	3.35	4.57	12.28	20.19	4.14	3.28	0.86	4.20	8.34
C5	4.36	26.13	30.49	9.16	4.63	4.53	12.15	21.31	7.28	6.39	0.90	3.41	10.69
C6	6.22	26.23	32.45	8.16	3.45	4.71	12.14	20.29	14.28	13.39	0.89	3.41	17.69
C7	5.52	26.16	31.69	12.45	7.82	4.63	12.27	24.72	11.89	11.09	0.80	4.20	16.08
C8	5.42	26.20	31.63	8.26	3.65	4.61	12.39	20.65	9.86	9.03	0.83	4.10	13.96
C9	5.61	26.16	31.77	8.08	3.21	4.87	12.17	20.26	4.12	3.25	0.87	3.43	7.54
C10	4.96	26.35	31.31	8.29	3.68	4.61	12.34	20.63	12.11	11.29	0.82	4.09	16.20
C12	5.30	26.15	31.45	8.13	3.33	4.80	12.35	20.48	4.03	3.19	0.84	4.09	8.12
C13_1	5.36	26.06	31.43	8.60	3.95	4.66	12.53	21.13	4.87	3.54	1.34	4.91	9.78
C13_2	5.37	26.09	31.46	8.62	3.98	4.64	12.51	21.13	4.86	3.53	1.33	4.90	9.76
C14	85.86	25.91	111.77	7.74	3.14	4.60	12.08	19.82	4.10	3.27	0.83	3.41	7.51
C15	4.60	26.37	30.97	8.25	3.64	4.61	12.25	20.50	4.14	3.34	0.80	4.20	8.34
C16	10.15	26.56	36.71	11.88	3.44	8.43	12.34	24.22	14.02	12.06	1.96	7.42	21.44
C17				13.48	10.33	3.15	12.05	25.53	4.56	3.75	0.81	3.41	7.97

Table D.152 Insert time of 40 MB. data size, 160 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.29	26.11	31.41	9.64	4.91	4.73	12.41	22.05	8.13	7.29	0.83	3.93	12.06
C3	4.98	26.20	31.18	10.40	5.56	4.84	12.46	22.86	8.15	7.36	0.78	3.92	12.07
C4	4.96	25.98	30.94	8.11	3.38	4.73	12.36	20.47	3.93	3.11	0.82	4.03	7.96
C5	4.39	25.84	30.23	9.48	4.76	4.72	12.19	21.67	6.01	5.19	0.83	3.28	9.29
C6	6.21	26.07	32.28	8.24	3.47	4.76	12.16	20.39	11.79	10.96	0.84	3.30	15.09
C7	5.56	26.10	31.67	12.74	8.01	4.73	12.32	25.06	9.12	8.34	0.78	4.05	13.17
C8	5.46	26.16	31.62	8.55	3.77	4.77	12.43	20.97	8.59	7.79	0.80	3.95	12.53
C9	5.59	26.31	31.90	8.01	3.11	4.90	12.17	20.18	3.92	3.13	0.80	3.29	7.22
C10	4.97	26.16	31.13	8.44	3.71	4.74	12.49	20.93	9.99	9.20	0.80	3.92	13.91
C12	5.33	26.20	31.53	8.30	3.42	4.88	12.39	20.69	3.83	3.06	0.77	3.92	7.75
C13_1	5.41	25.95	31.37	8.88	4.14	4.73	12.52	21.40	4.72	3.42	1.30	4.90	9.62
C13_2	5.42	26.00	31.42	8.90	4.15	4.75	12.53	21.42	4.62	3.32	1.30	4.89	9.51
C14	151.09	25.91	177.00	8.12	3.36	4.76	12.14	20.27	3.89	3.10	0.79	3.31	7.20
C15	4.64	25.85	30.49	8.38	3.64	4.73	12.33	20.70	4.11	3.31	0.80	4.03	8.14
C16	10.16	26.25	36.42	12.72	3.58	9.14	12.45	25.17	11.32	9.53	1.80	7.16	18.49
C17				13.88	10.51	3.38	12.11	26.00	4.23	3.45	0.78	3.27	7.51

Table D.153 Insert time of 40 MB. data size, 320 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.30	26.25	31.56	10.92	5.95	4.97	12.49	23.41	8.39	7.63	0.76	3.64	12.03
C3	5.10	26.16	31.26	10.93	5.99	4.93	12.61	23.54	8.37	7.60	0.77	3.61	11.98
C4	5.08	26.07	31.14	8.56	3.50	5.06	12.38	20.94	3.82	3.05	0.76	3.72	7.53
C5	4.46	26.28	30.74	10.20	5.23	4.97	12.16	22.36	5.75	4.97	0.78	3.02	8.77
C6	6.30	26.02	32.32	8.60	3.65	4.96	12.12	20.72	11.26	10.49	0.77	3.01	14.27
C7	5.75	26.06	31.82	13.29	8.29	5.00	12.33	25.62	9.49	8.73	0.76	3.72	13.20
C8	5.46	26.10	31.56	9.21	4.24	4.97	12.51	21.72	7.84	7.08	0.76	3.64	11.48
C9	5.68	26.23	31.91	8.54	3.27	5.26	12.14	20.67	3.77	3.01	0.76	3.02	6.80
C10	5.08	26.26	31.35	9.23	4.21	5.02	12.59	21.82	9.85	9.08	0.77	3.62	13.46
C12	5.36	26.29	31.65	8.47	3.50	4.97	12.39	20.86	3.81	3.06	0.75	3.64	7.45
C13_1	5.53	25.91	31.45	9.04	4.05	4.99	12.52	21.56	4.47	3.28	1.19	3.52	7.99
C13_2	5.53	25.96	31.49	9.15	4.16	4.99	12.53	21.68	4.48	3.30	1.18	3.51	7.99
C14	286.38	25.99	312.37	8.80	3.85	4.95	12.11	20.91	3.76	3.00	0.77	3.03	6.80
C15	4.80	26.30	31.10	8.99	3.91	5.08	12.32	21.31	3.80	3.04	0.76	3.72	7.51
C16	10.31	26.09	36.40	13.33	3.72	9.61	12.42	25.75	10.82	9.16	1.66	6.56	17.38
C17				14.47	10.84	3.63	12.16	26.63	4.05	3.29	0.76	3.04	7.09

Table D.154 Insert time of 40 MB. data size, 640 record redundancy, cold cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	5.34	26.23	31.57	11.75	6.22	5.53	12.72	24.47	8.17	7.51	0.66	3.03	11.20
C3	5.21	26.25	31.46	11.84	6.21	5.63	12.86	24.70	8.20	7.55	0.65	3.01	11.21
C4	5.20	26.41	31.61	9.57	4.30	5.27	12.55	22.12	3.68	3.02	0.66	3.08	6.77
C5	4.53	26.18	30.71	11.57	5.88	5.69	12.25	23.82	5.15	4.48	0.67	2.53	7.68
C6	6.37	26.38	32.75	10.08	4.46	5.62	12.26	22.35	10.48	9.81	0.67	2.50	12.98
C7	5.80	26.49	32.29	15.73	10.25	5.48	12.56	28.29	9.12	8.46	0.66	3.10	12.22
C8	5.51	26.41	31.93	10.29	4.60	5.68	12.76	23.05	7.70	7.06	0.64	3.01	10.71
C9	5.74	26.48	32.22	9.41	3.86	5.55	12.29	21.70	3.57	2.91	0.66	2.52	6.09
C10	5.21	26.34	31.55	9.98	4.60	5.39	12.84	22.83	9.40	8.75	0.65	3.01	12.41
C12	5.47	26.51	31.98	9.68	4.29	5.40	12.47	22.15	3.66	3.02	0.64	3.01	6.68
C13_1	5.69	26.19	31.88	10.26	4.53	5.73	12.53	22.79	4.09	3.08	1.01	3.60	7.69
C13_2	5.68	26.39	32.07	10.62	4.89	5.73	12.53	23.15	4.25	3.23	1.01	3.59	7.83
C14	552.92	26.10	579.02	9.69	3.95	5.75	12.27	21.96	3.52	2.85	0.66	2.51	6.03
C15	4.90	26.49	31.40	9.87	4.47	5.40	12.55	22.42	3.61	2.95	0.65	3.09	6.70
C16	10.57	26.23	36.80	14.78	3.90	10.88	12.69	27.47	10.47	9.02	1.45	5.41	15.87
C17				17.51	13.62	3.89	12.25	29.77	3.90	3.24	0.66	2.50	6.41

XI) Detail-Table for data size 40 MB. in warm cache

Table D.155 Replace time of 40 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.55	26.11	30.66	6.05	1.46	4.58	12.52	18.57	3.56	2.69	0.87	4.15	7.70
C3	7.46	26.11	33.57	16.99	12.39	4.60	12.34	29.32	4.35	3.52	0.83	4.14	8.49
C4	16.47	26.11	42.58	5.19	0.62	4.56	12.44	17.63	1.47	0.60	0.86	4.26	5.73
C5	3.71	26.07	29.77	5.75	1.21	4.54	12.29	18.04	4.48	3.59	0.90	3.75	8.23
C6	5.49	25.77	31.26	5.22	0.69	4.53	12.18	17.40	2.44	1.55	0.88	3.76	6.19
C7	19.51	26.16	45.68	6.00	1.56	4.45	12.47	18.48	2.07	1.20	0.87	4.20	6.27
C8	4.73	26.35	31.08	5.81	1.24	4.57	12.55	18.36	1.77	0.94	0.82	4.15	5.91
C9	4.97	25.96	30.93	5.59	1.06	4.53	12.13	17.72	1.54	0.64	0.89	3.53	5.07
C10	7.48	25.98	33.47	5.80	1.16	4.64	12.50	18.30	2.00	1.18	0.82	4.13	6.13
C12	4.64	26.33	30.98	5.18	0.60	4.58	12.47	17.65	1.93	1.09	0.84	4.09	6.02
C14	84.54	26.09	110.63	5.60	1.12	4.48	12.18	17.77	1.46	0.61	0.85	3.41	4.87
C15	14.90	26.13	41.03	5.28	0.75	4.54	12.32	17.60	1.45	0.58	0.87	4.21	5.66
C16	9.34	26.03	35.37	9.84	1.56	8.28	12.36	22.20	3.40	1.43	1.98	7.43	10.84
C17				10.85	7.69	3.17	12.20	23.06	1.82	1.00	0.82	3.41	5.23

Table D.156 Replace time of 40 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.58	25.99	30.57	6.19	1.45	4.74	12.39	18.58	3.13	2.29	0.84	3.94	7.07
C3	7.48	25.89	33.37	28.32	23.48	4.84	12.44	40.76	3.82	2.99	0.83	3.94	7.76
C4	16.53	26.02	42.55	5.31	0.64	4.67	12.30	17.61	1.44	0.59	0.85	4.04	5.48
C5	3.79	26.01	29.80	6.17	1.50	4.67	12.10	18.27	3.91	3.08	0.82	3.28	7.19
C6	5.49	26.23	31.72	5.42	0.69	4.74	12.21	17.64	2.22	1.39	0.84	3.28	5.50
C7	19.54	26.09	45.63	6.38	1.65	4.74	12.29	18.67	2.02	1.16	0.86	4.04	6.05
C8	4.76	26.02	30.79	6.13	1.34	4.78	12.38	18.50	1.75	0.93	0.82	3.93	5.67
C9	5.02	26.10	31.12	6.00	1.16	4.84	12.11	18.11	1.49	0.62	0.86	3.28	4.77
C10	7.47	25.97	33.44	6.03	1.32	4.71	12.44	18.48	1.96	1.16	0.80	3.93	5.89
C12	4.64	26.52	31.16	5.41	0.62	4.79	12.46	17.87	1.69	0.91	0.78	3.94	5.63
C14	150.22	26.09	176.31	5.76	1.02	4.74	12.10	17.87	1.43	0.59	0.84	3.27	4.70
C15	14.97	26.36	41.33	5.47	0.77	4.69	12.29	17.75	1.42	0.58	0.84	4.05	5.46
C16	9.41	26.05	35.46	10.79	1.59	9.20	12.44	23.22	3.22	1.39	1.83	7.17	10.39
C17				11.63	8.25	3.38	12.10	23.73	1.66	0.86	0.80	3.28	4.94

Table D.157 Replace time of 40 MB. data size, 320 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.68	26.22	30.89	6.75	1.73	5.02	12.52	19.27	2.92	2.15	0.77	3.64	6.56
C3	7.75	26.45	34.20	50.64	45.64	5.01	12.62	63.26	3.50	2.73	0.76	3.63	7.13
C4	16.97	26.38	43.35	5.75	0.69	5.06	12.35	18.09	1.37	0.59	0.78	3.75	5.12
C5	3.80	26.30	30.10	6.80	1.85	4.95	12.17	18.97	3.76	2.99	0.77	3.08	6.84
C6	5.63	26.27	31.90	5.77	0.82	4.95	12.15	17.92	2.02	1.24	0.78	3.02	5.04
C7	19.98	26.00	45.98	7.09	2.07	5.02	12.35	19.44	1.94	1.18	0.77	3.73	5.67
C8	4.81	26.15	30.96	6.41	1.42	5.00	12.49	18.90	1.66	0.89	0.76	3.68	5.34
C9	5.05	26.09	31.14	6.42	1.46	4.96	12.13	18.55	1.37	0.61	0.76	3.08	4.46
C10	7.69	26.02	33.71	6.36	1.41	4.95	12.59	18.95	1.89	1.12	0.77	3.65	5.54
C12	4.75	26.67	31.42	5.68	0.68	5.00	12.41	18.09	1.64	0.88	0.76	3.64	5.28
C14	287.05	25.94	312.99	6.07	1.06	5.01	12.12	18.19	1.38	0.60	0.78	3.02	4.40
C15	15.39	26.20	41.59	5.90	0.86	5.04	12.39	18.29	1.36	0.58	0.77	3.73	5.09
C16	9.67	26.10	35.77	11.22	1.77	9.45	12.48	23.70	3.12	1.34	1.78	6.62	9.74
C17				12.15	8.57	3.58	12.15	24.30	1.61	0.84	0.76	3.02	4.63

Table D.158 Replace time of 40 MB. data size, 640 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.75	26.42	31.17	7.17	1.80	5.37	12.65	19.82	2.67	2.01	0.65	3.00	5.67
C3	7.78	26.12	33.90	94.40	89.04	5.36	12.85	###	2.86	2.20	0.66	2.99	5.85
C4	17.19	26.29	43.48	6.08	0.77	5.31	12.35	18.43	1.23	0.58	0.65	3.09	4.32
C5	3.84	26.53	30.37	7.40	1.92	5.48	12.22	19.61	3.03	2.37	0.66	2.51	5.55
C6	5.66	26.33	31.99	6.39	1.00	5.39	12.25	18.64	1.86	1.18	0.68	2.50	4.36
C7	20.20	26.44	46.64	7.56	2.27	5.29	12.35	19.91	1.65	0.99	0.66	3.09	4.74
C8	4.85	26.29	31.14	6.98	1.46	5.52	12.69	19.67	1.45	0.80	0.65	3.01	4.46
C9	5.16	26.81	31.97	6.87	1.52	5.35	12.30	19.17	1.26	0.60	0.66	2.52	3.78
C10	7.78	26.27	34.05	6.73	1.44	5.29	12.84	19.57	1.67	1.01	0.66	3.01	4.67
C12	4.80	26.94	31.74	6.52	1.11	5.41	12.47	18.98	1.47	0.82	0.64	3.03	4.50
C14	548.67	26.13	574.80	7.40	1.81	5.59	12.28	19.67	1.26	0.58	0.67	2.61	3.87
C15	15.71	26.54	42.25	6.74	1.47	5.27	12.57	19.31	1.22	0.57	0.66	3.09	4.32
C16	9.71	25.94	35.65	12.72	2.00	10.72	12.65	25.38	2.74	1.17	1.57	5.41	8.15
C17				14.32	10.53	3.78	12.27	26.58	1.50	0.83	0.67	2.50	4.00

Table D.159 Delete time of 40 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.59	26.35	30.95	6.00	1.41	4.60	12.32	18.32	3.91	3.01	0.90	4.09	8.00
C3	4.27	26.09	30.36	17.09	12.50	4.59	12.32	29.41	4.66	3.77	0.89	4.12	8.78
C4	4.16	26.67	30.83	5.23	0.65	4.57	12.46	17.68	1.47	0.64	0.84	4.22	5.70
C5	3.71	26.52	30.23	5.83	1.28	4.55	12.20	18.03	4.65	3.75	0.90	3.45	8.10
C6	5.56	26.35	31.92	5.33	0.71	4.62	12.19	17.52	2.49	1.59	0.90	3.43	5.91
C7	4.78	26.32	31.10	6.34	1.75	4.59	12.51	18.85	2.06	1.25	0.81	4.21	6.27
C8	4.68	26.23	30.91	5.94	1.26	4.69	12.49	18.44	1.78	0.98	0.80	4.13	5.91
C9	4.92	26.17	31.09	5.76	1.11	4.65	12.30	18.06	1.55	0.73	0.83	3.44	4.99
C10	4.26	26.21	30.47	5.90	1.22	4.68	12.43	18.34	2.30	1.50	0.80	4.11	6.41
C12	4.66	26.74	31.39	5.15	0.61	4.54	12.50	17.65	1.87	1.05	0.82	4.13	5.99
C14	86.13	26.30	112.43	5.61	1.02	4.59	12.24	17.85	1.46	0.63	0.83	3.43	4.88
C15	3.85	26.52	30.37	5.37	0.82	4.56	12.43	17.80	1.41	0.61	0.80	4.20	5.61
C16	9.42	26.32	35.75	7.37	1.55	5.82	12.41	19.78	3.40	1.43	1.96	7.46	10.86
C17				10.89	7.72	3.17	12.06	22.95	1.89	1.06	0.82	3.41	5.30

Table D.160 Delete time of 40 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.57	26.16	30.73	6.15	1.40	4.75	12.39	18.53	3.15	2.31	0.84	3.92	7.07
C3	4.27	26.21	30.48	28.84	24.11	4.73	12.35	41.20	3.76	2.94	0.82	3.93	7.70
C4	4.17	26.39	30.56	5.51	0.69	4.82	12.34	17.85	1.38	0.62	0.76	4.06	5.44
C5	3.71	26.07	29.79	6.19	1.42	4.77	12.17	18.37	4.04	3.21	0.84	3.30	7.34
C6	5.54	26.60	32.14	5.40	0.69	4.71	12.15	17.55	2.22	1.39	0.83	3.29	5.50
C7	4.77	26.12	30.89	6.63	1.83	4.81	12.35	18.98	1.99	1.19	0.80	4.11	6.10
C8	4.70	26.21	30.91	6.06	1.27	4.79	12.35	18.41	1.72	0.95	0.77	3.93	5.66
C9	4.98	26.36	31.34	6.33	1.50	4.82	12.16	18.49	1.43	0.62	0.81	3.34	4.77
C10	4.29	26.15	30.44	6.10	1.27	4.83	12.34	18.44	1.94	1.18	0.77	3.94	5.88
C12	4.66	26.63	31.29	5.40	0.63	4.77	12.38	17.78	1.67	0.91	0.76	3.97	5.64
C14	151.63	26.17	177.79	5.81	1.05	4.77	12.11	17.92	1.42	0.59	0.82	3.36	4.77
C15	3.87	26.24	30.11	5.64	0.87	4.77	12.39	18.03	1.38	0.60	0.78	4.04	5.42
C16	9.44	26.38	35.82	10.77	1.58	9.19	12.39	23.16	3.21	1.41	1.81	7.25	10.46
C17				11.50	8.21	3.29	12.16	23.66	1.68	0.88	0.80	3.29	4.97

Table D.161 Delete time of 40 MB. data size, 320 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.64	26.09	30.73	7.36	2.40	4.97	12.31	19.68	2.95	2.19	0.76	3.75	6.70
C3	4.36	25.95	30.32	50.81	45.86	4.95	12.28	63.10	3.69	2.93	0.76	3.65	7.34
C4	4.22	26.26	30.48	5.70	0.72	4.98	12.33	18.03	1.37	0.61	0.75	3.90	5.26
C5	3.77	26.07	29.84	6.66	1.63	5.03	12.23	18.89	3.69	2.91	0.78	3.02	6.71
C6	5.60	26.08	31.68	5.90	0.84	5.07	12.10	18.00	2.03	1.25	0.78	3.03	5.06
C7	4.86	25.97	30.83	7.22	2.22	5.00	12.51	19.73	1.91	1.14	0.77	3.75	5.67
C8	4.75	26.32	31.08	6.50	1.54	4.96	12.32	18.82	1.68	0.93	0.75	3.63	5.31
C9	5.03	26.39	31.42	6.15	1.07	5.08	12.24	18.40	1.42	0.64	0.78	3.08	4.49
C10	4.35	26.20	30.55	6.48	1.51	4.97	12.28	18.76	1.90	1.13	0.77	3.65	5.54
C12	4.69	26.37	31.06	5.71	0.77	4.94	12.37	18.08	1.64	0.89	0.75	3.64	5.28
C14	287.46	25.99	313.45	6.07	1.15	4.92	12.09	18.16	1.36	0.59	0.77	3.02	4.38
C15	3.94	26.16	30.10	5.88	0.91	4.97	12.39	18.27	1.35	0.60	0.75	3.72	5.07
C16	9.57	26.15	35.72	11.21	1.78	9.44	12.29	23.50	2.92	1.36	1.56	6.55	9.47
C17				11.94	8.34	3.59	12.14	24.08	1.63	0.85	0.78	3.04	4.67

Table D.162 Delete time of 40 MB. data size, 640 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.66	26.33	31.00	7.64	2.26	5.38	12.36	20.00	2.69	2.05	0.64	3.01	5.70
C3	4.46	26.59	31.05	94.02	88.72	5.30	12.28	###	3.56	2.90	0.65	3.00	6.56
C4	4.26	26.46	30.72	6.18	0.86	5.32	12.43	18.60	1.26	0.62	0.64	3.11	4.37
C5	3.85	26.69	30.54	7.21	1.74	5.47	12.15	19.36	3.00	2.33	0.67	2.54	5.54
C6	5.68	26.34	32.03	6.36	0.98	5.38	12.21	18.56	1.83	1.15	0.67	2.52	4.35
C7	4.91	26.39	31.30	7.97	2.66	5.31	12.47	20.44	1.68	1.03	0.65	3.09	4.77
C8	4.78	26.62	31.40	7.23	1.72	5.51	12.36	19.59	1.49	0.85	0.64	3.04	4.53
C9	5.06	26.47	31.53	6.72	1.25	5.47	12.19	18.91	1.32	0.65	0.67	2.51	3.83
C10	4.37	26.24	30.61	7.09	1.71	5.38	12.28	19.38	1.63	0.98	0.65	3.02	4.65
C12	4.76	26.73	31.49	6.17	0.78	5.38	12.46	18.63	1.53	0.90	0.64	3.01	4.55
C14	549.22	26.12	575.33	6.81	1.25	5.56	12.19	19.01	1.25	0.58	0.67	2.51	3.76
C15	3.98	26.38	30.36	6.43	1.15	5.28	12.49	18.92	1.27	0.62	0.65	3.11	4.38
C16	9.59	26.10	35.70	12.70	2.00	10.70	12.25	24.96	2.51	1.17	1.34	5.42	7.93
C17				13.04	9.22	3.81	12.21	25.25	1.51	0.84	0.67	2.53	4.04

Table D.163 Insert time of 40 MB. data size, 80 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.55	26.29	30.84	6.01	1.42	4.59	12.38	18.39	3.31	2.45	0.87	4.10	7.41
C3	4.34	26.14	30.47	6.51	1.92	4.59	12.40	18.91	4.08	3.25	0.83	4.10	8.18
C4	4.21	26.29	30.50	5.24	0.69	4.55	12.32	17.56	1.51	0.64	0.87	4.23	5.73
C5	3.69	26.21	29.90	5.80	1.27	4.53	12.21	18.00	4.79	3.90	0.90	3.44	8.23
C6	5.59	26.34	31.93	5.24	0.67	4.57	12.23	17.47	2.41	1.52	0.89	3.43	5.84
C7	4.88	26.25	31.14	6.33	1.86	4.47	12.48	18.81	2.12	1.25	0.87	4.22	6.34
C8	4.73	26.33	31.06	5.97	1.32	4.65	12.53	18.50	1.80	0.98	0.82	4.12	5.92
C9	4.95	26.27	31.22	5.94	1.39	4.56	12.34	18.29	1.56	0.68	0.88	3.45	5.01
C10	4.31	26.45	30.76	6.02	1.36	4.67	12.67	18.70	2.14	1.31	0.82	4.11	6.25
C12	4.67	26.25	30.92	5.18	0.61	4.57	12.44	17.62	1.88	1.04	0.84	4.09	5.97
C13_1	4.82	26.14	30.96	5.15	0.69	4.46	12.48	17.63	2.51	1.16	1.34	4.95	7.46
C13_2	4.83	26.19	31.01	5.17	0.68	4.48	12.48	17.65	2.52	1.18	1.34	4.93	7.45
C14	86.02	26.01	112.02	5.63	1.05	4.59	12.23	17.86	1.50	0.62	0.88	3.41	4.91
C15	3.92	26.48	30.40	5.28	0.74	4.54	12.41	17.69	1.48	0.61	0.86	4.20	5.67
C16	9.46	26.66	36.12	9.57	1.30	8.27	12.38	21.96	3.38	1.43	1.95	7.47	10.85
C17				10.92	7.76	3.15	12.08	23.00	1.90	1.09	0.81	3.41	5.31

Table D.164 Insert time of 40 MB. data size, 160 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.59	26.21	30.80	6.40	1.64	4.76	12.40	18.80	3.14	2.31	0.83	3.92	7.07
C3	4.37	26.26	30.63	6.78	2.04	4.74	12.49	19.27	3.41	2.61	0.80	3.91	7.32
C4	4.32	26.06	30.38	5.40	0.67	4.73	12.33	17.72	1.46	0.61	0.85	4.05	5.51
C5	3.75	25.85	29.60	6.19	1.43	4.77	12.18	18.37	4.15	3.32	0.83	3.30	7.45
C6	5.59	26.14	31.73	5.40	0.68	4.72	12.16	17.56	2.12	1.29	0.83	3.29	5.41
C7	4.94	26.15	31.09	6.77	2.00	4.77	12.30	19.07	2.02	1.18	0.83	4.06	6.07
C8	4.76	26.31	31.08	6.50	1.74	4.76	12.44	18.94	1.76	0.97	0.80	3.93	5.69
C9	4.97	26.42	31.40	5.87	1.06	4.81	12.17	18.04	1.46	0.61	0.85	3.31	4.77
C10	4.35	26.26	30.61	6.26	1.45	4.81	12.49	18.75	2.01	1.21	0.80	3.92	5.93
C12	4.69	26.31	31.00	5.37	0.62	4.75	12.39	17.76	1.68	0.91	0.77	3.93	5.61
C13_1	4.87	26.13	31.01	5.25	0.69	4.56	12.50	17.75	2.26	0.95	1.30	4.91	7.16
C13_2	4.87	26.12	30.99	5.24	0.70	4.54	12.49	17.73	2.25	0.94	1.31	4.91	7.16
C14	151.36	25.94	177.30	5.95	1.22	4.73	12.23	18.18	1.45	0.59	0.86	3.28	4.73
C15	4.01	26.01	30.01	5.53	0.81	4.72	12.38	17.91	1.43	0.59	0.83	4.03	5.45
C16	9.45	26.42	35.87	10.70	1.52	9.18	12.44	23.14	3.22	1.41	1.81	7.17	10.39
C17				11.59	8.35	3.24	12.16	23.75	1.66	0.89	0.77	3.29	4.95

Table D.165 Insert time of 40 MB. data size, 320 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.65	26.30	30.95	7.17	2.20	4.97	12.51	19.68	2.98	2.21	0.76	3.72	6.69
C3	4.45	26.15	30.60	7.42	2.45	4.97	12.93	20.36	3.20	2.43	0.77	3.64	6.84
C4	4.41	26.24	30.66	5.73	0.74	4.98	12.36	18.09	1.37	0.61	0.76	3.91	5.28
C5	3.86	26.41	30.27	6.77	1.76	5.01	12.13	18.90	1.97	1.21	0.76	3.03	5.00
C6	5.64	26.06	31.70	5.90	0.84	5.07	12.11	18.01	8.85	8.08	0.77	3.03	11.89
C7	5.09	26.26	31.35	7.25	2.20	5.05	12.35	19.60	1.92	1.16	0.76	3.85	5.77
C8	4.80	26.17	30.97	6.70	1.78	4.92	12.49	19.19	1.68	0.93	0.76	3.63	5.32
C9	5.00	26.39	31.40	6.13	1.15	4.98	12.10	18.23	1.36	0.60	0.76	3.08	4.44
C10	4.45	26.20	30.65	6.99	1.88	5.11	12.57	19.56	1.91	1.14	0.77	3.62	5.53
C12	4.73	26.28	31.01	5.73	0.76	4.98	12.37	18.10	1.61	0.86	0.75	3.62	5.23
C13_1	4.91	26.07	30.99	5.38	0.76	4.62	12.49	17.87	2.13	0.92	1.21	3.53	5.66
C13_2	4.91	26.09	31.00	5.38	0.75	4.63	12.49	17.87	2.11	0.91	1.21	3.52	5.63
C14	287.67	26.15	313.82	6.13	1.22	4.91	12.09	18.22	1.35	0.59	0.76	3.02	4.37
C15	4.18	26.43	30.61	5.90	0.90	5.00	12.35	18.25	1.35	0.59	0.76	3.71	5.06
C16	9.70	26.18	35.88	11.06	1.61	9.45	12.43	23.49	3.05	1.32	1.73	6.54	9.59
C17				12.05	8.47	3.58	12.14	24.19	1.65	0.88	0.77	3.04	4.69

Table D.166 Insert time of 40 MB. data size, 640 record redundancy, warm cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.73	26.28	31.01	7.79	2.40	5.39	12.66	20.45	2.83	2.17	0.66	3.00	5.83
C3	4.55	26.51	31.06	7.57	2.23	5.34	12.80	20.37	2.97	2.31	0.65	3.00	5.96
C4	4.53	26.38	30.91	6.15	0.80	5.35	12.28	18.43	1.25	0.60	0.65	3.08	4.33
C5	3.89	26.15	30.04	7.28	1.83	5.45	12.16	19.44	3.09	2.42	0.67	2.53	5.62
C6	5.73	26.25	31.98	6.39	1.01	5.38	12.22	18.60	1.79	1.13	0.66	2.51	4.31
C7	5.14	26.35	31.49	7.67	2.27	5.40	12.34	20.01	1.69	1.03	0.66	3.10	4.79
C8	4.84	26.35	31.19	7.35	1.90	5.46	12.72	20.07	1.49	0.85	0.64	3.01	4.51
C9	5.08	26.48	31.56	6.61	1.28	5.33	12.24	18.85	1.24	0.58	0.66	2.52	3.76
C10	4.56	26.37	30.93	7.24	1.96	5.28	12.81	20.05	1.53	0.88	0.65	3.02	4.54
C12	4.82	26.52	31.34	6.21	0.78	5.43	12.44	18.66	1.47	0.82	0.64	3.02	4.48
C13_1	5.06	26.29	31.36	6.26	0.90	5.36	12.49	18.75	1.87	0.87	1.00	3.60	5.47
C13_2	5.05	26.36	31.41	6.29	0.90	5.39	12.49	18.78	1.89	0.88	1.00	3.58	5.46
C14	550.02	26.31	576.34	6.86	1.30	5.56	12.24	19.10	1.25	0.58	0.66	2.51	3.76
C15	4.26	26.62	30.88	6.50	1.19	5.31	12.50	19.00	1.23	0.58	0.65	3.08	4.31
C16	9.76	26.40	36.16	12.64	1.93	10.71	12.69	25.34	2.71	1.17	1.54	5.40	8.11
C17				14.46	10.64	3.82	12.21	26.67	1.50	0.83	0.67	2.53	4.03

XII) Detail-Table for data size 40 MB. in hot cache

Table D.167 Replace time of 40 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.59	26.27	30.85	5.92	1.34	4.58	12.34	18.26	1.79	0.91	0.87	4.10	5.89
C3	7.51	26.18	33.69	12.43	7.84	4.59	12.37	24.80	2.00	1.19	0.81	4.11	6.11
C4	16.42	26.64	43.06	5.19	0.62	4.57	12.34	17.54	1.46	0.59	0.87	4.21	5.67
C5	3.57	26.45	30.01	5.88	1.34	4.54	12.15	18.03	2.22	1.33	0.90	3.42	5.64
C6	5.51	25.99	31.50	5.16	0.64	4.53	12.16	17.33	1.82	0.92	0.90	3.42	5.24
C7	19.55	26.45	46.00	5.45	0.98	4.47	12.34	17.79	2.04	1.18	0.86	4.20	6.23
C8	4.72	26.51	31.22	5.57	0.97	4.60	12.33	17.90	1.76	0.94	0.82	4.09	5.84
C9	5.03	26.18	31.21	5.16	0.65	4.51	12.16	17.33	1.51	0.62	0.88	3.42	4.92
C10	7.54	26.30	33.84	5.58	0.96	4.62	12.35	17.93	1.99	1.16	0.82	4.09	6.08
C12	4.69	26.50	31.19	5.18	0.60	4.58	12.34	17.52	1.39	0.59	0.80	4.09	5.47
C14	85.18	26.27	111.45	5.12	0.61	4.51	12.14	17.27	1.43	0.60	0.83	3.41	4.84
C15	14.96	26.26	41.22	5.16	0.63	4.53	12.36	17.53	1.44	0.58	0.86	4.21	5.65
C16	9.43	26.21	35.64	9.43	1.15	8.28	12.39	21.82	3.36	1.39	1.96	7.44	10.79
C17				9.68	6.51	3.17	12.11	21.79	1.42	0.60	0.82	3.42	4.83

Table D.168 Replace time of 40 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.61	26.15	30.76	6.11	1.44	4.67	12.36	18.47	1.73	0.90	0.83	3.99	5.72
C3	7.54	25.99	33.53	20.16	15.37	4.80	12.47	32.63	2.00	1.17	0.83	3.98	5.99
C4	16.60	26.20	42.81	5.38	0.65	4.73	12.37	17.75	1.41	0.59	0.82	4.06	5.47
C5	3.74	26.26	29.99	5.78	1.22	4.56	12.11	17.90	2.12	1.29	0.83	3.28	5.40
C6	5.54	26.45	32.00	5.32	0.66	4.66	12.09	17.42	1.74	0.91	0.83	3.30	5.03
C7	19.68	26.30	45.98	5.76	1.02	4.74	12.30	18.06	1.97	1.15	0.82	4.06	6.03
C8	4.76	26.22	30.98	5.78	1.00	4.78	12.36	18.15	1.71	0.92	0.79	3.98	5.69
C9	5.04	26.28	31.32	5.52	0.66	4.85	12.10	17.62	1.47	0.61	0.86	3.30	4.77
C10	7.59	26.14	33.73	5.67	1.00	4.67	12.47	18.14	1.94	1.14	0.79	3.93	5.86
C12	4.79	27.06	31.85	5.40	0.62	4.78	12.33	17.73	1.37	0.58	0.79	3.94	5.30
C14	150.82	26.15	176.96	5.43	0.64	4.79	12.16	17.59	1.43	0.59	0.83	3.34	4.76
C15	15.03	26.33	41.37	5.32	0.67	4.65	12.36	17.68	1.37	0.57	0.79	4.05	5.41
C16	9.58	26.22	35.79	10.25	1.22	9.03	12.50	22.75	3.18	1.36	1.82	7.18	10.36
C17				10.23	6.84	3.39	12.12	22.35	1.38	0.59	0.79	3.29	4.67

Table D.169 Replace time of 40 MB. data size, 320 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.66	26.31	30.96	6.79	1.75	5.04	12.47	19.26	1.66	0.89	0.76	3.64	5.30
C3	7.71	26.52	34.23	34.57	29.62	4.94	12.62	47.18	1.89	1.12	0.77	3.63	5.52
C4	16.80	26.48	43.28	5.75	0.69	5.05	12.37	18.12	1.36	0.58	0.77	3.73	5.08
C5	3.77	26.28	30.05	6.47	1.44	5.03	12.12	18.59	2.01	1.25	0.76	3.01	5.02
C6	5.59	26.32	31.92	5.69	0.73	4.96	12.11	17.80	1.67	0.88	0.79	3.02	4.69
C7	19.91	26.03	45.94	6.13	1.11	5.02	12.42	18.56	1.87	1.10	0.77	3.72	5.59
C8	4.79	26.50	31.29	5.98	1.06	4.92	12.48	18.46	1.66	0.89	0.76	3.63	5.28
C9	5.13	26.39	31.52	5.73	0.71	5.02	12.13	17.86	1.38	0.61	0.77	3.02	4.40
C10	7.66	26.09	33.75	6.11	1.05	5.06	12.59	18.71	1.85	1.08	0.77	3.63	5.47
C12	4.82	27.03	31.85	5.69	0.67	5.02	12.37	18.06	1.34	0.58	0.75	3.64	4.97
C14	286.19	26.03	312.22	5.67	0.69	4.99	12.10	17.78	1.35	0.59	0.76	3.02	4.37
C15	15.24	26.31	41.55	5.80	0.75	5.05	12.42	18.22	1.33	0.57	0.76	3.76	5.09
C16	9.61	26.07	35.68	10.62	1.28	9.34	12.46	23.08	2.97	1.30	1.67	6.55	9.52
C17				10.99	7.42	3.56	12.16	23.15	1.35	0.58	0.76	3.03	4.38

Table D.170 Replace time of 40 MB. data size, 640 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.69	26.64	31.33	7.35	1.97	5.39	12.65	20.00	1.45	0.80	0.65	3.02	4.48
C3	7.79	26.34	34.13	64.09	58.69	5.41	12.89	76.98	1.65	0.99	0.66	3.02	4.67
C4	16.99	26.54	43.53	5.91	0.78	5.13	12.37	18.28	1.23	0.58	0.65	3.10	4.33
C5	3.81	26.58	30.39	7.03	1.60	5.43	12.15	19.18	1.69	1.01	0.68	2.55	4.24
C6	5.66	26.59	32.24	6.34	0.85	5.48	12.24	18.58	1.50	0.82	0.68	2.54	4.03
C7	19.91	26.84	46.76	6.49	1.23	5.27	12.36	18.86	1.63	0.97	0.66	3.12	4.75
C8	4.88	26.52	31.40	6.84	1.16	5.68	12.70	19.54	1.46	0.81	0.65	3.02	4.49
C9	5.14	26.74	31.88	6.34	0.80	5.54	12.23	18.57	1.26	0.59	0.67	2.52	3.78
C10	7.70	26.40	34.10	6.46	1.16	5.30	12.85	19.31	1.64	0.98	0.66	3.01	4.65
C12	4.84	27.20	32.04	6.15	0.75	5.40	12.47	18.62	1.22	0.57	0.65	3.03	4.26
C14	546.83	26.30	573.14	6.30	0.77	5.53	12.28	18.58	1.26	0.58	0.67	2.51	3.77
C15	15.48	26.57	42.05	6.19	0.87	5.32	12.34	18.53	1.22	0.56	0.66	3.12	4.34
C16	9.64	26.51	36.15	11.99	1.32	10.67	12.68	24.67	2.58	1.13	1.45	5.41	7.99
C17				12.97	9.11	3.86	12.30	25.27	1.25	0.58	0.67	2.51	3.76

Table D.171 Delete time of 40 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.53	26.27	30.79	5.91	1.37	4.55	12.26	18.17	1.84	0.94	0.90	4.13	5.97
C3	4.25	26.05	30.30	12.38	7.81	4.57	12.26	24.64	2.00	1.20	0.80	4.09	6.09
C4	4.07	26.52	30.58	5.19	0.62	4.57	12.30	17.49	1.44	0.60	0.84	4.21	5.65
C5	3.69	26.51	30.20	5.74	1.16	4.57	12.15	17.89	2.21	1.32	0.90	3.41	5.62
C6	5.53	26.27	31.81	5.20	0.62	4.58	12.13	17.33	1.81	0.92	0.89	3.43	5.24
C7	4.77	26.22	30.99	5.69	0.97	4.72	12.34	18.03	1.97	1.18	0.79	4.20	6.17
C8	4.68	26.24	30.92	5.59	0.99	4.60	12.30	17.90	1.74	0.95	0.80	4.11	5.86
C9	4.86	26.12	30.99	5.20	0.64	4.56	12.13	17.33	1.46	0.62	0.83	3.42	4.87
C10	4.26	26.16	30.42	5.59	0.97	4.62	12.25	17.84	2.03	1.22	0.81	4.09	6.11
C12	4.63	26.70	31.33	5.16	0.60	4.57	12.36	17.52	1.41	0.59	0.82	4.10	5.51
C14	85.44	26.15	111.59	5.21	0.64	4.57	12.15	17.36	1.42	0.59	0.82	3.42	4.84
C15	3.83	26.30	30.13	5.16	0.65	4.51	12.37	17.53	1.39	0.58	0.80	4.20	5.59
C16	9.28	26.22	35.49	9.47	1.11	8.37	12.38	21.85	3.36	1.40	1.96	7.44	10.81
C17				9.75	6.52	3.23	12.12	21.87	1.41	0.59	0.82	3.42	4.83

Table D.172 Delete time of 40 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.53	26.07	30.60	6.24	1.50	4.73	12.36	18.60	1.78	0.93	0.85	3.95	5.73
C3	4.28	26.06	30.34	20.19	15.46	4.73	12.33	32.51	1.95	1.17	0.78	3.98	5.93
C4	4.15	26.28	30.43	5.44	0.65	4.79	12.29	17.72	1.38	0.60	0.78	4.04	5.42
C5	3.71	25.90	29.62	5.94	1.19	4.74	12.14	18.08	2.13	1.30	0.84	3.29	5.42
C6	5.54	26.35	31.89	5.36	0.68	4.68	12.12	17.49	1.74	0.91	0.82	3.29	5.03
C7	4.79	25.95	30.74	5.80	1.03	4.76	12.35	18.15	2.00	1.16	0.84	4.06	6.07
C8	4.71	26.15	30.86	5.77	1.01	4.77	12.32	18.10	1.71	0.93	0.78	3.97	5.68
C9	4.95	26.17	31.12	5.49	0.67	4.82	12.12	17.61	1.46	0.62	0.83	3.36	4.82
C10	4.31	26.04	30.35	5.86	1.02	4.83	12.36	18.22	2.03	1.18	0.85	3.96	5.99
C12	4.66	26.50	31.16	5.42	0.63	4.79	12.37	17.79	1.36	0.59	0.77	3.95	5.32
C14	150.78	25.97	176.74	5.36	0.64	4.72	12.11	17.47	1.42	0.59	0.83	3.29	4.72
C15	3.84	26.07	29.90	5.43	0.68	4.75	12.33	17.76	1.36	0.58	0.78	4.05	5.41
C16	9.37	26.20	35.57	10.39	1.20	9.19	12.37	22.76	3.18	1.38	1.81	7.17	10.35
C17				10.09	6.75	3.34	12.10	22.19	1.39	0.59	0.79	3.29	4.68

Table D.173 Delete time of 40 MB. data size, 320 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.61	25.96	30.57	6.98	2.05	4.93	12.30	19.27	1.66	0.90	0.76	3.65	5.31
C3	4.34	25.85	30.18	33.14	28.20	4.94	12.27	45.41	1.88	1.13	0.76	3.64	5.52
C4	4.19	26.15	30.34	5.67	0.70	4.97	12.33	18.00	1.35	0.59	0.76	3.72	5.07
C5	3.74	26.25	30.00	6.64	1.61	5.03	12.12	18.76	2.01	1.25	0.77	3.02	5.04
C6	5.58	25.97	31.55	5.75	0.72	5.03	12.11	17.87	1.66	0.88	0.79	3.02	4.69
C7	4.81	25.90	30.71	6.04	1.09	4.94	12.41	18.45	1.88	1.10	0.77	3.72	5.60
C8	4.72	25.87	30.59	6.01	1.06	4.95	12.32	18.33	1.65	0.90	0.75	3.64	5.29
C9	4.97	26.26	31.23	5.74	0.71	5.02	12.12	17.86	1.39	0.61	0.78	3.03	4.42
C10	4.33	26.15	30.48	6.00	1.06	4.95	12.23	18.23	1.92	1.14	0.78	3.64	5.55
C12	4.65	26.23	30.89	5.59	0.66	4.93	12.36	17.95	1.34	0.58	0.76	3.64	4.98
C14	287.40	25.83	313.23	5.49	0.67	4.82	12.12	17.62	1.35	0.58	0.77	3.04	4.39
C15	3.84	25.99	29.84	5.72	0.76	4.96	12.39	18.11	1.34	0.57	0.77	3.73	5.07
C16	9.49	26.03	35.52	10.70	1.27	9.43	12.29	22.99	2.98	1.30	1.68	6.56	9.54
C17				10.71	7.11	3.60	12.16	22.87	1.37	0.59	0.77	3.05	4.42

Table D.174 Delete time of 40 MB. data size, 640 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.62	26.14	30.77	7.38	2.11	5.28	12.42	19.81	1.45	0.80	0.65	3.01	4.47
C3	4.41	26.07	30.49	63.97	58.74	5.23	12.34	76.31	1.66	1.00	0.66	3.02	4.68
C4	4.22	26.18	30.39	6.13	0.86	5.27	12.57	18.70	1.23	0.58	0.65	3.11	4.34
C5	3.79	26.55	30.34	7.51	2.13	5.38	12.24	19.76	1.69	1.02	0.67	2.53	4.22
C6	5.64	26.35	32.00	6.19	0.87	5.32	12.26	18.45	1.51	0.82	0.69	2.51	4.03
C7	4.86	26.18	31.04	6.61	1.22	5.38	12.54	19.14	1.64	0.98	0.66	3.09	4.73
C8	4.75	26.25	31.00	6.70	1.19	5.51	12.44	19.14	1.46	0.81	0.65	3.02	4.48
C9	5.03	26.33	31.37	6.38	0.86	5.51	12.24	18.61	1.28	0.60	0.68	2.53	3.81
C10	4.36	26.05	30.41	6.52	1.19	5.33	12.37	18.89	1.64	0.99	0.65	3.02	4.66
C12	4.71	26.55	31.27	6.07	0.76	5.31	12.49	18.56	1.22	0.57	0.65	3.01	4.23
C14	548.18	25.97	574.15	6.26	0.74	5.52	12.25	18.51	1.25	0.59	0.66	2.53	3.78
C15	3.92	26.23	30.15	5.97	0.88	5.09	12.52	18.49	1.22	0.57	0.65	3.12	4.34
C16	9.60	25.99	35.59	12.01	1.31	10.69	12.35	24.35	2.47	1.12	1.35	5.42	7.90
C17				11.85	8.01	3.84	12.26	24.11	1.26	0.59	0.67	2.51	3.78

Table D.175 Insert time of 40 MB. data size, 80 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.57	26.16	30.73	5.93	1.37	4.57	12.33	18.26	1.81	0.94	0.87	4.10	5.92
C3	4.33	26.05	30.38	5.84	1.27	4.57	12.35	18.19	1.82	0.95	0.86	4.10	5.91
C4	4.19	26.22	30.41	5.15	0.61	4.54	12.36	17.51	1.47	0.59	0.88	4.21	5.67
C5	3.75	26.26	30.01	5.72	1.20	4.52	12.14	17.87	2.26	1.36	0.90	3.40	5.66
C6	5.62	26.31	31.93	5.19	0.62	4.57	12.13	17.31	1.82	0.92	0.89	3.42	5.24
C7	4.85	26.20	31.05	5.58	0.97	4.61	12.33	17.91	2.06	1.18	0.87	4.21	6.27
C8	4.72	26.22	30.94	5.42	0.98	4.44	12.33	17.75	1.76	0.93	0.82	4.10	5.86
C9	4.94	26.17	31.11	5.17	0.64	4.53	12.20	17.37	1.51	0.62	0.89	3.42	4.94
C10	4.32	26.26	30.58	5.59	0.99	4.60	12.35	17.94	2.03	1.21	0.82	4.10	6.12
C12	4.65	26.23	30.88	4.95	0.60	4.35	12.34	17.29	1.38	0.58	0.79	4.08	5.46
C13_1	4.75	26.05	30.79	5.34	0.68	4.66	12.77	18.11	1.92	0.57	1.34	4.96	6.88
C13_2	4.75	26.03	30.78	5.32	0.65	4.67	12.78	18.10	1.91	0.58	1.33	4.96	6.87
C14	85.32	25.92	111.24	5.21	0.62	4.59	12.21	17.42	1.48	0.59	0.88	3.42	4.90
C15	3.94	26.28	30.22	5.14	0.63	4.50	12.37	17.51	1.44	0.58	0.86	4.21	5.65
C16	9.37	26.63	36.00	9.26	1.01	8.25	12.41	21.66	3.34	1.40	1.95	7.44	10.78
C17				9.74	6.52	3.22	12.11	21.85	1.41	0.59	0.81	3.41	4.82

Table D.176 Insert time of 40 MB. data size, 160 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.58	26.05	30.62	6.26	1.53	4.74	12.41	18.67	1.75	0.92	0.83	3.95	5.70
C3	4.34	26.31	30.65	6.03	1.31	4.72	12.47	18.50	1.78	0.95	0.83	3.94	5.72
C4	4.29	25.98	30.27	5.36	0.65	4.71	12.33	17.69	1.42	0.59	0.83	4.05	5.47
C5	3.78	25.74	29.52	5.98	1.24	4.74	12.17	18.15	2.12	1.29	0.83	3.28	5.40
C6	5.54	26.06	31.61	5.32	0.66	4.66	12.12	17.44	1.75	0.92	0.83	3.29	5.04
C7	4.88	26.07	30.94	5.74	1.03	4.71	12.32	18.06	1.97	1.15	0.82	4.05	6.02
C8	4.74	26.17	30.91	5.77	1.01	4.76	12.42	18.18	1.75	0.93	0.82	3.96	5.71
C9	4.95	26.34	31.30	5.44	0.67	4.77	12.14	17.58	1.47	0.62	0.85	3.38	4.85
C10	4.34	26.02	30.36	5.79	1.00	4.79	12.48	18.27	1.98	1.17	0.81	3.96	5.93
C12	4.66	26.20	30.85	5.39	0.63	4.76	12.35	17.74	1.36	0.58	0.78	3.96	5.31
C13_1	4.76	25.98	30.74	5.18	0.68	4.50	12.46	17.64	1.90	0.58	1.32	4.97	6.87
C13_2	4.77	25.99	30.76	5.12	0.66	4.47	12.42	17.54	1.89	0.57	1.32	4.92	6.81
C14	150.74	25.90	176.64	2.20	0.64	1.56	12.14	14.35	1.44	0.59	0.85	3.28	4.73
C15	3.96	25.82	29.78	5.32	0.67	4.65	12.31	17.63	1.40	0.57	0.83	4.05	5.44
C16	9.35	26.25	35.60	10.41	1.12	9.29	12.45	22.86	3.16	1.36	1.80	7.14	10.30
C17				9.98	6.76	3.22	12.10	22.07	1.39	0.59	0.80	3.28	4.66

Table D.177 Insert time of 40 MB. data size, 320 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.61	26.14	30.75	6.99	2.09	4.90	12.49	19.48	1.66	0.89	0.76	3.64	5.29
C3	4.41	26.01	30.42	6.87	1.99	4.88	12.60	19.47	1.68	0.91	0.77	3.64	5.31
C4	4.41	26.07	30.48	5.63	0.69	4.94	12.35	17.98	1.35	0.59	0.76	3.74	5.09
C5	3.83	26.31	30.14	6.57	1.59	4.98	12.14	18.71	2.04	1.27	0.77	3.01	5.05
C6	5.62	26.00	31.62	5.77	0.73	5.05	12.14	17.91	1.66	0.88	0.78	3.03	4.69
C7	5.01	26.04	31.05	6.17	1.08	5.09	12.42	18.59	1.88	1.11	0.77	3.73	5.61
C8	4.76	26.05	30.81	6.04	1.06	4.98	12.50	18.53	1.66	0.90	0.76	3.64	5.30
C9	4.98	26.17	31.16	5.69	0.71	4.98	12.11	17.80	1.39	0.61	0.77	3.05	4.43
C10	4.42	26.06	30.48	6.21	1.07	5.15	12.58	18.79	1.84	1.07	0.77	3.62	5.47
C12	4.70	26.23	30.93	5.61	0.66	4.95	12.36	17.97	1.34	0.58	0.75	3.62	4.96
C13_1	4.89	25.90	30.79	5.33	0.70	4.62	12.62	17.94	1.79	0.57	1.22	3.55	5.33
C13_2	4.87	25.95	30.82	5.36	0.69	4.67	12.61	17.97	1.78	0.57	1.21	3.55	5.33
C14	285.99	25.99	311.99	5.62	0.68	4.94	12.11	17.73	1.36	0.59	0.77	3.04	4.39
C15	4.14	26.25	30.38	5.68	0.75	4.93	12.42	18.10	1.33	0.57	0.76	3.73	5.06
C16	9.59	26.07	35.66	10.57	1.21	9.36	12.45	23.01	2.97	1.30	1.67	6.56	9.53
C17				10.66	7.09	3.56	12.17	22.82	1.35	0.58	0.77	3.06	4.41

Table D.178 Insert time of 40 MB. data size, 640 record redundancy, hot cache

Cmd.	nxd			sxd					lxd				
	update	serial	total	nett	sql	update	serial	total	nett	sql	update	serial	total
C1	4.67	26.09	30.76	7.47	2.16	5.31	12.72	20.19	1.45	0.80	0.65	3.01	4.47
C3	4.56	26.28	30.83	7.48	2.24	5.24	12.88	20.37	1.48	0.82	0.65	3.01	4.49
C4	4.44	26.18	30.61	6.09	0.79	5.30	12.55	18.64	1.23	0.58	0.65	3.09	4.32
C5	3.91	26.03	29.94	7.02	1.63	5.39	12.23	19.25	1.68	1.01	0.67	2.52	4.20
C6	5.68	26.12	31.80	6.39	0.87	5.52	12.29	18.68	1.48	0.82	0.66	2.51	3.99
C7	5.14	26.25	31.39	6.58	1.29	5.29	12.56	19.14	1.64	0.98	0.66	3.10	4.74
C8	4.79	26.34	31.14	6.54	1.19	5.34	12.77	19.31	1.46	0.81	0.64	3.02	4.48
C9	5.03	26.34	31.38	6.27	0.81	5.46	12.31	18.58	1.27	0.60	0.67	2.53	3.80
C10	4.52	26.23	30.75	6.35	1.18	5.17	12.82	19.18	1.70	1.04	0.66	3.00	4.70
C12	4.79	26.44	31.23	6.10	0.76	5.33	12.47	18.57	1.20	0.56	0.64	3.01	4.21
C13_1	5.00	26.32	31.32	6.04	0.73	5.31	12.49	18.53	1.57	0.56	1.01	3.60	5.17
C13_2	4.98	26.33	31.31	6.11	0.75	5.36	12.47	18.58	1.57	0.56	1.01	3.60	5.17
C14	546.56	26.19	572.75	6.15	0.75	5.41	12.27	18.42	1.25	0.58	0.67	2.53	3.78
C15	4.20	26.39	30.59	6.54	1.26	5.28	12.58	19.12	1.22	0.57	0.65	3.09	4.31
C16	9.71	26.22	35.93	11.96	1.28	10.68	12.76	24.72	2.57	1.12	1.45	5.41	7.97
C17				11.53	7.77	3.76	12.26	23.79	1.24	0.57	0.66	2.52	3.75

Bibliography

[1] Abiteboul, S. *Querying Semi-Structured Data.* in *Database Theory - ICDT '97, 6th International Conference.* 1997. Delphi, Greece. p. 1-18.

[2] Abiteboul, S., S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Simeon, *Querying documents in object databases.* International Journal on Digital Libraries, 1997. **1**(1): p. 5-19.

[3] Abiteboul, S., S. Cluet, and T. Milo. *A Database Interface for Files Update.* in *Proceedings of the 1995 ACM SIGMOD international conference on Management of data.* 1995. San Jose, California, United States. p. 386-397.

[4] Abiteboul, S., D. Quass, J. McHuge, J. Widom, and J.L. Winer, *The Lorel query language for semistructured data.* International Journal on Digital Libraries, 1997. **1**: p. 68-88.

[5] Ali, M.A., F. Alvaro, and P. Norman, *MOVIE: An incremental maintenance system for materialized object views.* Data & Knowledge Engineering, 2003. **47**: p. 131-166.

[6] Alpdemir, M.N., A. Mukherjee, N.W. Paton, P. Watson, A.A.A. Fernandes, A. Gounaris, and J. Smith. *Service-Based Distributed Querying on the Grid.* in *Proceedings of the Service-Oriented Computing - ICSOC 2003.* 2003. Trento, Italy: Springer-Verlag. p. 467-482.

[7] Amer-Yahia, S., F. Du, and J. Freire. *A Comprehensive Solution to the XML-to-Relational Mapping Problem.* in *Sixth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2004).* 2004. Washington, DC, USA. p. 31-38.

[8] Apache, *Xindice XML database.* 2001.http://xml.apache.org/xindice/

[9] Apache, *Xalan-Java.* 2005.http://xml.apache.org/xalan-j/

[10] Apache, *Xerces.* 2005.http://xerces.apache.org/

[11] Arenas, M. and L. Libkin, *A Normal Form for XML Documents.* ACM Transactions on Database Systems (TODS), 2004. **29**(1): p. 195-232.

[12] Babcock, C., *Internet Insight: XML Users Consider Nonstandard.* 2002, appears in Ziff Davis' eWeek 11 Feb. 2002.http://www.charlesbabcock.com/xquery.htm

[13] Beeri, C. and Y. Tzaban. *SAL: An Algebra for Semistructured Data and XML.* in *ACM SIGMOD Workshop on The Web and Databases.* 1999. Philadelphia, Pennsylvania, USA. p. 37-42.

[14] Bitton, D., D.J. DeWitt, and C. Turbyfill. *Benchmarking Database Systems: A Systematic Approach.* in *Proceedings of the International Conference on Very Large Data Bases.* 1983. Florence, Italy. p. 8-19.

[15] Bohannon, P., J. Freire, P. Roy, and J. Simeon. *From XML schema to Relations: A Cost-Based Approach to XML Storage.* in *Proceedings of the International Conference on Data Engineering.* 2002. p. 64-75.

[16] Bohm, M. and J.-J. Dubray, *Dom4J performance versus Xerces / Xalan.* 2005.http://www.dom4j.org/benchmarks/xpath/index.html

[17] Bohme, T. and E. Rahm. *XMach-1: A Benchmark for XML Data Management.* in *Proceedings of the German Database Conference (BTW2001).* 2001. Oldenburg, Germany. p. 264-273.

[18] Bonifati, A. and S. Ceri, *Comparative Analysis of Five XML Query Languages.* ACM SIGMOD, 2000. **29**(1): p. 68-77.

[19] Bonifati, A. and D. Lee, *Technical Survey of XML Schema and Query Languages*, in. 2001, University of California at Los Angeles and Politecnico di Milano.

[20] Bourret, R., *XML and Databases.* 2003.http://www.rpbourret.com/xml/XMLAndDatabases.htm

[21] Bressan, S., M.L. Lee, Y.G. Li, and B. Wadhwa. *XOO7: Applying OO7 Benchmark to XML Query Processing Tools.* in *Proceedings of the ACM International Conference on Information and Knowledge Management.* 2001. Atlanta, Georgia. p. 167-174.

[22] Breuer, P. and J. Bowen. *The PRECC Compiler-Compiler.* in *Proc. UKUUG/SUKUG Joint New Year 1993 Conference.* 1993. UK. p. 167-182.

[23] Carey, M.J., D.J. DeWitt, M.J. Franklin, N.E. Hall, M.L. McAuliffe, J.F. Naughton, D.T. Schuh, M.H. Solomon, C.K. Tan, O.G. Tsatalos, S.J. White, and M.J. Zwilling. *Shoring up persistent applications.* in *Proc. of the ACM SIGMOD Int'l Conference on Management of Data.* 1994. Minneapolis, Minnesota. p. 383-394.

[24] Carey, M.J., D.J. DeWitt, and J.F. Naughton. *The OO7 Benchmark.* in *Proceedings of the ACM SIGMOD International Conference on Management of Data.* 1993. Washington D.C. p. 12-21.

[25] Carey, M.J., D.J. DeWitt, J.F. Naughton, M. Asgarian, P. Brown, J.E. Gehrke, and D.N. Shah. *The BUCKY Object-Relational Benchmark.* in *Proceedings of the ACM SIGMOD International Conference on Management of Data.* 1997. Tucson, Arizona. p. 135-146.

[26] Ceri, S., S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca, *XML-GL: a Graphical Language for Querying and Restructuring WWW Data.* The International Journal of Computer and Telecommunications Networking, 1999. **36**: p. 1171-1187.

[27] Chamberlin, D., *XQuery: An XML query language.* IBM Systems Journal, 2002. **41**(2002): p. 597-615.

[28] Chamberlin, D., *XQuery from experts: A Guide to the W3C XML Query Language.* 2003: Addison-Wesley.

[29] Chamberlin, D., *XQuery from the experts: Aguide to the W3C XML Query Language.* Influences on the Design of XQuery, p. 143, ed. H. Katz. 2003, XQuery from the experts: Aguide to the W3C XML Query Language: Addison-Wesley.

[30] Chamberlin, D.D., J. Robie, and D. Florescu:. *Quilt: An XML Query Language for Heterogeneous Data Sources.* in *Proceedings of the Third International Workshop on the Web and Databases, WebDB 2000.* 2000. Dallas, TX. p. 53-62.

[31] Chen, Y., S. Davidson, C. Hara, and Y. Zheng. *RRXS: Redundancy reducing XML storage in relations.* in *Proceedings of the 29th VLDB Conference.* 2003. Berlin, Germany. p. 189-200.

[32] Choi, B. *What Are Real DTDs Like.* in *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB).* 2002. Madison, Wisconsin, USA. p. 43-48.

[33] Christophides, V., S. Abiteboul, S. Cluet, and M. Scholl. *From structured documents to novel query facilities.* in *Proceedings of the ACM SIGMOD Conference.* 1994. Minneapolis, Minnesota. p. 313-324.

[34] Chung, T.-S. and H.-J. Kim, *A two phase optimization technique for XML queries with multiple regular expressions.* The Journal of Systems and Software, 2002. **64**: p. 183-193.

[35] Chung, T.-S. and H.-J. Kim, *Techniques for the evaluation of XML queries: a survey.* The Journal of Systems and Software, 2003. **46**: p. 225-246.

[36] Clark, J., *XML Transformations (XSLT) Version 1.0.* 1999.http://www.w3.org/TR/xslt

[37] Clark, J. *Incremental XML Parsing and Validation in a Text Editor.* in *XML Conference & Exposition 2003.* 2003. Philadelphia, USA.

[38] codehaus, *Jaxen.* 2006.http://jaxen.codehaus.org/

[39] Deutsch, A., M.F. Fernandez, D. Florescu, A.Y. Levy, and D. Suciu, *A query language for XML.* Computer Networks, 1999. **31**: p. 1155-1169.

[40] Deutsch, A., M.F. Fernandez, and D. Suciu:. *Storing Semistructured Data with STORED.* in *SIGMOD Conference.* 1999. Philadelphia, Pennsylvania, United States. p. 431-442.

[41] dom4j.org, *dom4j.* 2005.http://www.dom4j.org/

[42] Fegaras, L. and R. Elmasri, *Query Engine for Web-Accessible XML Data.* The VLDB Journal, 2001: p. 251-260.

[43] Fernandez, M., Y. Kadiyska, D. Suciu, A. Morishima, and W.-C. Tan, *SilkRoute: A Framework for Publishing Relational Data in XML.* ACM Transactions on Database Systems, 2002: p. 438-493.

[44] Fernandez, M., W.C. Tan, and D. Suciu, *SilkRoute : Trading between Relations and XML.* Computer Networks, 2000. **33**: p. 723-745.

[45] Fiebig, T., S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T. Westmann, *Anatomy of a native XML base management system.* VLDB Journal, 2002. **11**(4): p. 292-314.

[46] Florescu, D. and D. Kossmann, *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*, in. 1999, INRIA, Rocquencourt, France.

[47] Florescu, D. and D. Kossmann, *Storing and querying XML data using an RDBMS.* IEEE Data Engineering Bulletin, 1999. **22**(3): p. 27-34.

[48] Fong, J. and T. Dillon. *Towards Query Translation From XQL to SQL.* in *Proceedings of the 9th IFIP 2.6 working conference on database semantics (DS9).* 2001. Hong Kong. p. 113-129.

[49] Ford, R., *The Concatenated Datastore - A Utility for Oracle Text.* 2000, Oracle.http://www.oracle.com/technology/sample_code/products/text/htdocs/concatenated_text_datastore/cdstore_readme.html

[50] Frasincar, F., G.-J. Houben, and C. Pau. *XAL: an algebra for XML query optimization.* in *Database Technologies 2002, Thirteenth Australasian Database Conference (ADC2002).* 2002. Melbourne, Australia.

[51] Goldman, R., J. McHugh, and J. Widom. *From Semistructured Data to XML: Migrating the Lore Data Model and Query Language.* in *Proceedings of the 2nd International workshop on the Web and Databases (WebDB '99).* 1999. Philadelphia, Pennsylvania, USA. p. 25-30.

[52] Goldman, R. and J. Widom. *DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases.* in *Proceedings of the Twenty-Third International Conference on Very Large Data Bases.* 1997. Athens, Greece. p. 436-445.

[53] Gray, J., *The Benchmark Handbook for Database and Transaction Processing Systems.* 2 ed. 1993, San Francisco: Morgan Kaufmann Publishers.

[54] Grinev, M., *XQuery Optimization Based on Rewriting*, in. 2003, Moscow State University.

[55] Grinev, M., A. Fomichev, S. Kuznetsov, K. Antipin, A. Boldakov, D. Lizorkin, L. Novak, M. Rekouts, and P. Pleshachkov, *Sedna: Native XML DBMS.* 2004.http://www.modis.ispras.ru/Development/sedna.htm

[56] Grinev, M. and S.D. Kuznetsov:. *Towards an Exhaustive Set of Rewriting Rules for XQuery Optimization: BizQuery Experience.* in *Advances in Databases and Information Systems, 6th East European Conference, ADBIS 2002.* 2002. Bratislava, Slovakia. p. 340-345.

[57] Han, W.-S., K.-H. Lee, and B.S. Lee, *An XML Storage System for Object-Oriented/Object-Relational DBMSs.* Journal of Object Technology, 2003. **2**(3): p. 113-126.

[58] Harold, E.R., *Processing XML with Java.* 2002: Addison Wesley.

[59] Harold, E.R., *XInclude*, in *XML 1.1 Bible.* 2004, Wiley Publishing.

[60] Harold, E.R., *XLinks*, in.*XML 1.1 Bible.* 2004, Wiley Publishing.

[61] IBM, *DB2 XML Extender.* 2005.http://www-306.ibm.com/software/data/db2/extenders/xmlext/index.html

[62] IBM and Informix, *DataBlade Developers Kit Coach.* 2004.http://www-306.ibm.com/software/data/informix/pubs/library/datablade/dbdk/start.htm

[63] Ipedo, *Ipedo XML Database: Technical Overview.* 2002.http://www.ipedo.com

[64] J.P.Lewis and U. Neumann, *Performance of Java versus C++*, in. 2004, Computer Graphics and Immersive Technology Lab,

University of Southern California.

[65] Jagadish, H.V., S. Al-Khalifa, A. Chapman, L.V.S. Lakshmanan, A. Nierman, S. Paparizos, J.M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu, *Timber: A Native XML Database.* VLDB Journal, 2002. **11**(4): p. 274-291.

[66] Jagadish, H.V., L.V.S. Lakshmanan, D. Srivastava, and K. Thompson. *TAX: A Tree Algebra for XML.* in *Database Programming Languages, 8th International Workshop, DBPL 2001.* 2001. Frascati, Italy. p. 149-164.

[67] Jain, S., R. Mahajan, and D. Suciu. *Translating XSLT Programs to Efficient SQL Queries.* in *Proceedings of the Eleventh International World Wide Web Conference, WWW2002.* 2002. New York, NY, USA., Honolulu, Hawaii, USA.: ACM Press. p. 616-626.

[68] java.net,*javacc.* 2005.https://javacc.dev.java.net/

[69] jdom.org, *JDOM.* 2005.http://www.jdom.org/

[70] Jiang, H., H. Lu, W. Wang, and J.X. Yu. *XParent: An efficient RDBMS based XML database system.* in *Proceedings of the 18th International Conference on Data Engineering.* 2002. San Jose, California. p. 335-336.

[71] Johnson, S.C., *YACC-yet another compiler compiler*, in. 1975, Technical Report Computing Science 32, AT&T Bell Laboratories, Murray Hinn, N.J.

[72] Kay, M.H., *SAXON The XSLT and XQuery Processor.* 2005.http://saxon.sourceforge.net/

[73] Khan, L., Q. Chen, and Y. Rao. *A Comparative Study of Storing XML Data in Relational Database Management Systems.* in *Proceedings of the International Conference on Internet Computing, IC'2002.* 2002. Las Vegas, Nevada. p. 277-282.

[74] Khan, L. and Y. Rao. *A Performance Evaluation of Storing XML Data in Relational Database Management Systems.* in *3rd International Workshop on Web Information and Data Management (WIDM 2001).* 2001. Atlanta, Georgia, USA. p. 31-38.

[75] Klettke, M. and H. Meyer, *Managing XML Documents in object-relational databases, PhD. Thesis*, in *Computer Science Department.* 1999, University of Rostock: Rostock, Germany.

[76] Krishnamurthy, R., V.T. Chakaravarthy, R. Kaushik, and J.F. Naughton. *Recursive XML Schema, Recursive XML Queries, and Relational Storage: XML-to-SQL Query Translation.* in *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004.* 2004. Boston, MA, USA. p. 42-53.

[77] Krishnamurthy, R., R. Kaushik, and J.F. Naughton. *Optimizing Fixed-Schema XML to SQL Query Translation.* in *Proceedings of the 28th VLDB Conference.* 2004. Hong Kong, China.

[78] Lee, D. and W.W. Chu. *Constraints-preserving Transformation from XML Document Type Definition to Relational Schema.* in *19th International Conference on Conceptual Modeling.* 2000. Salt Lake City, Utah, USA. p. 323-338.

[79] Lee, D. and W.W. Chu, *CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema.* Data & Knowledge Engineering, 2001. **39**: p. 3-25.

[80] Li, Q. and B. Moon. *Indexing and querying XML Data for Regular Path Expressions.* in *Proceedings of the 27th VLDB Conference.* 2001. Roma, Italy. p. 361-370.

[81] Liu, M. and T.W. Ling. *Towards Declarative XML Querying.* in *Proceedings of 3rd International Conference on Web Information Systems Engineering (WISE 2002).* 2002. Singapore: IEEE Computer Society. p. 127-138.

[82] Lu, H., G. Wang, G. Yu, Y. Bao, J. Lv, and Y. Yu. *XBase: making your gigabyte disk queriable.* in *Proc. of the ACM SIGMOD Int'l Conference on Management of Data.* 2002. Madison, Wisconsin. p. 630.

[83] Lu, L., M. Liu, and G. Wang. *A Declarative XML-RL Update Language.* in *Proceedings of 22nd International Conference on Conceptual Modeling (ER 2003).* 2003. Chicago, Illinois, USA: Springer-Verlag. p. 506-519.

[84] Lu, S., Y. Sun, M. Atay, and F. Fotouhi. *A new inlining algorithm for mapping DTDs to relational schemas*. in *Proceedings of the First International Workshop on XML Schema and Data Management (XSDM2003)*. 2003. New York: LNCS 2814. p. 366-377.

[85] Lv, T. and P. Yan, *Mapping DTDs to relational schema with semantic constraints*. Information and Software Technology, 2006. **48**: p. 245-252.

[86] Manolescu, I., D. Florescu, and D. Kossma. *Answering XML Queries over heterogeneous Data Sources*. in *Proceedings of the 27th VLDB Conference*. 2001. Roma, Italy. p. 241-250.

[87] Manolescu, I., D. Florescu, and D. Kossma, *Pushing XML Queries inside Relational Databases*, in. 2001, INRIA Technical Report No. 4112 Report no. 4112.

[88] McDonald, C., C. Beck, J.R. Kallman, and D.C. Knox, *Mastering Oracle PL/SQL : Practical Solutions*. 2004: Apress.

[89] McHugh, J., S. Abiteboul, R. Goldman, D. Quass, and J. Widom, *Lore: A Database Management System for Semistructured Data*. SIGMOD Record (ACM Special Interest Group on Management of Data), 1997. **26**(3): p. 54-66.

[90] McHugh, J. and J. Widom. *Query optimization for XML*. in *Proc. of the 25th Int'l Conference on Very Large Data Base*. 1999. Edinburgh, Scotland, UK. p. 315-326.

[91] Meier, W. *eXist: An Open Source Native XML Database*. in *Web, Web-Services, and Database Systems*. 2002. Erfurt, Germany. p. 169-183.

[92] Mendelzon, A.O., G.A. Mihaila, and T. Milo, *Querying the world wide web*. International Journal on Digital Libraries, 1997. **1**: p. 54-67.

[93] Microsoft, *The Data Tier: An Approach to Database Optimization*, in *Microsoft SQL Server 2000 Resource Kit*. 2001, Microsoft Press.

[94] Microsoft, *Microsoft SQL Server*. 2005.http://www.microsoft.com/sql/default.mspx

[95] Mignet, L., D. Barbosa, and P. Veltri. *The XML Web: a First Study*. in *The Twelfth International World Wide Web Conference (WWW2003)*. 2003. Budapest, Hungary. p. 500-510.

[96] Mo, Y. and T.W. Ling. *Storing and Maintaining Semistructured Data Efficiently in an Object-Relational Database*. in *The Third International Conference on Web Information Systems Engineering (WISE'00)*. 2002. Singapore. p. 247-256.

[97] Obasanjo, D. and S.B. Navathe. *A Proposal for an XML Data Definition and Manipulation Language*. in *VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb*. 2002. Hongkong Chaina. p. 1-21.

[98] Online-Library, *Introduction to the Dewey Decimal Classification.* 2003, Online Computer Library Center.http://www.oclc.org/dewey/versions/ddc22print/

[99] Oracle, *XML Database Developer's Guide - Oracle XML DB.* 2002.http://www.oracle.com/pls/db92/db92.docindex?remark=homepage

[100] Oracle, *Oracle 9i.* 2004.http://otn.oracle.com/documentation/index.html

[101] Papakonstantinou, Y., H. Garcia-Molina, and J. Widom. *Object exchange across heterogeneous information sources.* in *Proceedings of the Eleventh International Conference on Data Engineering.* 1995. Taipei, Taiwan. p. 251-260.

[102] Pardede, E., J.W. Rahayu, and D. Taniar. *On Using Collection for Aggregation and Association Relationships in XML Object-Relational Storage.* in *ACM Symposium on Applied Computing.* 2004. Nicosia, Cyprus. p. 703-710.

[103] Pardede, E., J.W. Rahayu, and D. Taniar, *Object-relational complex structures for XML storage.* Information and Software Technology, 2006. **48**: p. 370-384.

[104] Parr, T., *ANTLR.* 2005.http://www.antlr.org/

[105] PostgreSQL, *PostgreSQL.* 2005.http://www.postgresql.org

[106] Psaila, G. *ERX: A Conceptual Model for XML Documents.* in *Proceedings of the 2000 ACM Symposium on Applied Computing.* 2000. Como, Italy. p. 898-903.

[107] Robie, J., *The Design of XQL.* 1999.http://www.ibiblio.org/xql/xql-design.html

[108] Runapongsa, K. and J.M. Patel. *Storing and Querying XML data in Object-Relational DBMSs.* in *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops.* 2002. Prague, Czech Republic. p. 266-285.

[109] Runapongsa, K., J.M. Patel, H.V. Jagadish, Y. Chen, and S. Al-Khalifa, *The Michigan Benchmark: towards XML Query Performance Diagnostics.* Information Systems, 2006. **31**: p. 73-97.

[110] Russell, C., O. Schadow, T. Stanienda, F. Velez, R.G. Cattell, D.K. Barry, M. Berler, J. Eastman, and D. Jordan, *The Object Data Standard: ODMG 3.0.* The Morgan Kaufmann Series in Data Management Systems. 2000: Morgan Kaufmann Publishers.

[111] Sahuguet, A. *Kweelt: More than Just "Yet Another Framework to Query XML!* in *SIGMOD 2001 Electronic Proceedings.* 2001. Santa Barbara, CA. p. 602.

[112] Sartiani, C. and A. Albano. *Yet Another Query Algebra For XML Data.* in *Proceedings of the 6th International Database Engineering and Applications Symposium (IDEAS 2002).* 2002. Edmonton, Canada. p. 106-115.

[113] Schmidt, A., F. Wass, M. Kersten, D. Florescu, M.J. Carey, I. Manolescu, and R. Busse, *Why and How to Benchmark XML Databases.* SIGMOD Record, 2001. **30**: p. 27-32.

[114] Schoning, H., *Tamino-Software AG's Native XML Server*, in *XML Data Management: Native XML and XML-Enabled Database Systems*. 2003, Addison Wesley Professional.

[115] Seo, C., S.-W. Lee, and H.-J. Kim, *An efficient inverted index technique for XML documents.* Information and Software Technology, 2003. **45**: p. 11-22.

[116] Shanmugasundaram, J., J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. *Querying XML Views of Relational Data.* in *Proceedings of the 27th VLDB Conference*. 2001. Roma. Italy. p. 261-270.

[117] Shanmugasundaram, J., E.J. Shekita, J. Kiernan, R. Krishnamurthy, S. Viglas, J.F. Naughton, and I. Tatarinov, *A General Technique for Querying XML Documents using a Relational Database System.* SIGMOD Record, 2001. **30**(3): p. 20-26.

[118] Shanmugasundaram, J., K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. *Relational Databases for Querying XML Documents: Limitations and Opportunities*. in *Proceedings of the 25th VLDB Conference*. 1999. Edinburgh, Scothland. p. 302-314.

[119] Shimura, T., M. Yoshikawa, and S. Uemura. *Storage and Retrieval of XML Documents using Object-Relational Databases*. in *Database and Expert Systems Applications, 10th International Conference, DEXA '99*. 1999. Florence, Italy. p. 206-217.

[120] Software, A.G., *Tamino XML Server : White Paper.* 2003.http://www.softwareag.com

[121] Software, S., *Performance Matrics & Benchmarks: Berkeley DB*, in. 2005, Sleepycat Software, Inc.

[122] Sosnoski, D.M., *XML and Java technologies: Document models, Part 1: Performance.* 2001.http://www-106.ibm.com/developerworks/xml/library/x-injava, http://www.sosnoski.com/opensrc/xmlbench/results.html

[123] Suciu, D. *Semistructured data and XML*. in *The 5th International Conference of Foundations of Data Organization (FODO'98)*. 1998. Kobe Japan. p. 1-12.

[124] Tatarinov, I., Z. Ives, A.Y. Halevy, and Daniel S. Weld. *Updating XML*. in *Proceedings of 2001 SIGMOD Conference*. 2001. Santa Barbara, CA, USA. p. 413-424.

[125] Tatarinov, I., S.D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. *Storing and Querying Ordered XML Using a Relational Database System*. in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 2002. Madison, Wisconsin. p. 204-215.

[126] Tian, E.F., D.J. DeWitt, J. Chen, and C. Zhang, *The Design and Performance Evaluation of Alternative XML Storage Strategies.* SIGMOD Record (ACM Special Interest Group on Management of Data), 2002. **31**(1): p. 5-10.

[127] Tseng, F.S.-C. and W.-J. Hwung, *An automatic load/extract scheme for XML documents through object-relational repositories.* The Journal of Systems and Software, 2002. **64**: p. 207-218.

[128] Turker, C. and M. Gertz, *Semantic integrity support in SQL:1999 and commercial (object-)relational management systems.* The VLDB Journal, 2001. **10**: p. 241-269.

[129] UniSQL/X, *UniSQL/X User's Manual Vol I.* 2004.http://dev.unisql.com/dev/manuals/manuals.htm

[130] Varlamis, I. and M. Vazirgiannis. *Bridging XML-Schema and relational databases. A system for generating and manipulating relational databases using valid documents.* in *ACM Symposium on Document Engineering 2001.* 2001. Atlanta, Georgia, USA. p. 105-114.

[131] W3C, *The XML Query Algebra.* 2000.http://www.w3.org/TR/2000/WD-query-algebra-20001204

[132] W3C, *XML Linking Language (XLink) Version 1.0.* 2001.http://www.w3.org/TR/xlink

[133] W3C, *Document Object Model (DOM).* 2003.http://www.w3.org/DOM/

[134] W3C, *XQuery 1.0: An XML Query Language.* 2003.http://www.w3.org/TR/xquery

[135] W3C, *XML Inclusions (XInclude) Version 1.0.* 2004.http://www.w3.org/TR/2004/CR-xinclude-20040413/

[136] W3C, *XQuery Update Facility Requirements, W3C Working Draft.* 2005.http://www.w3.org/TR/xquery-update-requirements/

[137] Wang, G. and M. Liu. *Query Processing and Optimization for Regular Path Expressions.* in *Advanced Information Systems Engineering, 15th International Conference.* 2003. Klagenfurt, Austria. p. 30-45.

[138] Watson, P. *Databases in Grid Applications: Locality and Distribution.* in *Proceedings of the Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases, BNCOD 22.* 2005. Sunderland, UK: Springer-Verlag. p. 1-16.

[139] X-Hive, *X-Hive/DB.* 2005.http://www.x-hive.com

[140] XMLDB, *XUpdate.* 2002.http://www.xmldb.org/xupdate/xupdate-wd.html

[141] Yao, B.B., M.T. Ozsu, and N. Khandelwal. *XBench Benchmark and Performance Testing of XML DBMSs.* in *Proceedings of 20th International Conference on Data Engineering.* 2004. Boston. p. 621-632.

[142] Yoshikawa, M., T. Amagasa, T. Shimura, and S. Uemura, *XREL: A path based approach to storage and retrieval of XML documents using relational databases.* ACM Transactions on Internet Technology, 2001. **1**(1): p. 110-141.

[143] Zaniolo, C., S. Ceri, C. Faloutsos, R.T. Snodgrass, V.S. Subrahmanian, and R. Zicari, *How to Benchmark Database Updates*, in *Advanced Database Systems.* 1997, Morgan Kaufmann Publishers.

[144] Zhang, C., J.F. Naughton, D.J. DeWitt, Q. Luo, and G.M. Lohman. *On supporting containment queries in relational database management systems.* in *ACM SIGMOD Conference 2001.* 2001. Santa Barbara, CA, USA. p. 425-436.

[145] Zhou, A., H. Lu, S. Zheng, Y. Liang, L. Zhang, W. Ji, and Z. Tian. *VXMLR: A visual XML-relational database system.* in *Proc. of the 27th Int'l Conference on Very Large Data Base.* 2001. Roma, Italy. p. 719-720.

[146] Zwol, R.V., P.M.G. Apers, and A.N. Wilschut. *Modelling and Querying Semistructured Data with MOA.* in *proceedings of Workshop on Query Processing for Semistructured Data and Non-standard Data Formats.* 1999. Jerusalem, Israel.