University of Sheffield

Department of Computer Science

A PERFORMANCE ANALYSIS OF A HYBRID RELATIONAL-XML APPROACH TO STORE PARTIALLY-STRUCTURED DATA

Thesis submitted for the degree of PhD

by

Yasser Abdel Kader

March 2007

Abstract

Nowadays, huge amounts of data are stored outside the rigid boundary of highly-structured and traditional database management systems, such as World Wide Web, application data that deals with non-standard data formats, legacy systems and structured documents. On the one hand, this data does not conform to a pre-defined structure and yet it is not completely un-structured. This data is classified as semi-structured data. There is a need to store and manage the large existing collections of semi-structured data and to query it efficiently in a way similar to traditional databases. But as yet, a mature technology for doing so does not exist. However, eXtensible Markup Language (XML) has emerged as the *lingua franca* of the web. XML has the ability to represent all form of structured data (highly-, semi- and un-structured).

This research aims to enhance the performance of storing, querying and retrieving XML data that contain a combination of highly-structured and semi-structured data (this hybrid structuring can be described as partially-structured data), so as to better support classes of application where there is a fixed formal framework for data, but also an ad hoc component. One way to mange XML data is by using relational database management systems. This is based on the robust, well established and optimised performance relational database management systems can offer. The research presented in this thesis is concerned with seeking ways of further exploiting the latter advantages in adapting relational technology to store XML data.

To this end, the research has proposed a hybrid relational-XML storage model to store partially-structured XML encoded data, in which a combination of structure mapping and XML types are used within a relational database management system, so as to exploit pre-knowledge of the highly-structured part in query processing while allowing flexibility to store the semi-structured part. A set of experiments were designed to evaluate the query performance for partially-structured data using structure mapping to relational tables, XML types and the hybrid model. These experiments were evaluated using a standard benchmark set of queries.

The analyses of the experiments' results establish the impact on query performance as structuredness, volume and query characteristics change. The results of the experiments showed that there was no one storage model that outperforms all other models in all cases. In most of the cases, this hybrid model performed better than both the relational and XML data type models. The research proposed a method, by which the results of the performance analysis can be utilised by the database designer to seek optimal relational storage models for XML-encoded partially-structured data.

Acknowledgments

Throughout the course of my research, I have been helped and encouraged by many people. I would like to express my gratitude to all those who have supported me through out this experience.

Special gratitude is due to my supervisors for their encouragement, supervision and objective criticism and guidance. I am deeply indebted to Dr Barry Eaglestone for his continuous support and encouragement, for keeping me motivated and focused through the entire process. I am also deeply indebted to Dr. Siobhán North for her great help, valuable suggestions and constructive criticism.

My deepest gratitude are due to Dr. Gamal Mokhtar, the president of the Arab Academy for Science and Technology and Maritime transport, without whom help and support the conduction of my PhD would not be possible.

Finally, I owe many thank for my mother for her care and support. I wish to express my sincere loving thanks to my wife Rania Abdel Galil for her understanding, immense help and encouragement and for always being there for me and also to my beloved children Karim and Noura who gave me an enjoyable life outside the research.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

DOM	Document Object Model
DTD	Document Type Declaration
HTML	Hybrid Text Markup Language
NXD	Native XML Database
OEM	Object Exchange Model
WWW	World Wide Web
XLink	XML Linking Language
XML	eXtensible Markup Language
XPath	XML Path Language
XPointer	XML Pointer Language
XSchema	XML Schema
XSL	The Extensible Stylesheet Language
XHTML	The Extensible Hyper Text Markup Language

Chapter 1 Introduction

1.1 Introduction

Nowadays, huge amounts of data are stored outside the rigid boundary of highly-structured and traditional database management systems. One example of that is the World Wide Web, the largest source of data by volume ever created by human beings. Other examples include application data that deals with a non-standard data formats, legacy systems and structured documents (Suciu 1998). On the one hand, this data does not conform to a pre-defined structure and on the other hand it is not completely un-structured. Therefore, this data is classified as semi-structured data (Abiteboul 1997, Buneman 1997, Suciu 1998, Abiteboul et al. 1999, Abiteboul 2001 and Florescu 2005).

There is a need to store and manage the large collections of semi-structured data and to query it efficiently in a way similar to traditional databases, but as yet, a mature technology for doing so does not exist. However, eXtensible Markup Language (XML) has emerged as the *lingua franca* of the web (Vianu 2003) and can be seen as "the forthcoming semi-structured standard of the Web" (Abiteboul 2001). Consequently, XML data models and their query languages are now emerging, partly based on previous research into semi-structured data models and query languages. For example, Lore (McHugh et al. 1997 and Abiteboul et al. 1997) was firstly designed as a semi-structured database management system but then was migrated to store XML data (Goldman et al. 1999). However, a second and complementary approach towards establishing an XML database theory and technology has been to adapt that of conventional relational or object-relational databases. Examples can be found in Shanmugasundaram et al. 1999 and 2001, Shimura et al. 1999, Florescu and Kossmann 1999, Schmidst et al. 2000, Klettke and Mayer 2001, Yoshikawa and Amagasa 2001, Kudrass 2002, Kudrass and Conrad 2002, Runapongsa and Patel 2002, Tian et al. 2002, Amer-Yahia and Srivastava 2002, Bohannon et al. 2002, Kuckelberg and Krieger 2003, Han et al. 2003, Harding et al. 2003, Leonov and Khusnutdinov 2004, Pal et al. 2004, Lu et al. 2006, Na and Lee 2005, Balmin and Papakonstantinou 2005, Chaudhuri et al. 2005, Qin et al. 2005, Pardede et al. 2005, 2006 and Pal et al. 2006. The advantage of the latter approach is based on the robust, well established and optimised performance relational database management systems can offer. A recent empirical study of XML data management showed that using relational database management systems

outperform using native XML database systems in processing XML data (Lu et al. 2006). This finding depends on a number of factors such as document structuredness, data size and the queries' workload.

The research presented in this thesis concerns seeking ways of further exploiting the advantages of adapting relational technology to store XML data. Specifically, the research considers a class of XML data which is a hybrid between highly-structured and semi-structured data (this is defined in section 3.4 as partially-structured data). The primary aim is to establish for this class of partially-structured XML data, whether pre-knowledge of the structured component can be exploited when storing the data within a relational database will improve its query processing efficiency. This improvement can be achieved by exploiting relational query processing and optimisation technology for the highly-structured part rather than dealing with the data as totally semi-structured, while allowing flexibility in storing and querying the semi-structured part of the data.

1.2 Research Motivation

The motivations behind this research are rooted in issues relating to the structuredness of XML document collections and their implications on the query processing performance. The research is concerned with a class of XML-encoded data that can be described as a hybrid between highly-structured and semi-structured data, referred to as partially structured XML data. Specifically, this study seeks to exploit the knowledge of the highly-structured part to improve query processing performance.

XML has become a focus for research in both the database as well as the document research communities (as in section 2.3.1.3). This research effort is motivated by strengths of XML, including: its simple format, the separation of the data from how this data is formatted, the internationalisation capability, platform independence, extensibility, human readability as well as machine readability, processing instruction and the large investment in XML applications that already exists nowadays. These strengths make XML appropriate as a way to store and exchange data on the web. However, achieving good XML query processing performance is problematic because of the irregular structures inherent in the semi-structured data, which means that conventional query optimisation technology cannot be used in a straightforward way (section 2.2.3).

One possible approach to addressing the above querying efficiency problem is to exploit the inherent structures of specific XML documents. In developing this approach it is useful therefore to classify XML documents according to their structuredness, as has been done in (Barbosa et al. 2001, Yao et al. 2002 and Bourret 2005), where XML documents are classified either as highly-structured, semi-structured or un-structured (see section 3.3). Querying semi-structured or un-structured data is problematic for query processing and incurs significant overheads (see section 2.2.3), whereas the pre-knowledge of the uniform structures of highly structured data opens the gate for more efficient query processing using well established technologies, such as those developed for relational databases. However, many XML documents are in fact a combination of highly-, semi- and un-structured data. This poses a question; can querying overheads associated with these documents be reduced by exploiting the knowledge of the parts of a document for which the data is highly-structured? In order to address this question, it is necessary to focus on a class of hybrid highly-structured and semi-structured documents, which can be defined as partially-structured documents (as in section 3.4). In such documents, there is a well defined and prescribed structure in part of the document as well as an ad-hoc semi-structured part.

Given the existence of large data sets and applications that can be classified as partially-structured data, such as (bibliographic databases, movies databases, health care system databases and product catalogue databases), there is clearly a need for data management functionality for this class of data. That is to say, to organise, store, query, restructure and manipulate large collections of partially-structured XML data in an efficient way. This requirement is being addressed by applying two main strategies, i.e., developing native XML database management systems, and developing systems which utilise and extend conventional relational database management systems. A potential advantage of the latter approach is that it applies and builds on the years of research and development that provided a mature, stable, scalable and effective technology for query optimisation and processing of highly-structured data. The relational database is currently the major database technology in use and is likely to maintain its dominant position in the foreseeable future. So, the research concentrates on using relational databases for XML data management, but seeks a better way to store and query the class of partially-structured data.

Therefore, the basic motivation for this research is the need for improved query processing performance of partially-structured XML documents based on the use of relational database management systems. The following section presents the hypothesis emerging from this motivation.

1.3 Research Hypothesis

Following on from the argument developed in the preceding section, the research hypothesis is:

> *For the class of XML documents which contains both a prescribed highly-structured part and a semi-structured part, performance enhancement may be achieved over existing query processing techniques for semi-structured documents by using relational database query processing and optimisation technology to exploit pre-knowledge of the prescribed highly-structured part of the data.*

The research tests this hypothesis by introducing and evaluating a new model to store partially-structured documents. In the proposed model, the highly-structured part is stored using structure mapping into a relational database (Shanmugasundaram et al. 1999 and 2001,Shimura et al. 1999, Schmidst et al. 2000, Klettke and Mayer 2001, Yoshikawa and Amagasa 2001, Kudrass and Conrad 2002, Runapongsa and Patel 2002, Tian et al. 2002, Amer-Yahia and Srivastava 2002, Bohannon et al. 2002, Kuckelberg and Krieger 2003, Han et al. 2003, Harding et al. 2003, Leonov and Khusnutdinov 2004, Pal et al. 2004, Lu et al. 2006, Na and Lee 2005, Balmin and Papakonstantinou 2005, Chaudhuri et al. 2005, Pardede et al. 2005, 2006 and Pal et al. 2006). This strategy facilitates the use of existing relational optimisation techniques when querying the structured part of the data, instead of treating data as if it is totally semi-structured. On the other hand, the proposed model uses XML extensions to the relational model (SQL:2003, SQL:2006 (ISO/IEC 9075, part 14, XML-related specifications - SQL/XML)) to store the semi-structured part as a semi-structured data model, and therefore allowing flexibility in dealing with this part of the document. A number of experiments were designed to compare and evaluate the performance of the hybrid model against its two base models; structured mapping approach and storing the whole XML document into an XML data type. The research is concerned only with large, possibly complex documents where query processing performance becomes an issue.

1.4 Research Contributions

This section presents briefly the main findings and contributions of this research (these are discussed in more detail in section 6.2):

- The research has proposed a hybrid relational-XML storage model to store partially-structured XML encoded data, in which a combination of structure mapping and XML types are used within a relational database, so as to exploit pre-knowledge of the structured part in query processing.
- A performance analysis of the above proposed hybrid relational-XML model for storing and querying partially structured data, based on a standard benchmark set of queries (XBench, Yao et al. 2002, 2003, 2004), which establishes the impact on query performance as the structuredness, volume and query characteristics change.
- The research has studied the effect of partitioning partially-structured XML data in two different dimensions:
    - The first dimension concerns the ratio of semi-structured to highly-structured components of the schema.
    - The second dimension concerns the ratio of semi-structured to highly-structured data instances.
- A method, by which the results of the above performance analysis can be utilised by the database designer to seek optimal relational storage models for XML-encoded partially-structured data.

1.5 Outline of the Thesis

This thesis is organised as follows:

- ***Chapter 2 - Research Background***: This chapter sets the scene for the subsequent study of this research. It presents the literature review of relevant to the research. This chapter starts by introducing semi-structured data, its origin, definition, characteristics, data models and its query languages. Then it moves to the second key technology to this study which is XML. It presents its origin, strengths and weaknesses, its data models, and its query languages. Then it

compares and contrasts semi-structured data and XML. Finally, the different storage models for XML encoded data are discussed.

- ***Chapter 3 - Partially Structured XML***: This chapter builds upon the literature review in chapter two by firstly presenting the research motivations for the study presented in this thesis and formulating the hypothesis it addresses. Then it discusses a categorization of XML documents according to their degree of structuredness, so as to analyse structural properties for which this approach is applicable. Accordingly, it defines the class of partially-structured XML documents as a hybrid of highly-structured and semi-structured data. Then it shows an example of this class of data and presents its advantages. Appropriate storage models must be utilised in order to exploit structural pre-knowledge, hence, a review and discussion of different storage models for XML data proposed by research are presented and their potential for storing and querying partially-structured data are analysed. The conclusion of this analysis is an elaboration of the hypothesis, in which a storage structure is proposed, which has the potential to realise performance enhancements for partially-structured data.

- ***Chapter 4 - Experiment Design***: This chapter presents the design of a series of experiments to evaluate the research proposed storage model as a mean of testing the research hypothesis. These experiments are designed to compare the relative performance of the proposed hybrid model against the two base models it combines. This chapter discusses the objective of the experiment, followed by the strategy for achieving those objectives. It then presents the current SQL/XML standard and describes the different storage models that are used in the experiment. Then it discusses the current benchmarking techniques proposed by the research for XML, it compares between these different benchmarking techniques and nominates the most suitable benchmark technique that can be adopted by the research. It then discusses the adaptation needed for this technique; this includes both the data set and query set used in the experiment. It shows how the experiments are designed and how the results are to be evaluated. The chapter concludes by presenting both the hardware and the software to be used in the experiment.

- ***Chapter 5 - Experiment Analysis***: In this chapter, the experiments designed earlier are analysed. The chapter presents and discusses the results of the experiments grouped by their query functionality. This is followed by an overall analysis of the different storage strategies with respect to the different variants the experiments were designed to measure. These variants are: storage strategy, query type, data structuredness, scalability and database storage size. This is followed by a discussion about the experiments' limitations and general finding of the experiments' results.

- ***Chapter 6 – Conclusion and Future Directions***: This chapter concludes this research. The chapter discusses the main findings and contribution of this research. It also outlines some avenues of the future research work.

Chapter 2 Research Background

2.1 Introduction

This chapter sets the scene for the subsequent chapters in that it presents the literature related to this study. The chapter introduces semi-structured data, its origin, definition, characteristics, data models and its query languages. It then moves on to the second key technology relevant to this study which is XML. It presents the origins of XML, its strengths and weaknesses, its data models, and its query languages. Then it compares and contrasts semi-structured data and XML. Finally, the different storage models for XML are discussed.

2.2 Semi-Structured Data

Nowadays, huge amounts of data are stored outside the rigid boundary of traditional database management systems. One example is the World Wide Web, the largest source of data by volume ever created by human beings. Other examples include application data that deals with non-standard data formats, legacy systems and structured documents (Suciu 1998). On one hand, this data cannot conform to a pre-defined structure and on the other hand it is not completely un-structured. This data can be classified as semi-structured data (Abiteboul 1997, Buneman 1997, Suciu 1998, Abiteboul et al. 1999, Abiteboul 2001 and Florescu 2005). There is a need to store and manage the large collection of such data and to query it efficiently in a way similar to traditional databases, but as yet, a mature technology for doing so does not exist.

This section discusses semi-structured data by firstly introducing it. Then section 2.2.2 describes the *de facto* data model for semi-structured data, followed by a discussion of its query languages in section 2.2.3. The research issues related to semi-structured data are presented in section 2.2.4. Finally, section 2.2.5 concludes this discussion.

2.2.1 Introduction

This section presents the origin of semi-structured data, followed by the motivation for the research in this area. Then semi-structured data is defined and finally semi-structured data characteristics are presented.

2.2.1.1 Semi-Structured Data's Origin

The concept of semi-structured data was firstly introduced in 1995 in the context of data integration between heterogeneous information systems in the TSIMMIS project, Stanford University (Chawathe et al. 1994, Garcia-Molina et al. 1995, 1995a and Hammer et al. 1997). The main motivation was that the rigid traditional data models were not flexible enough to cope with the demands of data integration. The Object Exchange Model (OEM) (Papakonstantinou et al. 1995) was developed within the TSIMMIS project as a graph based approach. The OEM became the "*de facto* standard data model for the semi-structured data" (Suciu 1998) and it has been widely adopted in semi-structured data research. For example, it was used to provide the theoretical basis for the Lore project (McHugh et al. 1997) in which a complete semi-structured database was researched and developed. The OEM model is further discussed in section 2.2.2.1.

2.2.1.2 Motivations to Study Semi-Structured Data

Semi-structured data has become a focus for research in the database research field. The motivations for this can be summarised as follows:

- Data management: A huge amount of data is stored outside structured database systems in different formats such as text markup languages for example SGML (SGML 8879: Online, Goldfarb 1990), HTML (HTML: online) and XML (XML: online), legacy systems, scientific data that is stored in a very complex structure and file systems. Such data does not fit into the highly structured data models of database technology, but there is a need for it to be treated as a database. Treating this data as a database will allow an efficient way to organise, store, query, restructure and manipulate this data, while controlling redundancy, concurrency and security of the data.

- Data Integration: semi-structured data is used to facilitate data integration because of its flexible structure (as in the TSIMMIS project, Chawathe et al. 1994, Garcia-Molina et al. 1995, 1995a and Hammer et al. 1997). Although some source data may be highly structured and stored in a database system, other sources may lack this structure and must therefore be stored in an unstructured manner, such as a text format. However, because there is a need to integrate these heterogeneous data sources, it creates a need for a highly flexible data format to facilitate the integration of both structured and semi-structured

data. Another example of data integration research project was by Al-Wasil et al. (2006 and 2006a). They established an XML Metadata Knowledge Base (XMKB) to assist in the integration of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents.

- Data Exchange: the semi-structured data model has proven to be flexible enough to be used as a data exchange data model between different data sources (as in the OEM Papakonstantinou et al. 1995).

- XML: XML is emerging as the *lingua franca* of the web (Vianu 2003) and can be seen as "the forthcoming semi-structured standard of the Web" (Abiteboul 2001). Studying semi-structured data models and query languages directly affect the studying of XML data models and its query language. For example, Lore (McHugh et al. 1997 and Abiteboul et al. 1997) was firstly designed as a semi-structured database management system but then was migrated to store XML data as well (Goldman, R. et al. 1999). Section 2.4 discusses this relation in more detail.

- Browsing: Techniques are also required to view structured data as a semi-structured data for browsing purposes and sometimes ignore the schema, if it exists, for this purpose.

These are the main motivations for researchers to study semi-structured data. These motivations therefore pose the question: To what extent can these problems be solved by adapting the mature theories and technologies of relational database? This question is the focus of this thesis and is discussed in more detail in the following sections.

2.2.1.3 Definition of Semi-Structured Data

There are two keywords that can describe semi-structured data. (Abiteboul 1997, Buneman 1997, Suciu 1998, Abiteboul et al. 1999 and Abiteboul 2001) 'Self-describing data' is the first, where data structure is stored with the data itself as metadata using labels. These labels represent the semantic of each data element. Further, data values are associated with each other by an embedded hierarchy which represents the natural relationship between data elements. 'Schema-less' is the second, where there is no fixed, rigid schema that the data should follow.

The above features allow semi-structured data to be flexible enough to host irregular structures, represented by missing data on the one hand and duplicate data on the other. Further more, it allows the data structure to change rapidly and unpredictably.

The definitions of semi-structured data are based around the irregularity of the data it can represent. For example: Abiteboul (2001) defined semi-structured data as "Data that presents some regularity (it is not an image or plain text) but perhaps not as much (structured) as some relational data or ODMG data." In another paper, Abiteboul et al. (1999) defined it as "Data that is irregular or that exhibits type and structural heterogeneity since it may not conform to a rigid, predefined schema".

Neither of the two definitions is precise. Moreover, there is no clear cut formal definition of semi-structured data which all researchers have agreed upon.

In the context of this research, the second definition by Abiteboul et al. (1999) is used:

> *Semi-structured data is data that is irregular or that exhibits type and structural heterogeneity since it may not conform to a rigid, predefined schema.*

This definition fits this research, as it shows the flexibility of semi-structured data in its ability to host irregular data without a prerequisite rigid and predefined schema.

After defining semi-structured data in this section, the following section presents its characteristics in more detail.

2.2.1.4 Semi-Structured Data Characteristics

Characteristics of semi-structured data can be split into two main groups; from the structure point of view and from the schema point of view. The key characteristics of semi-structured data that relate to structure can be summarised as follows (Abiteboul 1997, Buneman 1997, Suciu 1998, Abiteboul et al. 1999 and Abiteboul 2001)

- The Structure is irregular: it is not totally unstructured (as images and plain text), on the other hand it does not conform to a rigid structure. As such, data does not fit, for example, into a tabular format as in relational data model. Data collections often contain heterogeneous and incomplete data elements or extra information for some data elements.
- Heterogeneous representation of information: semi-structured data is flexible enough that within the same data collection, the same data elements can be

diverse in kind or nature. For example, part of the data may contain temperature stored in Fahrenheit and in another part as Celsius. Data can be represented in one part of the collection as a string (name of person) and in another part as a tuple (first and last name of a person).

- An implicit structure: the structure is embedded inside the document itself and there is no attached external structure. Although there have been some attempts such as Data Guide in Lore (Goldman and Widom 1997) to discover the structure from the data itself and store it in a structure equivalent, in a sense, to the relational schema for relational databases called data guide.

- A partial structure: semi-structured data may comprise both fully and partially structured data. For example, portions of the data, such as bitmaps, may not have any formal structure, while other parts may have a rigid formal structure.

The above characteristics lead to the definition of semi-structured data as the class of data that has a degree of irregularity in its structure. This represents the first part of the adopted definition by Abiteboul et al. (1999).

The second view point that semi-structured data characteristics can be seen from is the schema and its role in defining the data. In general terms, a schema for any data collection defines the data objects that are permissible in that collection. An explicit schema exists in conventional structured data models to define the organisation or structure of the data. In fact, the schema is a key component of a database system, since the database design process leads to the definition of a schema which then prescribes how the data is structured. Also, the schema plays a very important role in query optimization, since it provides pre-knowledge of how data is structured.

The concept of a schema is also applicable to semi-structured data collections. However, unlike structured data models, a schema for a semi-structured data collection is not always explicitly defined. In some forms of semi-structured data, an explicit separate schema does not exist while in other forms an explicit schema exists, but it places loose constraints on the data (Buneman 1997). This directly indicates that the schema does not play the same role in semi-structured data as in conventional data models (such as relational and object oriented data models). In particular schema rules are normally violated by the different structural irregularities that typically occur within semi-structured data, either in missing or duplicated data elements.

The other issue related to the schema is that it can be contained within the data itself. Whether the schema for a semi-structured data collection is implicit or explicit, the heterogeneous structures within the collection require that the structure of each item in a collection must be made explicit, thus in addition to the schema which constrains the content of the collection, each item within the collection must have its own schema.

The above situation requires terminology to distinguish the different forms of schema associated with semi-structured data. Accordingly, I shall refer to the *implicit* or *explicit collection schema* which defines those items which may be part of the collection and the *item schemas* embedded in each data item within a collection.

Note that the requirement for item schemas leads to inefficiency in storing semi-structured data, since an item schema needs to be stored with each data element, even where they have an identical structure. This can also lead to inefficient query evaluation. Also, without having access to an explicit collection schema in advance, it may be necessary to traverse the whole data collection to look for a simple regular path expression. Furthermore, it is not an easy task to formulate queries without knowing the structure of the data (Suciu 1998). These problems have motivated research into collection schema extraction (such as Data Guides (Goldman and Widom 1997), adding structure to unstructured data (Buneman et al. 1997) inferring structure in semi-structured data (Nestorov et. al. 1997) and discovering structure associations of semi-structured data (Wang and Liu 1999)). These research efforts tried to discover the schema from semi-structured data using data mining techniques, and then used this schema to enhance query performance.

Other characteristics of semi-structured data from the schema point of view were described by Abiteboul (1997) as follows:

- A priori schema vs. a posterior data guide: unlike traditional structured data sources in which the structure is designed before the data, in semi-structured data the structure is discovered from the data itself.
- Indicative structure vs. constraining structure: since the schema does provide a loose constraint over the semi-structured data, the structure here is indicative rather than constraining.
- The schema can be very large: large irregularity of the data leads to a large description of the data structure.

- The schema collection may be ignored: in some queries of a discovery nature, data browsing or information retrieval search is used instead.
- The schema collection may evolve rapidly: because of the nature of the data, the schema can change much more rapidly than in the traditional databases.
- The types of data elements are eclectic: the data types are not precise. The same data element maybe represented by different data types in the same data collection.
- The distinction between a schema item and data is blurred: there are no border lines between the schema and the data; this is due to the schema being embedded inside the data and putting loose constraints on the data.

An important consequence of the above characteristics is that, because of the lack of a predictable pre-defined structure for data within a collection, the description of the data, that is to say, its schema item, is specific to each document or record, and is therefore contained within the data itself. This represents the second part of the definition by Abiteboul et al (1999), the data may not conform to a predefined schema.

The above discussion has focused on the inherent irregularity of semi-structured data and the consequential need to embed a definition of structure within the data itself. In addition, a definition of semi-structured data should define how the data may be structured. The latter is the focus of various attempts to design a semi-structured data model. The next section presents semi-structured data models in general. Then it is followed by the discussion of the *de facto* standard semi-structured data model; the Object Exchange Model OEM (Papakonstantinou et al. 1995).

2.2.2 Semi-structured Data Model

Having discussed the characteristics and importance of semi-structured data, this section discusses the ways in which semi-structured data can be modelled.

Semi-structured data comprises self-explanatory data, which means that associated with data values are labels that represent the semantics of the respective pieces of data. Further, data values are associated with each other by an embedded hierarchy which represents the natural relationship between data. Therefore semi-structured data has a natural representation as a rooted directed graph or a rooted directed tree with labelled

edges. Although cycles are allowed in the data, generally the term tree is used (Buneman 1997).

In the tree, the leaf nodes represent the data values; the edges represent the embedded hierarchy and the relation between different data. The edge labels denote the semantics of the data represented by the child node. Figure 2.1 shows an example of a semi-structured data represented as a label-edge graph. For example, the 'Author' (represented as edge label) is 'Miguel Nunes' (represented as a leaf node).

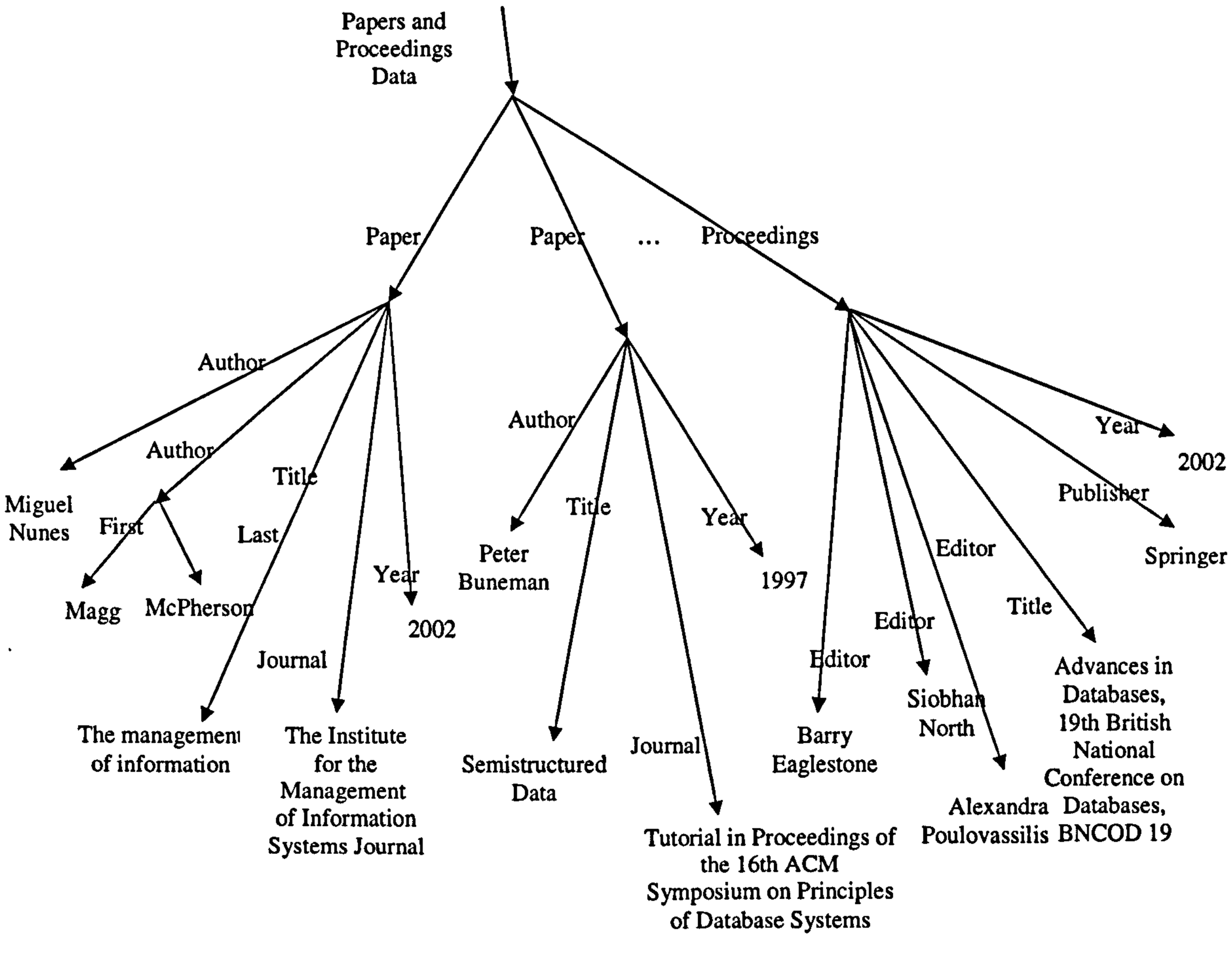


Figure 2.1 Semi-structured Data as Edge Labelled Directed Graph

The type of this kind of labelled tree can be represented, as in (Buneman 1997):

$$type\ label = int\ |string\ |\ldots\ |symbol$$

$$type\ tree = set\ (label \times tree)$$

The type label describes a tagged union of variant, while the type tree consists of a set of label/tree pairs. This model represents unordered edges in the tree.

A small variation to the above model was described in Loral (Abiteboul et al. 1997). There the leaf nodes are labelled with data, internal nodes are node labelled and edges are labelled only with symbols.

type base = int /string /...

type tree = base /set (symbol ×tree)

The Object Exchange Model OEM model (Papakonstantinou et al. 1995) is an example of the above representation. It is discussed in more detail in the following section.

The third possibility is to allow labels on internal nodes.

type label = int /string /... /symbol

type tree =label× set (label ×tree)

These models are all equivalent, in that each model can easily be transformed into either of the others. Specifically, it is easy to define mapping between the first two models. In addition to that, by introducing extra edges into the third representation, this model can be converted to one of the first two models. In the following section, the OEM model (Papakonstantinou et al. 1995) is presented in more detail as it is considered as "the *de facto* standard data model for the semi-structured data" (Suciu 1998).

2.2.2.1 OEM Data Model

In the Object Exchange Model OEM (Papakonstantinou et al. 1995), the graph structure of semi-structured data is represented as nested quadruple structures, which correspond to the nodes of the graph structure, comprising the following elements:

- Label: which describes what the object represents
- Type: the data type of the object's value, which can be either an atomic type (such as integer or string) or type set.
- Value: the value of the object. It can also be an object-Id value where the type is object-Id; this means that links can be provided in the model thus defining the edges of the graph structure.
- Object-Id: a unique identifier for the object or null

OEM is a logical data model, since it does not specify how data is stored physically. It is much simpler than the Object-Oriented data model since it supports only object nesting and object identifying. Other features such as classes, methods and inheritance are omitted. Also, labels are used to define item schemas instead of a schema in a similar object oriented model (therefore, the collection schema is implicit) (Papakonstantinou et al. 1995). This makes it simple enough to cope with the frequent changes typical in semi-structured data.

Abiteboul et al. (1999) have demonstrated that the OEM is sufficiently expressive to represent highly structured data such as the relational data model and the object-oriented data model or semi-structured data or even unstructured data. Also, it can represent, in a natural way any missing attributes, duplicate attributes with the same name and different types for the same attribute in different objects. It can also represent attributes with irregular structure, for example, in one object as an atomic element while in another as a record such as a name of a student as one field or as a record with first and last name fields.

This and the preceding sections have described some of the approaches to modelling semi-structured data, and the *de facto* standard semi-structured data model, OEM in more detail. There are variations to this model, i.e., the labels which denote the semantics of the data can either be associated with nodes or edges (Abiteboul et al. 1999). However, since it is possible to transform between these variations, the choice of which variant to use is determined by other factors, such as the ease of data manipulations and query operations. This latter aspect is overviewed in the following sections by presenting semi-structured data query languages.

2.2.3 Query Languages for semi-structured Data

This section presents the query languages for semi-structured data. In particular, it traces how they evolve to cope with its flexibility and its data model.

A number of query languages have been proposed to deal with semi-structured data and XML such as UnQL (Buneman et al. 1996), MSL (Papakonstantinou et al. 1996), Lorel (Abiteboul et al. 1997), StruQL (Fernandez et al. 1997), XQL (Robie et al. 1998), XML-QL (Deutsch et al. 1998) and Quilt (Chamberlin et al. 2000). Generally, researchers have explored two approaches to construct a query language for semi-structured data (Buneman 1997). The first is to adapt a conventional database query

language, such as Structured Query Language SQL (SQL: 2006) or Object Query Language OQL (Cattell et al 2000), by redefining the semantics and adding the appropriate features to cope with the new requirement needed to query semi-structured data. The second is to start from a language based on some formal notion of computation appropriate to navigating tree structures and filtering text data, and then to convert this language into an acceptable syntax. Section 2.3.3 presents querying XML in more detail. The following subsections present an example of each class, section 2.2.3.1 presents Lorel (Abiteboul et al. 1997) as an example of the first class while section 2.2.3.2 presents UnQL (Buneman et al 2000) as an example of the second class.

2.2.3.1 Lorel

Lore (Lightweight Object REpository) (McHugh et al. 1997) is a complete general purpose semi-structured database management system. Lorel (Abiteboul et al. 1997) is its query language. Lorel can be viewed as an extension of the Object Query Language OQL (Cattell et al 2000) while Lore as an extension of the ODMG data model (Abiteboul et al. 1997).

A typical example of a Lorel query is:

Select X.Title

From Paper X

Where X.Year > 2000

This query returns the titles of all papers where its year is greater than 2000.

There are a number of differences between Lorel and OQL (Suciu 1998):

- Type coercion: Lorel deals with type coercion while OQL does not. For example, if in the *where* statement the Year = 2000, then Lorel returns the data with a string data type as "2000" and integer as 2000
- Missing Attributes: OQL produces an error if an attribute is missing, while Lorel simply ignores any missing attribute.
- Singletons or sets attributes: If the data contains a set of attributes, then Lorel returns this data item as long as at least one of the data items satisfy the where statement condition. Meanwhile, OQL returns this data if all the data items

satisfy the where statement condition. They will produce the same behaviour in the case of a singleton data item.

- Generalized path expressions: by extending the OQL-like path expressions with wild characters to arbitrary regular expressions. For example, '?' denotes an optional path expression. '*' denotes Kleene closure. '+' denotes strict Kleene closure.

Finally, Lorel does not require a *from* clause, so the above query can be rewritten as

Select Paper.Title

Where Paper.Year > 2000

It has a rule that the common paths correspond to the same object (unless otherwise specified).

After illustrating briefly Lorel in this section as an example of adapting a query language for semi-structured data based on OQL, the next section presents UnQL as an example of a query language based on a formal notion.

2.2.3.2 UnQL

Unstructured Query Language UnQL (Buneman et al. 2000) presents an example of a query language based on formal notation (in contrast to Lore which was based on OQL).

UnQL model is based on structure recursion functions on a tree data structure. It can be extended to work on arbitrary graphs. These functions are introduced in UnQL query in a top-down manner. They use pattern matching to provide a means of selecting data. A typical example of an UnQL query syntax which returns all titles in a graph is:

```
fun     f1 (T1 U T2) = f1(T1) U f1 (T2)
      | f1 ({L: T})  = if L = title then {result: T} else f1 (T)
      | f1 ({})      = {}
      | f1 (V)       = {}
```

The uppercase symbols are variables. The patterns are represented in the left hand side of each equation. These functions are applied in order. L represents the label while T represents the value and hence the condition is to return a value when the label equals

the title. The third line represents that for an empty set, no variables are bound. V represents a catch-all clause, it matches anything and in particular it matches atomic values. The result is a set and therefore the duplication is eliminated. (Buneman et al. 2000).

UnQL was one of the first introduced query languages for semi-structured data which is simple and has an optimisable algebra. The following section compares Lorel and UnQL.

2.2.3.3 Lorel vs. UnQL

There are a number of points on which Lorel and UnQL can be contrasted:

- Base: Lorel is an extension of OQL, while UnQL is based on structure recursion.
- Coercion: Lorel uses coercion, while UnQL does not. For example. Lorel deals with the data if its type is integer (as 2000) or if its type is string (as "2000") while UnQL deals with the data either as an integer or as a string. Lorel is more flexible than UnQL in this scenario as a more precise knowledge of structure is needed for UnQL to express queries (Abiteboul et al. 1999).
- Complex restructure: UnQL allows a more complex restructure than Lorel by using the structural recursion. This complex transformation involves deep traversal of the data followed by a reconstruction of an entirely new graph. This can be done in UnQL by creating recursive functions based on a certain strict pattern (Abiteboul et al. 1999).

After briefly describing the two ways to construct a query language for semi-structured data and then contrasting them, the following section discusses in general the research issues surrounding semi-structured data.

2.2.4 Semi-structured Data Research Issues

Research in semi-structured data complements previous research in conventional databases. It started in the mid 1990's after the invention of the OEM model in 1995. The invention of the World Wide Web (and specifically XML in 1998) directed the research towards covering both technologies (semi-structured data and XML). This is because of the clear analogy between them which is discussed in more detail in section 2.4.

As evidence of this analogy, Lore (McHugh et al. 1997) started as a semi-structured database management system and then migrated (Goldman et al. 1999) to host XML data. Another recent example is in the Dagstuhal Seminar 'Foundations of Semistructured Data' (Neven et al. 2005). Almost all the research issues discussed are related to both semi-structured data and XML. In this section, the research conducted in general areas related to semi-structured data is discussed while in sections 2.5 and 3.5, research issues related explicitly to XML storage strategies are presented.

The literature on semi-structured data and XML varies across data representation, management and administration issues (such as web site management, general purpose management of semi-structured data, data conversion, schema specification and schema extraction) and performance issues (such as optimisations and formal aspects and indexing). Based on the discussion of semi-structured research issues by Suciu (1998), this section presents a summary of these research issues:

- General purpose management of semi-structured Data such as Lore system (McHuge et al. 1997 and Abiteboul et al. 1997). It was built in Stanford University. It is a complete database management system for semi-structured data. Lore started from scratch to cover all the aspects for a semi-structured database management. It used the OEM (Papakonstantinou et al. 1995) as its data model. The project was closed as a success in year 2000.
- Web Site Management: Fernandez et al. (1998, 2000) developed STRUDEL system for web site management by separating the management of the site's data, the creation and management of the site's structure and the visual representation of the site's pages The management of the site's data

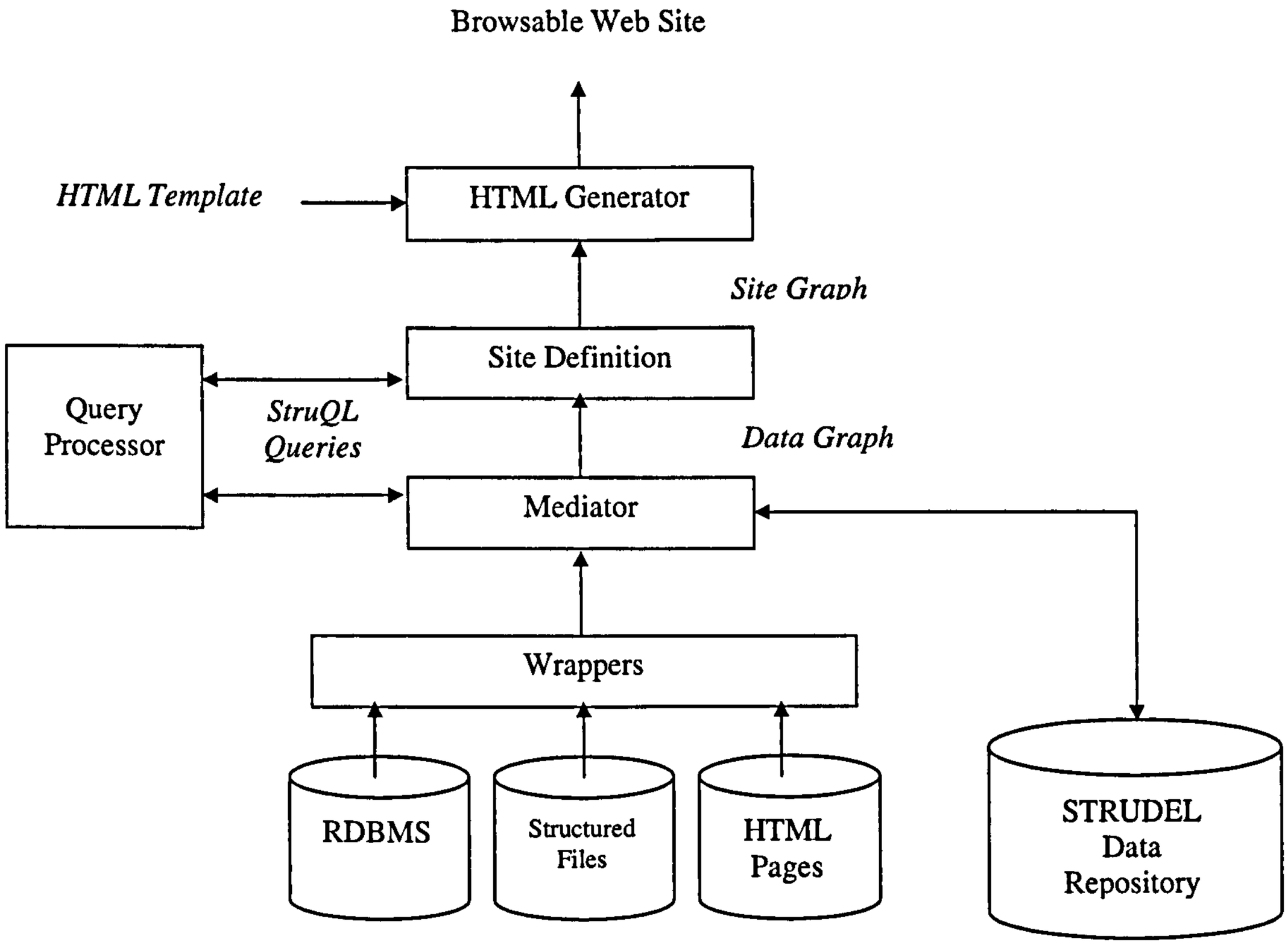


Figure 2.2 Strudel Architecture (Fernandez et al. 1998)

STRUDEL first integrates data from any number of heterogeneous data sources into a semi-structured repository. Then it applies a site-definition query to declaratively define the Web site's structure with the result called site graph which represents both site contents and structure and finally presents the visual presentation in Strudel's HTML-template language (Figure 2.2).

- Data Conversion: Siméon and Cluet (1998) and Cluet et al. (1998) proposed a YAT system for data conversion among heterogeneous data sources based on a middleware model which is named tree with ordered and labelled nodes (similar to a semi-structured Data model). YATL (YAT Language), the conversion language, is declarative, rule-based and features an enhancement pattern-matching customisation mechanism.
- Schema Specification: Buneman et al. (1997) proposed a new schema for semi-structured data by presenting both data and schema as edge-labelled graphs. They then studied the analogy between the graph database and graph schema

and showed how schema can improve the optimisation and decomposition of queries.

- Schema Extraction: Another example is the Data Guides (Goldman and Widom 1997), which is one of the novel features of Lore. They are concise and accurate structural summaries of data stored and are created from the data itself. They are used in browsing data, formulating queries, storing information such as statistics and enabling query optimisation. The paper by Buneman et al. (1997), which was previously mentioned in Schema Specification, can also be considered as a form of schema extraction.

- Optimisations and Formal Aspects: Abiteboul and Vianu (1997) introduced a web model as an infinite semi-structured set of objects. They studied declarative query languages (such as first-order logic, Datalog and Datalog with negation) based on that model.

- Indexing: McHugh and Widom (1999) studied the indexing of semi-structured data in the Lore DBMS and they developed the following indices:

    - Vindex: a value index over atomic values (such as integer, string and real) based on the type coercible which can be built selectively.
    - Tindex: a text index which locates string atomic values containing specific words or groups of words that can be built selectively.
    - Lindex: a link index locates the parent of a specific object.
    - Pindex: a path index for fast access to all objects reachable via given labelled paths.

The above body of research addresses a number of general areas. Each of these areas has potential for valuable research. However, this research focuses on the performance and flexibility of storing and querying semi-structured data in general.

2.2.5 Conclusion and Way Forward

The preceding sections of this chapter have discussed semi-structured data. Its origins, motivations, definition, characteristics, data models and query languages as well as its related research issues.

An understanding of semi-structured data is a necessary preliminary to the research into issues related to the XML database and to be explored in the rest of this thesis, since XML implements a specialised form of semi-structured data. Accordingly, the basics of XML are discussed in the following section. Section 2.4 compares and contrasts semi-structured data and XML.

2.3 XML

The Extensible Markup Language (XML) has become, in a very short period, the base for data presentation and data exchange between homogenous and heterogeneous applications on the Internet and Intranets. Moreover, it is now fundamental to emerging applications both in the academic and business domains such as e-learning and e-commerce. For example, BizTalk (a business process management server software) is based on XML technology and is used in more than 6000 organizations world wide to enable them to automate the exchange of information and integrate business processes (BizTalk web site).

This following section 2.3.1 introduces XML. Then section 2.3.2 describes its data model followed by a discussion of its query languages in section 2.3.3. Finally, section 2.3.4 concludes this discussion about XML.

2.3.1 Introduction

In this section, XML, another key technology to this research is presented. Firstly, it discusses the origins of XML, followed by how it can be categorised, its strengths and weaknesses and finally its related technologies.

2.3.1.1 XML Origins

The Extensible Markup Language (XML) was adopted by the World Wide Web Consortium (W3C: Online) as a technology used for storing and exchanging structured documents and data on the Web (XML: online). The first working draft of XML was published by the W3C in November 1996 (XML: 1996 Online). The first XML recommendations were published in February 1998 (XML: 1998 Online). The most recent XML recommendations (fourth edition) were published in August 2006 and edited September 2006 (XML: 2006 Online).

XML's roots belong to the document community not to the database community (Widom 1999, Vianu 2003), since XML is a simplified subset version of the Standard Generalized Markup Language (SGML 8879: Online). SGML is the widely used international standard for text processing defined by the International Organization for Standardization (ISO: online). A comparison between XML and SGML is beyond the scope of this research but can be found online in Clark (1997). XML documents can be easily sent, received, and processed on the Web in a similar way to HTML documents (XML: online).

Despite the fact that XML's roots belong to the document community, there is a clear analogy between XML and database theories in general and semi-structured data in particular. Section 2.4 discusses this relationship in more detail.

XML was introduced to complement HTML rather than to replace it. XML's goal is to describe data, in contrast with HTML, where the goal is to format data. W3C introduced XHTML (XHTML: 2002 online) as the new generation of HTML as a reformulation of HTML 4.0 in XML 1.0 format. The basic differences between XML and HTML are (Abiteboul et al. 1999):

- New tags can be defined in XML documents, whereas in HTML all tags are pre-defined within the language.
- XML structure can be nested to arbitrary depth, whereas HTML does not have this flexibility.
- A description of XML grammar can be stored within the document. This contrasts with HTML, since HTML is used for data presentation, there is no need for a definition of the data's grammar.

After presenting XML's origins and its relation with HTML in this section, the following section presents the different categories of XML documents.

2.3.1.2 XML Document Categories

As defined by W3Schools (W3Schools web site: online), "An XML document contains structured or semi-structured data in verbose user-defined tags presented in a hierarchical way (tree-like structure)". XML documents can be categorized into two main types (Bourret 2005), data-centric and document-centric. The former uses XML as a method of data transport and is mainly designed for machine representation of highly

structured data, such as product lists and inventories. The second category uses XML to store documents with less regular structure and is mainly designed for human consumption, such as book contents, emails and advertisements.

The following table summarizes the differences between a data-centric and a document-centric XML document (Kim et al. 2002)

Document- Centric XML	*Data-Centric XML*
Irregular and un-structure content	Structured content
Large amount of mixed content	Little or more probably no mixed content
Order is significant	Order is insignificant
Human consumption	Machine consumption

Table 2.1 Document-Centric vs. Data-Centric XML (Kim et al 2002)

XML documents can also be categorised as a hybrid of these two types. For example, a collection of XML documents that contain data about products may contain some highly structured data such as the price and a product's name combined with semi-structured data such as a product's specifications. Section 3.3 discusses in more detail the categorisation of XML documents according to their degree of structuredness.

2.3.1.3 XML Strengths and Weaknesses

Since the first XML recommendation in 1998 (online), XML has become a focus for research in both the document and the database research communities. The widely acknowledged strengths of XML include:

- Its simple format: since it is a plain text format.
- The separation of the data from how this data is formatted: Users can display XML data in any electronic device such as computers, mobile phones and personal digital assistants (PDA) using a stylesheet language such as XSLT (XSLT: 1999 online).
- Modelling capability: it can model highly-, semi- and un-structured data.
- Platform independence: It is not bound to a specific platform. It is an open standard technology that uses Unicode in its implementation.

- Extensibility: the keyword for XML. It allows users to add their own tags to describe the data.
- Human readability as well as machine readability.
- Processing Instruction: XML may contain extra processing instructions using the PI element.
- The large investment in XML applications that already exists nowadays.

These strengths make XML appropriate as a way to store and exchange data on the web. However, there are widely acknowledged weakness of XML listed below:

- The relational database model was built upon a strong mathematical and theoretical foundation; while in contrast, XML as a subset of SGML does not have the same theoretical foundation.
- Although XML by itself is a simple plain text format, it is surrounded by a huge amount of different technologies that can make it too complex to work with and benefit from all its advantages. For example: DTD, XML Schema, XPath, XQuery, XSLT, Xpointer, DOM, SAX, XForms, XLink. etc. are different technologies related to XML. The related technologies to this research are to be discussed later in this chapter.
- XML is a verbose text format and not a binary format; this makes its storage (if it is stored naturally as a text file format) and/or transmission less efficient (although neither storage nor bandwidth is such a significant problem nowadays).

Despite these weaknesses, XML's strengths make it very widely accepted and used. In 2003, it was anticipated by the database community and document community that a huge number of web sites would store their data as XML in the future, beside XML is emerging as the *lingua franca* of the web (Vianu 2003). Various organizations (such as the Cooperative Association for Internet Data Analysis (www.caida.org) and Internet Domain Survey (www.isc.org/ds/) statistically analyze the growth of the web from the network point of view (that is to say from the number of servers connected to the internet) but not based on the web contents and type of files used. Therefore it is hard to give an estimate of the XML content of the web. On the other hand, other researches such as Barbosa et al. (2006) statistically analysed a sample of about 200,000 XML

documents from two broad categories: a) macro-level describing the XML web and its contents and b) document level describing structural properties of typical XML documents. This study showed that there is an increase of the use of the XML as a data storage media on the web.

2.3.1.4 XML Technologies

XML is a crowded field with many different technologies related to it. Some of the technologies are discussed in section 2.3.2.2 such as XML Info Set, XPath 1.0, DOM and XQuery and XPath data model. XQuery is presented in detail in section 2.3.3.2. In this section, other XML technologies are briefly discussed to show their purpose, status and description.

- SAX (online): stands for Simple API for XML. It was originally released in 1998 as a common event-based API for parsing XML documents. While DOM (online) creates a tree of nodes into the memory to access data in XML documents, SAX uses a different approach. It notifies the application by a stream of parsing events. Although SAX accesses the data sequentially, it is useful when the document is relatively big, since there is no need to load the entire document into memory.

- DTD (online): stands for Document Type Definition. Its purpose is to define the structure of a collection of XML documents. It can be stored internally inside the XML document itself or externally with a reference from within the document. It only supports one data type (string). It was inherited from SGML.

- XML Schema (online): as DTD, its purpose is to define the structure of a collection of XML documents. It was designed to overcome the shortcomings of DTD. It is anticipated that it will replace DTD in the future since it is much richer and covers more issues than DTD. For example, it supports different data types and XML Name Space (online). Although this was not the case in a recent study by Barbosa et al. (2006), they showed in a sample of about 200,000 XML documents that 48% of the documents referenced DTD while only 0.09% of the documents referenced XML Schema. This may be likely to be due to the fact that the sample of this experiment was gathered a short time after the release of the XML Schema.

There are many more technologies related to XML such as XForms (online), XSLT (online), XLink (online), XPointer (online), RDF (online), SOAP (online) and others. They are not discussed here since they are not related directly to this line of research.

After introducing XML in this section, the following section describes its data model followed by its query languages.

2.3.2 XML Data Model

XML is a document markup format and not a data model (Widom 1999, Vianu 2003) as it lacks the basic definitions required of a data model such as an abstract definition of its components. However, in order to establish efficient XML-encoded data management and querying technologies, there is a need to map XML-encoded information into a true data model. This is the first step toward query transformation and optimisation.

In order to make the following discussion tangible, Figure 2.3 shows an example of an XML document.

```
<Publication>
    <Paper>
        <Author>Migual Nunes</Author>
        <Author><First>Maggie</First><Last>McPherson</Last></Author >
        <Title>The Management of Information</Title>
        <Title>The Institute for the Management of Information Systems
        Journal</Title>
        <Year>2002</Year>
</Paper>
<Paper>
        <Author>Peter Buneman</Author>
        <Title>Semi-structured Data</Title>
        <Journal> Tutorial in Proceedings of the 16th ACM Symposium on
        Principles of Database Systems</Journal >
        <Year>1997</Year>
</Paper>
...
<Proceeding>
```

```
<Author> Barry Eaglestone</Author>
<Author> Siobhan North</Author>
<Author> Alexandra Poulovassilis</Author>
<Title> Advances in Databases, 19th British National Conference on
Databases, BNCOD 19</Title>
<Publisher> Springer</ Publisher >
<Year>2002</Year>
</ Proceeding >
</Publication>
```

Figure 2.3 XML as Node Labelled Directed Graph

From the above example, it can be seen that the basic building block of any XML document is an XML element (such as paper element). Any element is bounded by matching tags. It may contain raw data, other element(s) or a mixture of both (an author element is contained within the paper element). An XML document should have one root element at the top of the document hierarchy (in the above example, it is the publication). Since XML is a document format, the order of the elements is important and therefore it has to be considered when designing the XML data model.

XML attributes can be associated with elements in order to give more information about the element. Attributes must be strings. There is no clear rule when to use a sub-element and when to use an attribute. The following XML data has the same semantics.

```
<Proceeding year ="2000">
        ...
</ Proceeding >
```

And

```
<Proceeding>
        ...
        <Year>2002</Year>
</ Proceeding >
```

Attributes can not be repeated inside the same element and their order inside the element is not relevant.

As semi-structured data, XML can be modelled as a directed graph, in which the nodes represent XML elements. Attached to each node is the element data. The edges

represent the hierarchical relationship between different XML elements. For example, in figure 2.4, *Paper* element is represented as a node in the graph and its relation with the *Author* element is represented by an edge. The main difference between this representation and the semi-structured data representation is that XML is a "Node Labelled Graph", where XML denotes a graph with labels on nodes while semi-structured data is an "Edge Labelled Graph", since it denotes a graph with labels on edges (Abiteboul et al. 2001).

To describe this difference in a formal way, a directed graph G can be defined as a set of N as nodes and a set of E as Edges. $G = (N,E)$. Where each edge E is a pair of nodes (x,y) where x represents the *source* while y represents the *target*.

In an *edge-labelled* graph $G = (N,E,F_E)$, F_E is an *edge labelling function* that maps each edge to a label while in a *node-labelled* graph $G = (N,E,F_N)$, F_N is a *node labelling function* that maps each node to a label. (Wood online)

It is an easy process to convert between the two models especially in the case of a tree although it is more complex in the case of graph data representation (to resolve the issue of reference between elements) (Abiteboul et al. 1999). The following figure (2.2) represents the previous example (Figure 2.1) as a node labelled graph.

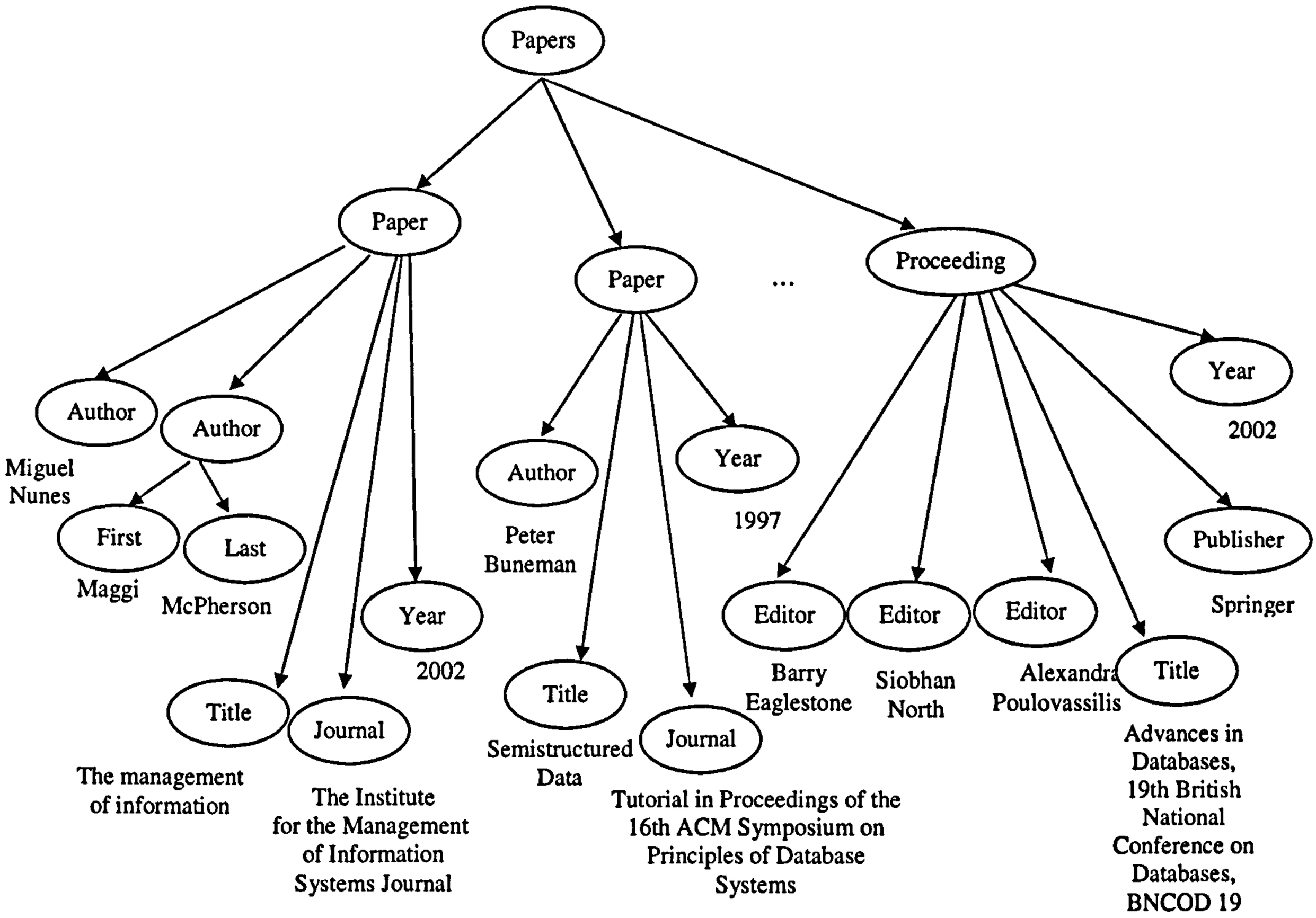


Figure 2.4 XML as Node Labelled Directed Graph

This is not the only difference between XML and semi-structured data. For example, XML is ordered while a semi-structured data model is not ordered and there is no analogy to XML attributes in semi-structured data models. These differences are discussed in more detail in section 2.4.

Defining a formal data model for XML is the subject of the following section followed by the proposed data models by the W3C for XML documents.

2.3.2.1 Formal XML Data Model

Defining a formal data model for XML provides a solid basis for a rigorous investigation of query optimisation and transformation for XML databases. This section describes the data models proposed by various researchers to formally define XML.

XML is a document format and not a data model (Widom 1999, Vianu 2003). In particular, the XML definitions (XML [Online]) lack operations that can manipulate XML data, structures and queries. However, definition of a formal data model for XML is a popular research topic, and a number of studies into XML databases propose formal data models (Lore (McHugh and Widom 1999), Beech et al. 1999, SAL (Beeri and Tzaban 1999), TAX (Jagadish et al. 2001), XAL (Frasincar et al. 2002), Kim et al. 2002, Paparizos et al. 2002, W3C XQuery data model (Online 2003), Novak, L. and Zamulin 2005 and 2006 and Paparizos and Jagadish (2006)).

XML data models can be briefly categorized into two main types:

- Relational-based data models (Codd 1970). In this type of model, the data structure used is the relation (table). The data is modelled using tables and relations between tables and within them.

- Graph-based data model (for example Beech et al. 1999). The data structure used in this type of model is the graph, consisting of nodes (vertices) and edges that represent the relation between different nodes. The graph could be cyclic (for example XAL: (Frasincar et al. 2002)) or acyclic (tree) (for example TAX: A tree algebra for XML (Jagadish et al. 2001)), depending on how the reference edge (IDREF and IDREFS) in an XML document is represented. An example of a model that works both ways is Lore (McHugh and Widom 1999). There are a number of proposals defining graph-based formal models such as "A formal data model and algebra for XML" (Beech et al. 1999), Lore (McHugh and Widom 1999), W3C XQuery data model (Online 2003), XAL (Frasincar et al. 2002), SAL (Beeri and Tzaban 1999), A data model and algebra for document-centric XML (Kim et al. 2002), a Physical Algebra for XML (Paparizos et al. 2002).

Both approaches are relevant to this research. While the relational-based approach provides a way of applying relational query optimisation techniques, the graph-based model provides the most direct and natural formalism for XML documents. It directly describes the hierarchical composition of an XML document, as nested tagged elements, and links within and between documents. Each one of these models has some advantages and disadvantages regarding the modelling of XML data. These can be summarized as follows:

- Relational Model: It is too rigid to easily contain semi-structured data. It has well-defined theories regarding query optimisation and operations. Also, there are some major differences. For example, the XML data is ordered while the relation is a set, and therefore unordered. The handling of duplicate as well as non-complete data is another major difference between the two.

- Graph-based Models: graphs are a natural representation of XML documents. Since XML is based upon nested tagged nodes (represented by graph vertices or nodes) that contain the data in elements or attributes and inter-document links with intra-document links represented by the graph edges. So, the graph-based data model is most commonly used for XML data since it can represent its complex structure. However, the query optimisation techniques are still emerging and there is no solid mature technique compared to the relational approach.

So, based on that, the Graph-based model is the appropriate way to define a data model that can represent XML documents. One good example of these data models is 'A formal data model and algebra for XML' (Beech et al. 1999) for the following reasons:

- It is a very well defined and simple data model.
- It can represent both data-centric and document-centric collections of XML documents.
- It is based on XML Infoset (Online 2001) which is a W3C recommendation that defines a set of specifications needed to refer to the information in an XML document. XQuery (the de facto standard query language for XML) is based on XML Infoset and is described in more detail in section 2.3.3.3.
- It defines a number of operations to deal with collections of XML documents.
- Although it does not define optimisation strategies or physical algebra operations, it allows scope to develop such optimisation techniques.
- A number of formal models such as XAL (Frasincar et al. 2002) were inspired by it.

The next section describes in more detail this data model.

2.3.2.1.1 Detailed Description of the Formal XML Data Model

The formal XML data model by Beech et al. (1999) can be described as follows:

- A node-labelled directed ordered graph
- *V*: The graph vertices (or nodes) are either an element or a data value. Each vertex must have a parent (another element vertex) with the exception of the root (as a special case which has a fictitious root vertex). Each vertex has a unique, immutable and system generated identifier. This is different to the ID which may be used for internal referencing between XML elements.
- The graph directed edges are one of three types; E, A or R:
    - *E*: a set of directed elements on containment edges. It relates a parent element to a child which could be another element where the name of the edge is the generic identifier of the child's name or a data value with a special name ~data, a comment with a special name ~comment or a processing instruction with a special name ~PI
    - *A*: a set of directed attribute edges. It relates an element to its attribute data value.
    - *R*: a set of referenced edges. It relates an element to another referenced element via IDREFs or IDREFSs or XLinks, URIs or other reference mechanisms.
- *O*: represents a set of ordering relations of child elements within a parent element. It represents the total order among all edges; while it does not represent order among different types nor elements with a different parent vertex. This order is defined for the three different types of edges as follows:
    - *E*: the order of the children as they appear within the parent element
    - *A*: not defined, since the XML attributes are un-ordered
    - *R*: the order as they were in the document in the case of IDREFS
- Vertices have two basic properties:
    - *value*: returns the system generated identifier for this element, and in the case of a value vertex it returns its value

- *type*: which is either the element in the case of an element vertex or the data type of the value in the case of a value vertex

- Element vertices derived properties (based on edge and order information)
  - *gi*: name of this vertex (namespace qualified if relevant)
  - *parent*: vertex parent
  - *referredby*: set of vertices that reference this vertex through a reference edge
  - *childelements*: set of all element containment edges from this vertex
  - *attributes*: set of all attribute edges from this vertex
  - *references*: set of all reference edges from this vertex
- *E*: has the following basic properties
  - *Parent*: returns the 'from vertex'
  - *Child*: returns the 'to vertex'
  - *Name*: returns the name of the edge
  - *Type*: returns E
- *A*: has the following basic properties
  - *Parent*: returns the referring vertex
  - *Name*: returns the name of the edge
  - *Value*: the attribute value
  - *Type*: always A
- *R*: has the following basic properties
  - *Parent*: returns the 'from vertex'
  - *Child*: returns the 'to vertex'
  - *Type*: Rx where R indicate that this is a reference and X indicates the kind of that (XLink, URI, ...)
  - *Refedge*: the set of attributes or element edges that provide the reference information

- E, *A*, *R* has the following derived properties
    - *Next*: returns the following edge
    - *Previous*: returns the previous edge
- *O*: has the following basic properties
    - *e*: returns the current edge
    - *successor*: the successor of the current edge

So, briefly based on the previous definitions, a formal data model can be defined as follows:

- A graph $G = (V, E, A, R, O)$ represents the data model for XML elements, where
    - $V = V_{element} \cup V_{int} \cup V_{string} \cup \ldots$
    - *E*: represents the set of directed element containment edges
    - *A*: represents the set of directed attribute edges
    - *R*: represents the set of directed reference edges
    - *O*: represents the total order between edges of a particular class E, A or R, that connect a parent element to its children

In Appendix A, there are three examples showing how to model an XML document using the above data model. The first example is a data-centric XML document, the second is a document-centric XML document and the last one is a hybrid document. These XML documents were extracted - with some modifications - from the department of Information Studies, University of Sheffield web site (Online).

Following the discussion of the formal data model for XML in this section, section 2.3.3.2 discusses the XML query algebra based on this formal data model. The next section discusses the proposed data models by the W3C for XML.

2.3.2.2 W3C proposed data models for XML

The World Wide Web Consortium (W3C Online) proposed different data models for an XML document for different purposes. These data models are:

- XML Info Set (2nd recommendation online 2004): its purpose is to refer to information stored inside XML documents. Therefore it is used by other

technologies as the base to refer to data inside an XML document. It consists of eleven node types and it does not require a DTD or XML Schema validation.

- XPath 1.0 Data Model (recommendation online 1999): its purpose is to address parts of the XML document. It is used by other specifications such as Xpointer and XSLT. It consists of seven elements and it does not require DTD or XML Schema validation.
- DOM (recommendation level 3 online 2004): stands for document object model. Its purpose is to access and update the structure of a document dynamically. It can be used to model both XML and HTML. It consists of twelve elements and it does not require DTD or XML Schema validation.
- XQuery 1.0 and XPath 2.0 data model (XDM candidate recommendation online 2005). It is the only data model that can be used to model a collection of XML documents. It consists of eight elements and it can use DTD or XML Schema to validate a document if it exists.

Salminen and Topma (2001) summarised these different data models. The summary is presented in the following table.

	XML Info Set	Xpath 1.0	Dom 1.0 Level 2.0	XQuery 1.0 and XPath 2.0
Purpose	To refer to the information stored in the XML document	To address parts in an XML document (it works as a query language but for one document only)	To access and update the contents and the structure of documents dynamically	To define precisely the information contained in the input to an XSLT or XQuery processor.
Status	2nd Edition Recommendation (2004)	Recommendation (1999)	Level 3 Recommendation (2004)	Recommendation (2007)
What is modelled	XML	XML	XML or HTML	Collection of XML or parts

	XML Info Set	Xpath 1.0	Dom 1.0 Level 2.0	XQuery 1.0 and XPath 2.0
No of Nodes	11	7	12	8
Node Type	1. Document 2. Element 3. Attribute 4. Processing Instruction 5. Unexpanded Entity Reference 6. Character 7. Comment 8. The Document Type Declaration 9. Unparsed Entity 10. Notation 11. Namespace	1. Root 2. Element 3. Text 4. Attribute 5. Namespace 6. Processing instruction 7. Comment	1. Document 2. Document Fragment 3. Document Type 4. Entity 5. Notation 6. Element 7. Attr (Attribute) 8. Processing Instruction 9. Comment 10 .Entity Reference 11. CDATA Section 12. Text	1. Document 2. Element 3. Attribute 4. Text 5. Namespace 6. Processing instruction 7. Comment 8. Reference
DTD validity required	No	No	No	XML document can be validated against a DTD if it exists.

Table 2.2 Different XML Data Models Proposed by the W3C Salminen and Topma (2001)

Each of the above models tries to solve a specific problem and to satisfy a specific requirement. Therefore, these models are used in different contexts in the XML world.

They are there to model XML from a technology point of view while the other formal models are used to model XML from a theoretical point of view.

After presenting the formal data models for XML proposed by researchers and those proposed by W3C, the following section discusses query languages for XML.

2.3.3 Query Languages for XML

The analogy between semi-structured data and XML leads to the possibility that the query language discussed in section 2.2.3 for semi-structured data could be suitable for XML as well. There are a number of query languages designed from the start for XML such as XQL (Robie et al. 1998), XML-QL (Deutsch et al. 1998) and XQuery (online). In the following section, XML query requirements in general are discussed followed by a discussion of the XML query algebra. Then XQuery, the query language adopted by the W3C as the *de facto* standard for querying XML is discussed in section 2.3.3.2.

2.3.3.1 XML Query Requirements

There are a number of main requirements for the XML Query Languages, (for example: Maier 1998, XML Query Requirements 2001 (Online), Bonifati and Ceri 2000 and Fernandez et al. 1999). Maier's list (1998) addresses a number of issues, i.e., implementation strategy, the design characteristics of the query language, and its expressiveness. The following list summarises these requirements:

- Orthogonal
    - XML Output: The result of the XML query is an XML document. This also allows compositionality (Abiteboul et al. 1999) as the result of the query can be used as an input for another query and the result must be in the same data model.
- Expressiveness and completeness
    - Query Operations: The Query language must perform the following operations
        - Selection: Selecting a subset of the document or the whole document based on content, structure or attributes.
        - Extraction: Removing particular elements of a document.

- Reduction: Removing selected sub-elements of an element.
- Restructuring: Constructing a new set of element instances to hold queried data.
- Combination: combining more than one element into one.
- Preservation of Order and Association: as XML is an ordered document, the query language must preserve this order in the results
    - Mutually Embedding with XML: XML can contain query statements while a query statement can contain XML elements.
    - Xlink (Online) and Xpointer (Online) Cognizant: a query language should be aware of XLinks and XPointers.
    - Namespace Alias Independence: The query language should not depend on namespace aliases local to an XML document
    - Support for New Data types
    - Suitable for Metadata

- Implementation
    - Server-side Processing: Queries can be executed remotely on the server with no dependence on resources in its creation context for evaluation.
    - Programmatic Manipulation: the query statements can be easily created by software applications rather than by being written by the system users or programmers.
    - XML Representation: the representation of the language is in XML, so there is no need for a special mechanism to store the query statements.

- Schema Features
    - No Schema Required: If the schema is not known, the XML Query must depend on the self-describing feature of the XML Document.
    - Exploit Available Schema: If the schema is known, the Query language must make use of it for error detection

- The Query Language Semantics: The language must have a clear definition to allow efficient processing.
    - Precise Semantics: this is to allow the determination of the result structure, equivalence and containment.
- Compositional Semantics: the query expression should be the same wherever it appears.

However, the above features do not address in detail, the expressiveness of XML query languages, for example, in the way in which the notion of relational completeness does for relational languages. To do so, it is first necessary to establish some formal description of the manipulations that an XML query language should be able to apply to an XML database. Such formalism will have a role equivalent to that of relational algebra for relational databases. It is then possible to evaluate the expressiveness of any XML query language, in terms of those expressions in the formalism that the language can also express.

After describing the general requirements of the XML query language, the following section presents in more detail XML Query algebra.

2.3.3.2 XML Query Algebra

The XML query algebra discussed in this section is based on the formal XML data model (Beech et al. 1999) presented in section 2.3.2.1. Its goal is to operate on a collection of XML documents, allowing the selection of a whole document or a part of a document based on specific criteria, and restructuring the results as a new XML document. It allows the use of joins between XML documents. It deals with XML's graph structure, heterogeneity of types, ownership vs. reference and finally the order of the document. It also allows composability and therefore transformation and optimization. Its operation can be summarized as follow:

- *Navigation Operations*: the 'follow' operation (ϕ) starts with a set of vertices and then follows edges of a given type (E, A, R, or any) and with a given name returns a set of edges. To get a set of vertices it must be composed with a child operation. The 'inverse follow' operation ϕ_{inv} also takes the edge type and name as parameters and a set of vertices, it returns all the edges of that type and name that lead to the specified set of vertices.

- *Selection Operation:* the 'selection' operation (σ) allows the selection of a given collection based on a given criteria, it returns a collection where the criteria is true. Properties of vertices and edges can be used to construct the selection criteria as well as standard comparison operations (=, <>, <, >, <=, >=) and Boolean operations (and, or, not). The selection operation also supports Existential and Universal qualification.

- *Join Operation:* when two documents are queried with a join condition between them, the Cartesian product is calculated. Then, for the true join condition only between the two vertices, a virtual reference edge is created and can be used as a normal reference edge during the evaluation of the query only.

- *Construction Operations:* are used in building a new fragment of a document based on the selection conditions. The "*expose*" operation returns the fragment of a document identified by navigation operations in conjunction with selection operations. The "*return*" operation returns copy of the fragment of a document identified by navigation operations in conjunction with selection operations. The "create edge" and "create vertex" are used to create a new fragment of XML by attaching edges and vertices to the root and recursively attaching edges and vertices to the attached vertices.

- Other Operations:
    - "Σ *sort*" which orders a set of edges
    - "χ *unorder*" which indicates that the order is not important and this can help in the query optimization
    - "μ map" which applies a specified function to a collection of edges or vertices. It does not include the input collection in the result collection (Unlike the kleene star *)
    - "* kleene star" operator is used to indicate the possibility of infinite repetition of a function. It can for example allow the navigation among paths repeatedly until reaching a fixed point.
    - "δ distinct" which eliminates duplicates from a set of vertices or edges.
    - "pick" transforms an element of a collection into a singleton

- "flatten" flattens any collection of any nesting depth into a flat collection
- "γ group by" operator is used to create element and attribute vertices in the result that aggregate or summarize information from a group of similar vertices or edges.

The section presented the basic operations of the XML algebra. The following section briefly discusses Xquery, the *de facto* XML query language standard.

2.3.3.3 XQuery

XQuery (online) is a query language maintained by W3C (online) as the *de facto* XML query language. Its status is a candidate recommendation (November 2005). It is derived from Quilt (Chamberlin et al. 2000) and has a lot of features from other technologies such as XPath 1.0 (online), XQL (Robie et al 1998), XML-QL (Deutsch et al. 1998), SQL (SQL: 2003) and OQL (Cattell et al 2000). It is based on the XML data model in XQuery 1.0 and XPath 2.0 (online). It was designed to query XML data as well as XML documents.

XQuery is a functional language with XPath (online) expressions as its basic building blocks. It is a case sensitive language (as XML). It follows the FLWOR model (inherited from Quilt (Chamberlin et al. 2000)) which can be described as:

- *For:* optional clause, it is used to bind variables to an expression. The variable name starts with $ such as $x in the following example.
- *Let:* optional clause, it is used to bind variables as well. The difference is that *Let* avoids repeating the variable in the result. For example, it can be used to represent an average result which is similar in a way to *Group By* in SQL.
- *Where:* optional clause, it is used to specify the selection criteria.
- *Order by:* optional clause, it used to specify sort-order the results.
- *Return:* it specifies what the result will look like.

An example of an XQuery is:

for $x in doc ("document_name.xml")/Publication/paper

where $x/year > 2000

Order by $x/title

Return $x/title

This query returns the titles (ordered alphabetically) of all papers where its year is greater than 2000.

Although XQuery is not a standard yet, it has a wide support by all the major DBMSs such as Oracle (Krishnaprasad et al. 2005, Murthy et al. 2005), Microsoft SQL Server (Pal et al. 2005 and 2006, Rys 2005, Rys et al. 2005 and Lacoude 2006), IBM DB2 (Beyer et al. 2005 and Özcan et al. 2006) and Sybase (Singh et al. 2005). It is anticipated that it will be the XML query language for the future. XQuery is further discussed in chapter five since it is used in the evaluation of the proposed model.

2.3.4 Conclusion and Way Forward

Throughout section 2.3, XML origins, strengths and weaknesses, data models and query languages were discussed. This follows the discussion of semi-structured data in section 2.2. Both of them are key to this research. So, the next section compares and contrasts semi-structured data and XML in more depth.

2.4 Semi-Structured Data and XML

Having introduced the OEM as a semi-structured data model (see 2.2.2.1) and XML (see 2.3), this section now presents a comparative analysis to establish where commonalities exist between these and conventional data models (relational and object-oriented data models). The aim is to show that despite the differences, XML and OEM are more related than XML and traditional database data models. The following comparison is based on the analysis by Abiteboul (1997) and Abiteboul et al. (1999)

- *Natural*: XML is a document markup format; it is not a data model (see 2.3.1.1). While OEM is a data model (see 2.2.2.1).
- *Origin*: The XML is a subset version of the Standard Generalized Markup Language (SGML [Online]). A comparison between XML and SGML can be found online in Clark (1997). XML documents can be easily sent, received, and processed on the Web in a similar way to HTML documents (XML: online). This means that the roots of the XML belong to the document community. Consequently, it differs from OEM, which was originally inspired by the database community to be used for data integration between heterogeneous data

sources in the TSIMMIS project, Stanford University (Chawathe et al. 1994, Garcia-Molina et al. 1995, 1995 and Hammer et al. 1997, OEM (Papakonstantinou et al. 1995)).

- *Schema*: XML may or may not have associated schema. Consequently, there are two levels of XML validation. A 'Well formatted' XML document must be syntactically correct, for example, the tags must be nested correctly. The stronger notion of correctness is that of 'validity'. A 'Valid XML' is a document that is well formatted and conforms to an explicit collection schema, for example, in the form of a DTD or XML Schema. In comparison, the semi-structured data's schema is discovered from the data itself (self-describing), i.e., the collection schema is implicit such as Data Guide of Lore (Goldman and Widom 1997). Where it discovers the schema from the semi-structured data. In both cases the schema may be ignored for browsing purposes. This contrasts with both the relational and object-oriented data models, where the schema must be fixed, and fully defined prior to data entry. Consequently, XML and OEM are more flexible than traditional data models.

- *Structure:* both XML and OEM can be used to represent data with irregular, implicit and partial structures. The most natural way to model XML-encoded data is as a node-edge graph whereas the OEM data model is most naturally represented as a labelled-edge graph (section 2.3.2). This difference is minor compared with the regular, explicit and highly structured nature of relational and object-oriented data models.

- *Order:* a major difference between XML as a document format and OEM and the relational model is that XML preserves the order of data because it is a document format. In contrast the relational model is based on the set theory and by definition a set is not ordered. The object-oriented model re-introduces the order of the data into its model by using other data types such as list and arrays.

- *Textual Data:* XML is a text document format, and therefore it provides only text data types while a semi-structured model allows for non-textual data types. To represent a binary data in an XML file, it has to be linked as an external entity.

- *Mixing Elements and Text:* this is purely a document feature in the XML documents whereby inside a text there may be another element. For example <address> West Street <city> Sheffield </city> </address>. There is no analogy to these features in any of the other database data models.

- *Constructs:* another feature of the XML document is that it may contain constructs, such as comments and processing instructions (PI) that give some instruction regarding the XML document to the receiving application. The first line of an XML document is given as an example (<?xml version="1.0" encoding="ISO-8859-1"?>). Another example is a CDATA element; It is ignored by the parser and therefore it can contain any special character inside it such as ("<", ">"). There is no analogy for these constructs in OEM or the relational data model. For the object-oriented data model, some analogy exists by using of the object's methods (compared with processing instructions).

- *Reference:* XML elements can be referenced by defining the "Id" attribute with an element and then using IDRef and IDRefs in another element to reference this element. OEM has an analogy by creating an edge from one node to another node. The foreign key in the relational model and OID and relationship in an object-oriented model can be seen as some sort of referencing although the reference in XML and OEM is much richer than in database data models.

The following table summarises the above comparison between XML, OEM, relational and object-oriented Data models.

Feature	**XML**	**OEM**	**Relational**	**Object-Oriented**
Origin	Document Community	Database Community	Database Community	Database Community
Schema	Optional	Post Data	Fixed, defined prior to data	Fixed, defined prior to data
	May be ignored	May be ignored		

Feature	XML	OEM	Relational	Object-Oriented
Structure	Irregular, Implicit, may be partial	Irregular, Implicit, may be partial	Regular, Explicit	Regular, Explicit
	Node-Edge Graph	Labelled-Edge Graph	Tables	Objects
Order	Ordered	Not Ordered	Not Ordered	Using lists and Arrays for Order
Mixing Elements and Text	Available	Not Available	Not Available	Not Available
Constructs	Comments, Processing Instruction, Start with an optional construct (<?xml version ="1.0"?), CDATA, Entities (&lt = <)	No Analogy	No Analogy	Using methods analogy with PI
Reference	XML Reference (IDRef, IDRefs)	No Analogy	Foreign Keys	OID and Relationships
Nesting	Available	Available	N/A	Available

Table 2.3 Comparing XML, OEM, Relational and Object-Oriented Data Models

The above comparison shows that XML has a more complex data structure and is more naturally related to the OEM model rather than conventional database data models. Consequently, to store XML data into a relational database, there is a need to shred the data into smaller parts or to store the whole document as a CLOB data field. The next section and section 3.5 discuss the different storage models proposed by the research to store XML data.

2.5 XML Storage Strategies

The XML storage strategy is a core subject to this research, as the research is centred on achieving a query processing performance enhancement for a hybrid class of XML documents (which is explored thoroughly in chapter 3). XML Storage strategies can fall broadly into three main categories as follows:

- Use of the file system to store XML documents in their native text file format (Abiteboul et al 1993, Milo and Suciu 1999, Rizzolo and Mendelzon 2001, Cooper et al. 2001, Tian et al. 2002 and Madria et al. 2007).

- Use of relational database management systems: They store XML data in relational or object-relational databases, and thus utilise established relational technology such as query optimisation (Shanmugasundaram et al. 1999 and 2001,Shimura et al. 1999, Florescu and Kossmann 1999, Schmidst et al. 2000, Klettke and Mayer 2001, Yoshikawa and Amagasa 2001, Kudrass 2002, Kudrass and Conrad 2002, Runapongsa and Patel 2002, Tian et al. 2002, Amer-Yahia and Srivastava 2002, Bohannon et al. 2002, Kuckelberg and Krieger 2003, Han et al. 2003, Harding et al. 2003, Leonov and Khusnutdinov 2004, Pal et al. 2004, Lu et al. 2006, Na and Lee 2005, Balmin and Papakonstantinou 2005, Chaudhuri et al. 2005, Qin et al. 2005, Pardede et al. 2005, 2006 and Pal et al. 2006)

- Use of novel storage structures: such as object-oriented databases (Christophides et al 1994, Chung et al. 2001), native XML databases (as in Salminen and Tompa 2001, Information Manager (Interleaf: online), Astoria (Astoria Software: online), Timber (Paparizos et al 2003), Tamino (Schoning and Wasch 2000 and software age: online), Teratext DBS (Teratext: online)) or semi-structured databases (as in Lore (McHugh et al 1997 and McHugh and Widom 1999, Goldman et al 1999)), storing XML semantics (Pasila 2002) or using the vectorizing technique (Buneman 2005).

The scope of this study is explicitly focused on the relational model approach, as presented in chapter one. This is because the relational database is the most widely database technology used nowadays and will probably remain in a dominant position for the foreseeable future. Therefore in this section the first and the third approaches are discussed in sections 2.5.1 and 2.5.2 respectively, while the relational approach is

discussed later in depth in section 3.5. Added to that, there are two specific storage models (Ozone (Lahiri et al. 1999) and STORED (Semi-structured TO Relational Data Deutsch et al. 1999) which are very much related to this study and are discussed in detail in section 3.5 too.

2.5.1 Using the File System

In the first approach, the XML documents are stored in a file system in their native text file format (Tian et al. 2002). This is a similar approach to the one proposed by Abiteboul et al (1993) to store and query SGML documents using the file system. It can be used with any XML document regardless of its level of structuredness, which makes it suitable for un-structured, semi-structured, highly-structured or hybrid documents. It is the simplest and easiest approach to implement since it does not require any processing of XML documents prior to their storage, and hence does not need any middleware or DMBS to manage the document collections. To query any XML document, the document has to be parsed and loaded into the memory (for example, as a DOM tree) and then standard techniques such as Xpath (Online) and XQuery (Online, Fernandez et al. 1999) can be used to query it. However, as found by Tian et al. (2002), this approach shows serious limitations arising from the need to load the whole XML document into the memory. This results in an increase of the query response time in proportion to any increase in the size of the document.

This performance limitation can be reduced by using external indices to address part of the XML document directly. For example, indices have been used by Rizzolo and Mendelzon (2001) to improve the query performance by summarizing path information. They developed ToXin as an indexing scheme for XML data. Their storage model consists of two different types of structures: a path index that summarizes all paths in the database and can be used for both forward and backward navigation starting from any node, and a value index that supports predicates over values. Another system in the same family is that presented by Madria et al. (2007) who use indices for regular path expressions to efficiently process Xpath queries. Other examples of the use of indices are T-indexes (Milo and Suciu 1999) and that proposed by Cooper et al. (2001). As in any storage structure, there is a performance penalty in managing such indices when inserting and deleting data to and from the XML documents. The other main disadvantage of this approach is the lack of the DBMS features such as concurrency

control, redundancy control, security, access and transaction control etc. Nevertheless, it can be used efficiently in some scenarios, such as a collection of small XML documents, which are not frequently updated. In such cases, the above disadvantages are minimised.

2.5.2 Using Novel Storage Structure

In this section, novel approaches to storing and querying XML are described. These include using object-oriented databases, native XML databases, semi-structured databases such as Lore (McHugh et al 1997 and McHugh and Widom 1999, Goldman et al 1999), storing XML semantics (Pasila 2002) or using the vectorizing technique (Buneman et al. 2005).

2.5.2.1 Object-Oriented Databases

Using an object-oriented database is in a sense similar to using a relational database to store an XML document (Chung and Kim 2003). It can be achieved either by mapping the XML graph to the object-oriented structure or by mapping XML data itself to an object-oriented equivalent schema. The object-oriented paradigm provides more modelling power than the relational data model, for example by using lists, arrays and union types, and has the potential to capture behavioural semantics. Christophides et al (1994) proposed a natural mapping from SGML documents to object-oriented databases. They proposed a formal extension to query languages to deal with SGML documents. Chung et al. (2001) extended the same approach by using inheritance to extract the equivalent object-oriented schema.

The problems with using object-oriented rather than relational database generally are (Leavitt 2000)

- *Object-Relational Databases*: The development of object-relational databases reduces the gap between relational and object-oriented databases in storing complex and multimedia data. This means that it is no longer an advantage for object-oriented databases to deal with this type of data over relational databases.
- *Performance*: Although object-oriented databases can store data objects as units, and therefore retrieve them faster than the relational database which has to break them into smaller parts, the optimization techniques used in relational databases give them an edge over the object-oriented databases.

- *Standardization*: A long established mathematical standard is the basis for relational databases which is not the case for object-oriented databases.

Thus, despite the advantage of better modelling using the object-oriented paradigm, the above problems create a barrier to the widespread use of object-oriented databases in general and in dealing with XML data in particular.

2.5.2.2 Native XML Databases

The aim of the native XML databases (NXD) is to provide a means to define, create, store, validate, manipulate, publish, and retrieve XML documents acting as its native storage unit (Salminen and Tompa 2001 and Fiebig et al. 2002). They can be categorized into two main approaches, document management systems (Information Manager (Interleaf: online) and Astoria (Astoria Software: online)) and data management systems (Timber (Paparizos et al 2003), Tamino (Schoning and Wasch 2000 and software age: online), Teratext DBS (Teratext: online)).

XML:DB Initiative (Online) defined Native XML Databases (NXD) as follows:

- NXDs define a (logical) model for an XML document - as opposed to the data in that document - and store and retrieve documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and the document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.
- A NXD has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.
- Is not necessary to have any particular underlying physical storage model. That is to say, the NXD can be built on top of a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed and/or compressed files. This is hidden from the database user.

As Staken (2001) summarises; native XML databases mainly store XML documents and their components, the basic unit to deal with the database is the XML document, NXD may not be a standalone database, it can be built over any other DBMS and it does not necessary store XML in a native text format.

The advantages of using NXD are:

- There is no need for mapping the XML data to a different data model as the data is mapped to a tree data model
- As the data will not be dispersed into small parts and stored in different disk parts, this will probably lead to a better data retrieval performance.

The disadvantage of this approach is its lack of a single and common well-defined data model. This leads to the absence of a solid base for formally defined query operations and therefore optimisation. As these databases evolve, and the database management kernel becomes more optimised, better performance can be expected to be achieved in the foreseeable future.

2.5.2.3 Lore

A Lightweight Object REpository for Semistructured Data (Lore) (McHugh et al 1997 and McHugh and Widom 1999) started as a complete semi-structured database management system in Stanford University then it was migrated to host XML data (Goldman et al 1999). Lore was built from scratch, so all the database management system functions (such as query language, multiple indexing techniques, a cost-based query optimizer, multi-user support, logging, and recovery) had to be designed and implemented. Loral (Abiteboul et al. 1997) is the query language for Lore (described in section 2.3.2.1). Lore advantages include its novel technologies such as DataGuides (Goldman and Widom 1997), indexing techniques (Vindex, Tindex, Lindex and Pindex, as discussed in section 2.2.4), management of external data, and proximity search. The project produced an academic prototype for a semi-structured database management system and was completed in year 2000. However, Lore remained as an academic prototype and did not developed more toward becoming a database management system product. This could be considered as its disadvantage.

2.5.2.4 Store XML Semantics

In this approach Pasila (2002) proposed the ERX model (Entity Relationship for XML). This model stores the semantics of XML rather than modelling the XML document to a relational model or to a tree structure. It describes concepts and complex relationships of the data. However, there is a need for the complete DTD for the XML document to

be converted either manually or automatically (Pasila 2003) to an ERX model which in some cases is not available.

2.5.2.5 Vectorizing Approach

The final approach uses Vectorizing storage for XML (Buneman et al. 2005): The idea behind this technique is to decompose an XML document into a set of vectors that contain the data values and a compressed skeleton that describes the structure. Storing data by column (as opposed to storing it by row) is an old technique (Batory 1979) which is especially useful when using a small number of columns from the database. This technique is similar in a way to the structured mapping approach and shares its strengths and weaknesses (instead of storing the XML document by rows, it will be stored by columns). These weaknesses are mainly the complexity of the generated schema, the inherent ambiguities and contentions that must be resolved and the inflexibility of the resulting relational schema, especially in situations where the structure of the XML document collection is volatile.

In this section, different storage models used to deal with the problem of storage of XML-documents were presented. Research in XML Benchmarking (XMach-1 (Böhme and Rahm 2001, 2002), XMark (Schmidt et al. 2002), X007 (Brassan et al. 2002), and XBench (Yao et al. 2002, 2003, 2004) and The Michigan Benchmark (Runapongsa et al. 2002a, 2006) shows that there is no absolute correct way to store XML. Each one of these models has its advantages and disadvantages and its successful and un-successful scenarios based on the type of the data and the query scenarios used. XML Benchmarking is further discussed in Chapter 4.

2.6 Conclusion

In this chapter, the base for this research has been presented. First, it introduced semi-structured data, its origin, definition, characteristics, data models and query languages. Then XML was presented also with its origin, strengths and weaknesses, data models, and query languages. Then semi-structured and XML were compared and contrasted in section 2.4. Finally, in section 2.5, some of the different storage models for XML were discussed in more depth.

The discussion in the next chapter builds upon this by firstly presenting the research motivations and formulating the research hypothesis. Then, a categorization of XML

documents according to their degree of structuredness is discussed. The research narrows down to a class of hybrid-structured XML document defined as containing partially-structured data. How relational databases can be utilized to store and query this class of documents is also explored. This is followed by a presentation of the different storage models based on the relational model to store and query XML data as well as related storage models proposed by the literature to store and query partially-structured data. This leads to an elaboration of the hypothesis, in which a storage structure is proposed, which has the potential to realise performance enhancements for partially-structured data. Chapter three concludes by presenting the storage model proposed by this research to deal with partially-structured XML documents. Chapter four presents the design of the experiments to evaluate the performance of this proposed model

Chapter 3 Partially Structured XML

3.1 Introduction

Chapter two presented a review of XML databases in general to establish the context and motivation of this research. It defined semi-structured data and XML, their data models and query languages. It compared and contrasted them and then it reviewed the different storage models proposed for collections of XML documents in general. This chapter builds upon the review by firstly presenting the research motivation for the study in this thesis and formulating the hypothesis it addresses in section 3.2. The hypothesis is a proposition that performance enhancements can be achieved by exploiting pre-knowledge of consistent structure within XML documents and/or across XML document collections. Section 3.3 therefore discusses a categorization of XML documents according to their degree of structuredness, so as to analyse structural properties for which this approach is applicable. Accordingly, section 3.4 defines the class of partially-structured XML data as a hybrid of highly-structured and semi-structured data, shows an example and presents its advantages. Appropriate storage models must be utilised in order to exploit structural pre-knowledge. Accordingly, section 3.5 reviews and discusses storage models for XML data, and analyses their potential for storing and querying partially-structured data. Special attention is given to relational approaches to storing XML data and the related storage models proposed for partially-structured data in this section. The conclusion of this analysis is an elaboration of the hypothesis, in which a hybrid relational and XML storage structure is proposed which has potential to realise performance enhancements for partially-structured data. This storage model is presented in Section 3.6. Finally, section 3.7 summarises the chapter, presents its conclusions and outlines how the issues raised in this chapter are addressed in subsequent chapters.

3.2 Research Motivations and Hypothesis

This section presents the research motivations and hypothesis based on the issues that emerged in the previous chapter, in which the tension between technologies and models for XML document collections and databases were identified. The motivation for this study is presented in section 3.2.1. This is followed by the hypothesis in section 3.2.2.

3.2.1 Motivation

The motivation behind this research is rooted in issues relating to the structuredness of XML document collections (as in section 3.3) and the implications for the query processing performance. This section further discusses the class of XML data which is a hybrid between highly-structured and semi-structured data, referred to as partially structured XML (as in section 3.4). Specifically, the study seeks to exploit the knowledge of the highly-structured part to improve query processing performance. The latter is the focus of this research, documented in the remainder of this thesis.

As discussed in section 2.3.1.3, XML has become a focus for research in both the database and document research communities. This research effort is motivated by strengths of XML, including: its simple format, the separation of the data from how it is formatted, the internationalisation capability, platform independence, extensibility, human as well as machine readability, processing instruction and the large investment in XML applications that already exists nowadays. These strengths make XML appropriate as a way to store and exchange data on the web. However, achieving good XML query processing performance is problematic because the irregular structures inherent in semi-structured data mean that the conventional query optimisation technology cannot be used in a straightforward way (section 2.2.3).

One possible approach to addressing the querying efficiency problem is to exploit the inherent structures of specific XML documents. In developing this approach it is useful therefore to classify XML documents according to their structuredness, as has been done in Barbosa et al. 2001, Yao et al. 2002 and Bourret 2005, where XML documents are classified either as highly-structured, semi-structured or un-structured (see section 3.3). As has been discussed in 2.2.3, querying semi-structured or un-structured data is problematic for query processing and incurs significant overheads, whereas the pre-knowledge of the uniform structures of highly structured data opens the gate to more efficient query processing using well established technologies, such as those developed for relational databases. However, many XML documents are in fact a combination of highly-, semi- and un-structured data. This poses a question; can querying overheads associated with these documents be reduced by exploiting the knowledge of the parts of a document for which the data is highly-structured? In order to address this question, it is necessary to focus on a class of hybrid highly-structured and semi-structured documents, which can be defined as partially-structured (as further explored in section

3.4). In such documents, there is a well defined and prescribed structure in part of the document as well as an ad-hoc semi-structured part.

There are a number of examples of databases that deal with XML data and can be considered as partially-structured data, among them:

- Bibliographic Databases such as Medline (http://www.ncbi.nlm.nih.gov/) and DBLP (http://dblp.uni-trier.de/). The highly-structured part contains the authors, title and year published while the semi-structured part contains text description and comments on publications.

- Movies Databases such as IMDB (http://www.imdb.com): the highly structured part includes the title, year, director of the movie while the semi-structured part includes film reviews.

- Health care systems: the highly-structured part includes the name, address and date of birth of patients and the semi-structured part includes the description of the illness or doctor visits. (for example a health care system in Brazil using both Java and XML available at

  http://java.sun.com/develper/technicalArticles/XML/brazil)

- Product catalogue: the highly-structured part includes the name, price, make and model of the product while the semi-structured part includes the specification and reviews of that product.

Given the existence of large data sets and applications, such as the above, there is clearly a need for data management functionality for this class of documents. That is to say, to organise, store, query, restructure and manipulate large collections of partially-structured XML documents in an efficient way. This requirement is being addressed at present by applying two main strategies, i.e., developing native XML database management systems, and developing systems which utilise and extend conventional relational database management systems. A potential advantage of the latter approach is that it applies and builds on the years of research and development that provided a mature, stable, scalable and effective technology for query optimisation and processing of highly-structured data. The relational database is the major database technology in use nowadays and is likely to maintain its dominant position in the foreseeable future.

So, the research concentrates on using relational databases for XML data management, but seeks a better way to store and query the class of partially-structured data.

Therefore, the basic motivation for this research is the need for improved query processing performance of partially structured XML documents using relational database management systems. The following section presents the hypothesis based on this motivation.

3.2.2 Research Hypothesis

Following on from the argument developed in the preceding section, the research hypothesis is:

> *For the class of XML documents which contains both a prescribed highly-structured part and a semi-structured part, performance enhancement may be achieved over existing query processing techniques for semi-structured documents by using relational database query processing and optimisation technology to exploit pre-knowledge of the prescribed highly-structured part of the data.*

The research tests this hypothesis by introducing and evaluating a new model to store partially-structured documents (as explained in section 3.6). In this model, the highly-structured part is stored using a highly-structured data model (relational database), this allows it to utilise existing optimisation techniques developed for conventional databases to deal with the structured part of the data instead of treating data as if it is totally semi-structured. On the other hand, the proposed model deals with the semi-structured part as a semi-structured data model, and therefore allowing flexibility in dealing with this part of the document. A number of experiments are designed to compare and evaluate the performance of this proposed hybrid model against the two base models. The research is concerned only with large, possibly complex documents, where query processing performance becomes an issue.

The hypothesis narrows this research to study a class of partially-structured XML documents. The next section presents how XML documents can be categorized according to the degree of structuredness. A partially-structured XML document is defined later based on this classification.

3.3 XML Degree of Structuredness

This section discusses how XML documents can be categorised according to their degree of structuredness. The section firstly discusses different ways to categorise XML documents, then the categorization adopted in this thesis is presented.

In the previous chapter, section 2.3.1.2 explained how an individual XML document can be classified into two main categories (Bourret 2005), data-centric and document-centric. The following table summarizes the differences between a data-centric and a document-centric XML document (Kim et al. 2002)

Document- Centric XML	*Data-Centric XML*
Irregular and un-structure content	Structured content
Large amount of mixed content	Little or more probably no mixed content
Order is significant	Order is insignificant
Human consumption	Machine consumption

Table 3.1 Document-Centric vs. Data-Centric XML (Kim et al 2002)

Barbosa et al. (2001) categorized XML documents according to their degree of structuredness as textual documents (e.g. books) and data documents (e.g. a catalogue of books). Each of these categories can have different characteristics as well as different indexing and querying scenarios. They proposed a fuzzy measure to determine how close a document is to each extreme. According to their definition, a document has low structuredness if it is similar to a textual document while it has a high structuredness if it is similar to a data document. Suciu (2002) proposed a similar classification, un-structured (the web) and fully structured (relational data) with a spectrum of partially structured data in between. In this context, partially-structured data was defined in a different manner (different to the definition adopted by this research in section 3.4, this research considers the data which belongs to the spectrum between highly-structured and un-structured as semi-structured data). Yao et al. (2002, 2003, 2004) developed XBench, a family of XML benchmarks. They classified XML applications into data-centric (DC) and text centric (TC), then classified the XML data into single document (SD) and multiple documents (MD) scenarios. Using this classification produces four different scenarios (TC/SD, TC/MD, DC/SD and DC/MD). The single document

scenario can cover databases such as an e-commerce catalogue that consists of a single document with complex structures (deep nested elements) or dictionaries. The multiple document scenarios cover databases that consist of a set of XML documents, such as an archive of news documents or order transaction data.

This research concentrates on the single document scenario. However, the research generalises to multiple documents scenarios, since the multiple document collections can be trivially transformed into a single document by concatenating these documents into one single document and adding one top level in the document hierarchy. For example, a collection of N documents can trivially be transformed into a single document in which the level 0 tag corresponds to the collection as a whole, and with the N level one sub-elements, which represent each document in the multiple documents scenario.

This research takes these classifications further by classifying XML documents into three categories instead of two according to the degree of structuredness of the data. The main reason for this is to exclude the un-structured documents from the scope of this research, since there is no possibility of exploring their structured part, and concentrate on both highly-structured and semi-structured documents. In this classification, the documents can be categorized as:

- A highly-structured document contains only highly-structured data, i.e. data that has a common and defined structure or organization. Normally, the data can fit easily into a relational data model (Codd 1970) or an object-oriented data model (Cattell et al 2000).

- A semi-structured document contains only semi-structured data (Abiteboul 1997, Buneman 1997, Suciu 1998, Abiteboul et al. 1999 and Abiteboul 2001). For example, data that is irregular or that exhibits type and structural heterogeneity since it may not conform to a rigid, predefined schema as defined in section 2.2.1.3

- An un-structured document contains only un-structured data (Buneman et al 1996, Buneman et al 1997) with no structure defined at all for the data, i.e., raw data or images that lack defined structure or organization.

This classification (as discussed in the next section) provides the base for defining a partially-structured XML document as a hybrid of a highly-structured and a semi-

structured data. As explained in the previous chapter, XML as a document format is flexible enough to represent any of these classifications (section 2.3).

An individual XML document can be classified as highly-structured if it has a well defined and regularly repeated structure. For example, if a relational table is represented as an XML document (as in figure 3.1), this XML document could be classified as highly structured. In such a case, there is a higher probability that the document will be validated against a DTD or an XML schema (some form of explicit collection schema). This is not mandatory, since the existence of a DTD or XML schema does not indicate that the document has a specific degree of structuredness. For example, an un-structured XML document could be validated against a DTD.

An XML document could hold a more complex structure and still be classified as highly structured. The key factor is that the hierarchy that it represents is well defined and repeated regularly. A data-centric document is a highly-structured document in nature because, as defined earlier, its main use is to transfer structured data between different information systems.

Table1

Id	Name	Phone
1	John Smith	1234567
2	Mick Hunter	
3	John Cameron	7654321

```
<Table1>
 <Record>
  <Id>1</Id>
  <Name>John Smith</Name>
  <Phone>1234567</Phone>
</Record>
<Record>
  <Id>2</Id>
  <Name> Mick Hunter </Name>
  <Phone></Phone>
</Record>
 <Record>
   <Id>3</Id>
   <Name> John Cameron</Name>
   <Phone>7654321</Phone>
 </Record>
</Table1>
```

Figure 3.1 An Example of a Highly-Structured XML Document

On the other hand, an XML document is flexible enough to host semi-structured data, with all its irregularity in type and structure. For example, in the same XML document, data elements with the same tag could be represented in more than one structure or they could be missing altogether. Figure 3.2 shows an example of a semi-structured XML document. As this example shows, the first contact element contains a person "name" tag with a "phone no" tag, the second contact element contains "first name" and "last name" tags (an example of data represented in more than one structure), it contains a "fax no" tag while a "phone no" tag is missing, and finally, the last contact element contains a "name" tag, but it is a University "name" tag and not a person "name" tag. It

contains a "web site" tag but the "phone no" and "fax no" tags are missing. Normally, there is no XML schema to validate the XML document, but if it exists, it is a complex one and must contain all the different scenarios for different data representation.

```
<Contacts>
 <Contact>
  <Name>John Smith</Name>
  <Phone>1234567</Phone>
</Contact>
< Contact>
  <FirstName>Mick</FirstName>
  <LastName>Hunter</LastName>
  <Fax>165235</Fax>
</Contact>
< Contact>
  <Name>University of Sheffield</Name>
   <Website>www.shef.ac.uk</ Website >
 </Contact>
</Contacts>
```

Figure 3.2 An Example of a Semi-Structured XML Document

Finally, an XML document could represent un-structured data. Figure 3.3 shows an example of an unstructured XML document.

<BBC Website Address:

"http://news.bbc.co.uk/sport1/hi/football/world_cup_2006/teams/portugal/5116722.stm" >

Figo cleared for England showdown

Van Bommel and Figo went head to head on Sunday

Luis Figo can play in Saturday's World Cup quarter-final against England after Fifa ruled out further action for a headbutt on Holland's Mark van Bommel.

The Portuguese skipper was booked and because referee Valentin Ivanov took action, Fifa will not intervene.

Fifa communications director Markus Siegler said: "He was sanctioned immediately by the referee at the time.

"The referee's report is being analysed but it is very unlikely anything will happen as he has been sanctioned."

</BBC Website >

Figure 3.3 An Example of an Un-Structured XML Document

As the above example shows, there is no common structure between the data inside the document or the document can be just a plain text document, for example, data that is not related to each other or mixed data elements.

A document-centric (or text-centric) XML document can be classified as highly-structured or somewhere in between semi-structured and un-structured depending on the degree of structuredness of its data. An example of document-centric would be an XML document containing a text book. If it has a regular structure such as chapters, sections, paragraphs and so on, then it can be classified as a highly-structured XML document. On the other hand, if it is just a plain text it is considered as un-structured.

Categorizing XML according to its degree of structuredness can be seen on the level of XML elements. An XML document consists of elements; these elements could be complex elements containing other elements, or simple elements containing only raw data. An XML document must have one complex element as a root element (if it has a simple element, this means that the XML document has only one data element). A complex element by itself could be considered according to its degree of structuredness;

it can be considered as highly-structured if it always has the same sequence of complex sub-elements and / or the same sequence of simple sub-elements, or it could be considered as semi-structured or un-structured, if this condition is not met. In other words, an XML document can be classified as highly-structured if its root element is considered as a highly-structured element. Taking this classification to the level of the elements opens the door to documents that are a hybrid between highly-structured and semi-structured; that is to say, contain highly-structured elements as well as semi-structured elements.

There is no clear-cut line between defining a document as highly-structured, semi-structured, or un-structured. The document may appear as un-structured initially, but after analysis, it could become semi-structured or highly-structured. This classification of XML documents is important to this research, since this study is concerned with the hybrid of highly-structured and semi-structured. The following section builds upon this informal classification to define this hybrid model.

3.4 Partially-Structured XML

Partially-structured data is a hybrid of highly-structured and semi-structured data (Lahiri et al. 1999). A partially-structured XML document is defined in section 3.4.1, followed by an example, then its advantages compared to highly-structured and semi-structured data.

The next section, 3.5, discusses the proposed storage model in the literature for XML and how these models perform when used to store partially-structured data.

3.4.1 Defining a Partially Structured XML Document

Lahiri et al. (1999) introduced the notion of partially-structured data in the Ozone system. Their definition is as follow:

> *"A hybrid data that is partially structured and partially semi-structured. It contains entry points from structured data to semi-structured data and vice versa."*

The above provides a general definition of data sets that comprise a mixture of both highly-structured and semi-structured data. However, the definition is not specific to XML-encoded data, since the focus of Lahiri et al's work was on managing hybrid data

set in general. Since, the focus of this research is partially-structured XML-encoded data; a more specific definition is required. This definition builds upon definitions of the degree of structuredness of XML elements (defined and discussed in section 3.3). In this discussion, XML element is defined as a highly-structured element if it has the same sequence of complex sub-elements and / or the same sequence of simple sub-elements. Otherwise, the element can be defined as a semi-structured element. The definition must include the notion of hierarchical structuring between the different element components, since this is an inherent feature in an XML document. Also, the definition must include the notion of "entry points" for access to, and navigation between the highly-structured and semi-structured elements and vice versa contained within, or linked to the same or other XML document. Specifically, entry points can be accessed and navigated using the XML technologies which provide inter and intra links between different elements within the same document or in another document, such as IDREF, XLink and XPath. The proposed definition, therefore, is:

> *A partially-structured XML Document comprises within its hierarchical structure at least one highly-structured element and at least one semi-structured element, where*
>
> - *Each sub-tree rooted by a highly-structured element can contain a combination of highly-structured sub-elements and/or semi-structured sub-elements as its nodes.*
> - *Each sub-tree rooted by a semi-structured element, must comprise only semi-structured sub-elements, as its nodes.*
>
> *The relationship (or the entry point) from any element (either highly-structured or semi-structured) to any other element within a partially-structured XML document is defined by the document's hierarchical structure. Or, is defined by an element link, which is implemented using inter and/or intra XML technologies to link to different parts inside or outside the document (such as IDREF, XLink and XPath).*

As the above definition states, a partially-structured XML document allows the mixture of both highly-structured and semi-structured data in the same XML document, where an entry point (or a relationship) exists between these two parts. This research is concerned with an individual and complex partially-structured XML document (single

document scenario – as explained in section 3.2.1). Based on the classification of XML documents according to its degree of structuredness, an XML document can be seen as a structured document that hosts semi-structured components. It contains one or more complex elements that can be classified as highly structured as well as one or more complex elements that can be classified as semi-structured. The highly-structured elements could contain other highly-structured or semi-structured elements, while once an element is defined as semi-structured; all its sub-elements are defined as semi-structured. The relationship between different elements can be presented in the hierarchy of the document or by using normal linking techniques defined by XML technologies such as IDREF to link to another element inside the same document or XLink to link to another XML document or XPointer to link to a specific element in another XML document.

This definition is adopted throughout this research to represent a partially-structured XML document. The next section shows that no extension is needed to the XML Schema to validate a partially-structured XML document.

3.4.2 XML Schema Representing a Partially-Structured XML Document

A defining aspect of partially-structured data is that it contains a part that is structured and therefore has a prescribed structure. The structure of XML-encoded data can be made explicit using a variety of XML technologies, including DTD and XML Schema. In my research I assume that the structure of the structured component of a partially-structured XML document is prescribed by an XML schema. Specifically, the XML Schema (W3C XML Schema online) can be defined as a formalization of constraints that apply to a class of XML documents (Vlist 2002). It can be used to validate any document inside a collection of partially-structured documents. There is no extension needed to the XML Schema to validate a partially-structured XML document. Any semi-structured part of the document can be defined in the schema as anyType (XML Schema Part 0 Online). The definition of an element in this way allows the unconstrained definition of this element and all its sub-elements. This represents exactly the semi-structured part inside a partially-structured document, since the tree starting from this element represents the semi-structured part.

The following section gives an example of a partially-structured document and how its XML Schema is defined using anyType element definition.

3.4.3 Example of a Partially-Structured XML Document

A number of domains where partially-structured data scenarios occur have already been identified (in 3.2.1). One example can be found within the health care domain, an XML document containing patients' information can include:

- *Highly-Structured part*: patient Id, name, address, phone, email and date of birth
- *Semi-Structured part*: description of the illness, doctor visits

Figure 3.4 shows an example of a patients XML document.

```
<Patients>
    <Patient>
        <Id>12345</Id><Name>George Wilson</Name>
        <Address> <Street>West Street</Street><City>Sheffield</City>
            <PostalCode>S12345</PostalCode>
        </Address>
    <Phone>2345678</Phone>
<Phone>077343545</Phone>
<DateofBirth>01/01/1972</DateofBirth>
<Illnesses>
    <Illness> Pneumonia
        <Description>xlink:type="simple"
 xlink:href="http://Illness Description.com/List.xml#Pneumonia">
        </Description>
    <Date>25/12/2004</Date> Advised to rest for 5 days
    <DoctorName>Mick Bil</DoctorName>
    </Illness>
    <Illness> <Date>25/11/2003</Date> car accident leading to an X-
    Ray on his leg <XRayDate>25/11/2003</XRayDate >
    <XRayTechnitionId>LK</ XRayTechnitionId >
    </Illness>
  </Illnesses>
</ Patient>
```

Figure 3.4 An Example of Partially-Structured XML document

Figure 3.5 shows the XML Schema for the XML document shown in figure 3.4, the main issue is that the Illness sub element is represented by anyType. This means that there is no constraint on it and all its sub-elements

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.PatiantSystem.com"
  xmlns=" http://www.PatiantSystem.com"
  elementFormDefault="qualified">
 <xs:element name=" Patients">
  <xs:complexType>
   <xs:sequence>
    <xs:element name=" Patient">
     <xs:complexType>
      <xs:sequence>
<xs:element name="Id" type="xs:integer"/>
<xs:element name="Name" type="xs:string"/>
       <xs:element name="Address">
        <xs:complexType>
         <xs:sequence>
    <xs:element name="Street" type="xs: string"/>
     <xs:element name="City" type="xs:string"/>
    <xs:element name="PostalCode" type="xs:string"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     <xs:element name="Phone" type="xs:string"/>
     <xs:element name="DateofBirth" type="xs:date"/>
    <xs:element name="Illnesses" type="xs:anyType"/>
    </xs:sequence>
  </xs:complexType>
```

```
        </xs:element>
      </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Figure 3.5 XML Schema for Partially-Structured XML document

As this example shows, a partially-structured XML document is a hybrid of highly-structured and semi-structured. The entry point from structured part to the semi-structured part is represented in the hierarchy of the document as an illness element which is a child element of the patient element. The reverse is represented inside the illness element itself as it references another element inside another XML document using XLink/XPointer.

The above illustrates that there is no need to extend the XML Schema to validate a partially-structured XML document.

3.4.4 Advantages of Using a Hybrid Model to Store Partially-Structured Data

Based on the previous discussion, a number of questions arise; can a purely highly-structured data model be used efficiently alone to store and query partially-structured data? Can a purely semi-structured data model be used efficiently alone to store and query partially-structured data? Or is there a need for a hybrid model to better deal with partially-structured data? This section discusses these questions by presenting the disadvantages of using either these systems alone to reach a conclusion on the need for a hybrid model.

- Using a highly-structured data model alone to store partially-structured data

  Representing partially-structured data using a highly structured data model (such as a relational data model) yields a complex data structure with redundancies when storing the semi-structured part of the document. This arises when mapping the semi-structured data part into the highly-structured data model due to the structural complexity of the semi-structured part. Any small change to the data structure could yield a large evolution in the highly structured data schema. The structured data

model does not provide the navigational search approach to its data (data browsing) which is a useful feature of the semi-structured data especially when the structure of the data is not known.

- Using a semi-structured data model alone to store partially-structured data

  A purely semi-structured data model such as Lore (McHugh et al. 1997) does not take into consideration the structured part of the data for the highly-structured part of the document. This means that it deals with the whole document as if it were semi-structured without taking advantage of its strong structure and typing of its highly-structured part to provide optimisation either for storing or querying.

As the above two points showed, there are disadvantages of using a pure highly-structured data model or pure semi-structured data models to store and query partially-structured data. This poses a question; can a hybrid model produce a better performance in storing and querying partially-structured data?

To answer the above question, the research proposes a partially-structured data model. This model deals with the highly structured part of the document as highly-structured data and therefore benefiting from the optimization technique available for this part of the document, and it deals with the semi-structured part of the document as semi-structured data and therefore providing a flexible approach to storing and querying this part of the document. This model is explained in section 3.6 after discussing the use of relational databases to store and query XML documents in general and the other related storage models proposed in the literature to deal with partially-structured data in particular. These are explored in the next section.

3.5 Related Work

As discussed in section 2.5, XML storage models can be categorized in three main sections; those using the file system, those using mature database management systems and those using novel storage structures. Two of these categories were discussed in this section (2.5), namely storing XML as its native text files and novel storage structures. In this section, a more in-depth and detailed discussion of using relational database management systems to store XML data is presented, this is because the research hypothesis (see 3.3.2) explicitly limits the scope of this study to use relational

databases. Their suitability for dealing with partially-structured data is presented. Two other systems, which are related to storing partially-structured XML documents, are discussed as well. These systems are the Ozone system (Lahiri et al. 1999) and STORED (Semi-structured TO Relational Data) System (Deutsch et al. 1999). The Ozone system was designed primarily for partially-structured data and is described in section 3.5.2 while the STORED system was designed for semi-structured data in general, but can deal with partially-structured data. The STORED system is described in section 3.5.3. Finally section 3.5.4 discusses the pros and cons of these models, builds upon them and motivates the hybrid model proposed in this thesis to store and query partially-structured data.

3.5.1 Using Relational Database Management Systems

Since the emergence of XML as a new technology in 1998 (XML 1998: online), using relational or object-relational database management systems to store XML documents was considered as an option (Shanmugasundaram et al. 1999 and 2001,Shimura et al. 1999, Florescu and Kossmann 1999, Schmidst et al. 2000, Klettke and Mayer 2001, Yoshikawa and Amagasa 2001, Kudrass 2002, Kudrass and Conrad 2002, Runapongsa and Patel 2002, Tian et al. 2002, Amer-Yahia and Srivastava 2002, Bohannon et al. 2002, Kuckelberg and Krieger 2003, Han et al. 2003, Harding et al. 2003, Leonov and Khusnutdinov 2004, Pal et al. 2004, Lu et al. 2006, Na and Lee 2005, Balmin and Papakonstantinou 2005, Chaudhuri et al. 2005, Qin et al. 2005, Pardede et al. 2005, 2006 and Pal et al. 2006). This is due to the robust, well established and optimised performance a relational database can offer. A recent empirical study of XML data management shows that relational database management systems outperform native XML database systems in processing XML data (Lu et al. 2006). This finding depends on a number of factors such as document structuredness, data size and the queries' workload. As this research hypothesis stated in section 3.2.2, this study uses relational database management systems as the base to store and query partially-structured XML documents.

Due to the mismatch between the complex tree structure of XML and the simple flat structure of relational databases, there are many possible frameworks in which to store an XML document into a relational structure. These frameworks can be categorized into two main groups. The first is to "shred" the document into smaller parts (as in

Shanmugasundaram et al. (1999), Shimura et al. (1999) and Florescu and Kossmann (1999), Klettke and Mayer (2001), Yoshikawa and Amagasa (2001), Kudrass and Conrad (2002), Runapongsa and Patel (2002), and Kudrass (2002)) and the second is to store the whole document as one part into a relational structure (as in Kudrass 2002, SQL Server 2005 XML online, Oracle Database 10G Release 2 Online).

Shredding an XML document into smaller parts that can fit into a relational structure can be subcategorized into the following approaches:

- *Structured-Mapping Approach:* Shanmugasundaram et al. (1999), Klettke and Mayer (2001), Kudrass and Conrad (2002) and Runapongsa and Patel (2002). In this approach, the XML document is mapped to an equivalent relational structure. Thus, the hierarchical structure of an XML document is mapped as a collection of tables, one of which represents the document itself and the others represent its nested tagged elements. The lowest-level of tagged elements are represented as table columns. For example, if an XML element contains the first name, last name and date of birth, it can be mapped to a relation containing three columns (first name, last name and date of birth) as well as any other columns to represent the hierarchal link between this element and other elements and to represent the order. To implement this approach, there are four steps to be followed (Atay et al. 2004, 2007)

  1. Schema mapping: Build the appropriate relational structure to host XML data.
  2. Data mapping: inserting XML data into the target structure.
  3. Query mapping: translate XML queries into SQL queries.
  4. Reverse data mapping: publish XML data from relational data.

- *Model-Mapping Approach:* Shimura et al. (1999) and Florescu and Kossmann (1999), Yoshikawa and Amagasa (2001) and Kudrass (2002). This approach maps XML to a generic schema. The XML graph structure itself is mapped into a relational schema in contrast to mapping the XML data structure as the previous approach. That is to say, this approach models XML nodes and edges not the XML data itself. There are three different ways to store the graph edges; Edge, Binary, and Universal (Florescu and Kossmann 1999).

- The Edge approach stores all edges of the graph that represent an XML document in a single table (named the edge table). This table contains object identifiers for the source and the target objects. It also contains the label of the edge, a flag to indicate if it is an inner node or a value node and an ordinal number indicating the order of the elements (Florescu and Kossmann 1999).
- The Binary approach stores all edges with the same label into one table. This approach corresponds to horizontal partitioning of the Edge approach (Florescu and Kossmann 1999).
- The Universal approach creates a single table to store all the edges in the XML document. This table corresponds to the results of a full outer join of all binary tables (Florescu and Kossmann 1999).

There are two different ways to store the values; as separate value tables and Inlining (Florescu and Kossmann 1999).

- The separate value tables approach creates a value table for each conceivable data type. There is a link between the graph edge table and these value tables. In this case, the flag in the edge table indicates which value table to reference (Florescu and Kossmann 1999).
- The Inlining approach stores the value on the node in the same edge table and there is a value column for each data type (Florescu and Kossmann 1999).

The three ways to store the graph's edges and the two ways to store the values lead to six alternative mapping schemas.

- *XPath-Based Mapping Approach:* XRel (Yoshikawa and Amagasa 2001) used the XPath (Online) data model to model XML documents into elements, attributes and text nodes. These nodes are stored in three different tables. Another table used is the Path table, which contains *pathid* and *pathexp* (path expression), *pathid* is a unique number per path while path expression contains the actual path expression starting from the root for this node. The main feature of this system is that recursive queries (expressed as '//' in XPath expression) can be transformed to a string match. For example, if the XPath expression is

dept//firstname, this means any path expression that starts with *dept* and ends with *firstname* (for example, *dept/lecturer/firstname*, *department/papers/authors/firstname*), this can be expressed as *pathexp like '#/dept#/%#/firstname#/'*. Handling the recursive approaches in other models (such as model-mapping approach) could produce a number of joins equal to the length of the longest path matching the query. A similar approach is the one used by (Khan and Rao 2001 and Khan et al. 2002). In this approach, the path approach uses dot '.' instead of slash '/', so the path expression is specified as (*.dept.lecturer.firstname*). They utilised the DTD to solve the problem of set value (as it is defined in DTD using *). It used a unique sequence number starting from zero to identify the element position in the set. This is unlike XRel approach, which uses the region concept, as each element has a start and end attribute defining the number of bytes counted from the beginning of the document till the start and end of this element. The start attribute can tell the order of the elements inside the set and start and end is also used to identify the ancestor/descendant relationship (for example, if an element starts at 10 and ends at 100, and another element starts at 50 and ends at 60, then the second element is a descendant element of the first one).

The structure mapping approach is suitable for highly structured documents (Lu et al. 2006) and specifically for storing a large number of XML documents that conform to a static and limited number of document structures or DTDs (Yoshikawa and Amagasa 2001). The model-mapping approach is more suitable for the semi-structured or un-structured documents. Its disadvantage is that when querying the data, it can produce a huge number of internal joins.

The alternative approach to "shredding" XML into a relational database is to store the whole XML document as one unit. This can be subcategorized into the following two types:

- Storing the document as CLOB field (Kudrass 2002). To query an XML document stored in a CLOB field, a similar approach to the one used to query an XML document stored in a text file is used. The document is loaded in the memory (for example, as a DOM tree) and accessed using standard techniques such as Xpath (Online) and XQuery (Online, Fernandez et al. 1999). As found by Tian et al. (2002) this approach shows serious limitations with big XML

documents since it requires the presence of the whole XML document in the memory. Therefore, the query response time increases linearly with the size of the document. One approach to overcome this problem is to create text aware index and store it in the database (as in Grossman et al. 1997) to use it to retrieve XML data from the CLOB field. In both cases (with and without the indices), it is not possible to retrieve a part of the document, the system has to retrieve the whole CLOB field, then either use the DOM approach or indices to retrieve the necessary data from it. Another scenario is to use the tools provided by the DBMS for searching text, such as full-text, proximity, synonym and fuzzy searches. DBMS are starting to make these searches XML-aware which may improve the use of this technique in the future (ORACLE Database 10G Release 2 Online).

- The second approach is to store the document as an abstract XML data field. This was recently introduced in the relational model. SQL:2006 (online) is the latest standard for relational databases and it standardises the use of XML data types in relational databases. Currently, a number of commercial database systems implement this standard in their DBMS, for example Oracle, Microsoft, IBM... etc. These DBMSs offer a number of ready-to-use indices which will inevitably show better performance in querying the document as opposed to accessing the document in CLOB (for example, SQL Server 2005 XML online).

Storing the whole XML document in a relational database provides a flexible way to store un-structured and semi-structured XML documents, simply because these documents are stored without any need for pre-processing. The round-trip problem (reconstruction of the whole document from its shredded parts) does not exist. This simplicity comes at a price when using the CLOB approach, a similar approach to querying XML document stored in the file system is used to query XML document stored in CLOB. In this case, these two approaches (using the file system and CLOB approach) share the same serious limitations as explained in section 2.5.1 (Tian et al. 2002), since they require loading of the whole XML document into memory. So, the query response time increases linearly with the size of the document. While storing XML documents as the new XML data type with the use of indices offered by the DBMS is a more promising approach, its performance has yet not been tested because this feature was recently released (for example, MS SQL Server 2005 which was

released in December 2005). This approach is used in this study and is explained in more detail in the following chapter.

3.5.2 Ozone System

In the previous section, the storage models based on the relational data model to store and query XML documents in general were presented. In this section, a storage model which was proposed especially for partially-structured data is presented, the Ozone system (Lahiri et al. 1999).

The Ozone system introduced partially-structured data as an integration of highly-structured and semi-structured data. It extended the structured object database model ODMG (Cattell et al. 2000) to host both semi-structured data as well as the Object Query Language OQL (named OQL^S) to store and query semi-structured data, and is based on the Object Exchange Model OEM (Papakonstantinou et al. 1995) and the Lorel Language (Abiteboul et al. 1997).

The idea behind the Ozone system is to extend the ODMG model by a new, built-in class type OEM which then allows crossover points from the structured part of the data the semi-structured part and vice versa. It extends the semantics of OQL to cover this extension by introducing new semantic operations that allow this crossover. The syntax of the extension (which is called OQL^S) is identical to OQL. The semantics is also identical in the case of a query on only the highly-structured part of the database. The following subsections explore the Ozone system in more detail by describing its motivating example, and then its design concepts.

3.5.2.1 Ozone Motivating Example

The Ozone system (Lahiri et al. 1999) uses an example of a simplified on-line broker that sells products on behalf of different companies. This is a good example of partially-structured data since some information such as a product's number and name and companies' details are highly structured while product information and reviews about products are semi-structured because each type of product can have details which are specific to it and not applicable to other product types. For example, computers may have a processor attributes while monitors may have a screen size attribute. Figure 3.6 shows the design of the structured part of the data, while figure 3.7 shows a possible semi-structured part for the *'product info'* attribute. This reveals how a link could be

made from the structured part to the semi-structured through the '*product info*' attribute and also the reverse through the '*competing*' attribute.

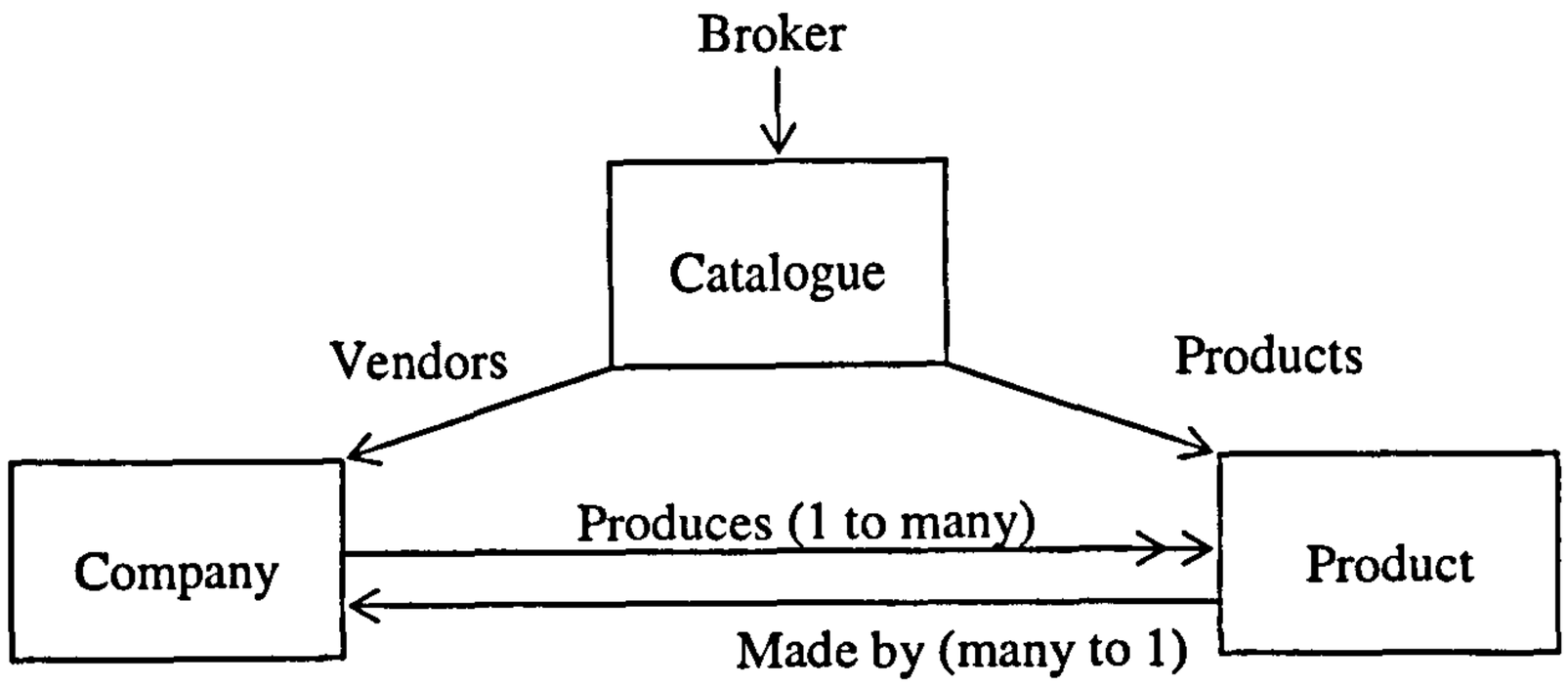


Figure 3.6 Structured ODMG classes in the retail-agency database for the Ozone System Example (Lahiri et al. 1999)

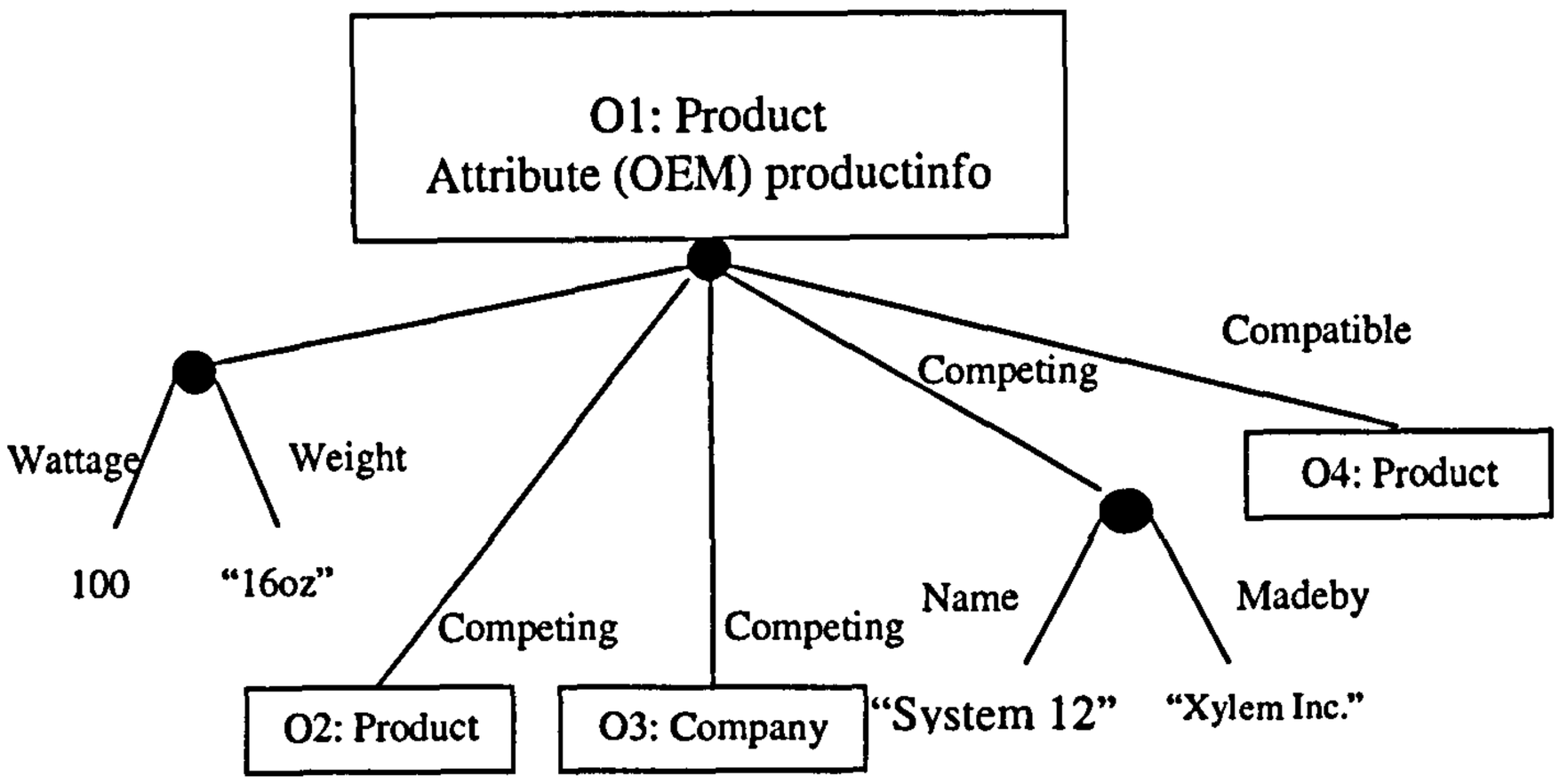


Figure 3.7 Example OEM graph for the prodinfo attribute of a Product object for the Ozone Example (Lahiri et al. 1999)

Finally, the semi-structured part is flexible enough to show a competing product such as "System 12" which is not part of the structured database as a complex OEM object.

3.5.2.2 Ozone Design Concept

The previous example shows how both structured and semi-structured data can be linked. In this section, some design concepts of the Ozone system are discussed.

The Ozone system proposed adding a new OEM class; this allows the storage of semi-structured data in the ODMG model. This OEM class can be classified into two

categories, OEMcomplex and OEMatomic representing complex and atomic OEM objects respectively. OEMcomplex is a collection of pairs (label (as string), value (as OEM object)). Because an XML document represents an ordered graph; there are two subclasses for this collection as *OEMcomplexset* for the unordered collection and *OEMcomplexlist* for the ordered one.

OEMatomic represents atomic values, which are also represented by subclasses. For example OEM_integer encapsulates the type integer. It represents OEM (integer) objects. And so on for the other atomic types such as OEM_real, OEM_string, and OEM_Boolean ... etc. OEM_object can be used to reference any other class inside the database and to be the cross over point from the semi-structured to the structured part since the object class is a super class of all the classes inside the database, but for performance reasons, and instead of making it general, which leads to determining the type of the class at run time, the Ozone system defines an OEM class for each class in the database to be used as a reference to that class such as OEM_product and OEM_company. Finally, the atomic OEM objects encapsulate non-atomic literal values (tuple, set, list...etc), which could be represented by equivalent complex objects. But also for performance reasons, the Ozone system defines additional OEM subclasses encapsulating the types of non-atomic literal types that most likely to be used in queries.

3.5.3 STORED System

After presenting relational storage models and the Ozone system in the last two sections, in this section, a storage model proposed for semi-structured data but relevant for partially-structured data is presented, it is the STORED (Semi-structured TO RElational Data) System (Deutsch et al. 1999).

The STORED system defines a declarative query language that specifies a storage mapping from a semi-structured data model (an ordered version of the OEM data model (Papakonstantinou et al. 1995)) to a relational data model and an overflow graph. Using data mining, the mapping tries to exploit any patterns in the data and then create the equivalent relational structure to store this data, and any part of the data which cannot be fitted into the relational structure, is stored in an overflow graph. When a query is executed, it is re-written into a query over the relational store. Any update over the semi-structured data is also re-written into an update over the relational store. The two possible applications of this system are:

- To store and manage existing semi-structured data sources efficiently.
- To convert relational sources into a semi-structured format such as XML.

The STORED system can accept partially-structured data and store the highly-structured part of it into a relational structure while storing the semi-structured part of it as an overflow graph. This idea of dividing the document into two parts and storing each part according to its natural format is important and this research will follow it up as described later in this chapter.

3.5.4 Discussion

The Ozone system is very specific about the problem that it is applied to. For example, it defines a class called OEM_product, which is only applicable for the problem definition it is designed for. For the STORED system, although the initial mapping of semi-structured data into the relational data model may yield a good result, with time, the performance can degrade, because of changes in both the data and the query mix; this requires a new mapping to be generated. On the other hand, there is an overhead in dealing with the overflow graph to store semi-structured data.

One of the future extensions of the the Ozone system as described in the Ozone research paper (Lahiri et al. 1999) was:

- Object-relational Ozone: to define a similar semi-structured extension to the object-relational data model

This is the first point that this research takes further. An XML data type is introduced in SQL:2003 and implemented in a number of database management systems such as Microsoft SQL Server (SQL Server 2005 XML online) and Oracle (Database 10G Release 2 Online). This is the base for extending the object-relational data model to host the semi-structured part of the document. The second point is based on STORED system. The proposed system follows the idea proposed by STORED to store the structured part of the document by mapping it to an equivalent relational structure. The proposed model is different in the way the semi-structured part of the document is stored. In STORED this part is stored as an overflow graph while in the proposed system it is stored as an XML data type. Storing the semi-structured part in XML data type provides a more flexible approach compared to the overflow graph proposed by

STORED. There is no need for a special algorithm to insert data or to query data compared with the overflow graph data representation.

The following section describes in more detail the model proposed in this research to store and query partially-structured XML data.

3.6 The Proposed Model for Partially-Structured XML Documents

As explained previously, partially-structured data is a hybrid model between highly-structured and semi-structured data. The proposed model for partially-structured XML documents is a hybrid between a structure mapping approach and storing XML as an XML data type. The mapping of the XML document is as follows:

- The highly-structured part is mapped into an equivalent relational schema, therefore benefiting from the optimisation techniques offered by the relational database for this part of the document.
- The semi-structured part is stored as XML data type, therefore allowing a flexible way to manage this part of the document.

Comparing this model with the existing models for partially-structured data as they were discussed in the previous section, this model is based on the relational data model (vs. Ozone (Lahiri et al. 1999) model which was based on the object oriented data model). There is a similarity between The STORED (Deutsch et al. 1999) model and the proposed model when dealing with the highly-structured part, as both of them map the highly-structured data part to an equivalent relational schema. The difference is when dealing with the semi-structured part. In STORED, the semi-structured part is mapped to an overflow graph. Using the overflow graph needs a special algorithm to insert and update data in this part of the document. However, in the proposed model, there is no need for a special algorithm to deal with this part as it is mapped to an XML data type.

To query the proposed storage model, the query language is a mix between SQL and XQuery, where SQL can be used to query the structured part while XQuery can be used to query the semi-structured part.

In the proposed model, the inflexibility of dealing with the semi-structured part using a relational structure can be avoided as the semi-structured part of the document is mapped to an XML data type, whereas the highly-structured part is mapped to an

equivalent relational structure. This allows the structured part to take advantage of the full power of the query optimisation offered by the relational database management system while allowing more flexibility for the semi-structured part of the document. Hence, the irregularity and type and structure heterogeneity of the semi-structured part of the document does not present a problem in the proposed model.

To summarise, the proposed hybrid system accepts both highly-structured and semi-structured data, stores them as their natural data model, taking the advantage of both of them and providing links between them so users can query and navigate both simultaneously.

3.7 Conclusion

In this chapter, the research's motivations were presented and the hypothesis was derived. That hypothesis postulates that better query processing can be achieved using relational database technology to exploit pre-knowledge of XML data structure. Accordingly, a proposed hybrid model for partially-structured data was presented. This hybrid approach is based on two base models; structured mapping approach and storing the whole XML document into an XML data type. Further, the chapter argues that in addition, the hybrid approach is likely to have performance advantages over storing the whole XML document into an XML data type and can be more flexible than in storing the whole document as a structured mapping approach.

A number of experiments were constructed to test the performance of this proposed mode. Chapter four discusses in more detail the construction of these experiments while the results of the experiments and the analysis of these results are presented in chapter five.

Chapter 4 Experiment Design

4.1 Introduction

In the previous chapter, the research motivations were discussed and the research hypothesis was formulated. The research hypothesis is a proposition that performance enhancements can be achieved by exploiting pre-knowledge of prescribed structure within XML documents and/or across XML documents collections. This was followed by a discussion of how XML documents can be categorised according to their degree of structuredness, so as to analyse structural properties for which this approach is applicable. Accordingly, the definition of the class of partially-structured XML documents as a hybrid of highly-structured and semi-structured data was presented. This was followed by a discussion of different XML storage models for XML data, and an analysis of their potential for storing and querying partially-structured data. The conclusion of this analysis was an elaboration of the hypothesis, in which a proposed storage model was presented, which may have a potential to realise performance enhancements for partially-structured data.

The proposed model was designed to exploit pre-knowledge of highly-structured components of the data, while at the same time allowing flexibility in storing ad-hoc semi-structured data. Specifically, my hypothesis is that enhanced querying can be achieved by storing partially structured XML documents in a relational database so that mature relational query optimisation technology can be applied when querying the structured part of the document. In order to test this hypothesis, a model has been developed to achieve this relational representation. The structured parts of the document are mapped onto a relational structure that models their structures, while the semi-structured parts are stored as instances of the XML data type.

This chapter presents the experiments' design to evaluate the proposed storage model as a mean of testing the research hypothesis. These experiments were designed to compare the relative performance of the proposed hybrid model against the two base models it combines; mapping the whole XML document to an equivalent relational structure and storing the whole XML data into an XML data type.

Accordingly, to achieve this objective, this chapter is organised as follows: section 4.2 discusses the objective of the experiments, followed by the strategy for achieving those

objectives, then it presents the current SQL/XML standard and it ends by describing the different storage models that were used in the experiments. Section 4.3 discusses the current benchmarking techniques proposed by the research for XML in general, and this section concludes with a comparison between these different techniques and identifying one of these techniques to be used in the experiments. Section 4.4 presents the adaptation needed for the chosen XML benchmark technique to deal with partially-structured XML documents. This includes both the data set and the query set used in the experiments. Section 4.5 shows how the experiments are conducted and how the results are evaluated while section 4.6 presents the experimental environment which includes both the hardware and the software used in the experiments. Finally section 4.7 concludes this chapter. The experiments' results and analysis are discussed in the following chapter.

4.2 Experimental Design

This section outlines the experimental design. Section 4.2.1 defines the objectives of the experiments, followed by section 4.2.2 which explores the strategy to achieve these objectives. Section 4.2.3 discusses the SQL/XML standard. Finally, section 4.2.4 presents the storage models used in the experiments.

4.2.1 The Objective of the Experiments

The research hypothesis as it was presented and discussed in more detail in section 3.2.2 is:

> *For the class of XML documents which contains both a prescribed highly-structured part and a semi-structured part, performance enhancement may be achieved over existing query processing techniques for semi-structured documents by using relational database query processing and optimisation technology to exploit pre-knowledge of the prescribed highly-structured part of the data.*

In accordance with this hypothesis, a hybrid model was proposed (section 3.6) to deal with partially-structured XML-encoded data. This model takes into consideration the utilisation of the known knowledge of the prescribed structured part of a partially-structured data collection. This model is a hybrid model which combines a storage mapping approach (Shanmugasundaram et al. 1999 and 2001,Shimura et al. 1999,

Schmidst et al. 2000, Klettke and Mayer 2001, Yoshikawa and Amagasa 2001, Kudrass and Conrad 2002, Runapongsa and Patel 2002, Tian et al. 2002, Amer-Yahia and Srivastava 2002, Bohannon et al. 2002, Kuckelberg and Krieger 2003, Han et al. 2003, Harding et al. 2003, Leonov and Khusnutdinov 2004, Pal et al. 2004, Lu et al. 2006, Na and Lee 2005, Balmin and Papakonstantinou 2005, Chaudhuri et al. 2005, Pardede et al. 2005, 2006 and Pal et al. 2006) to represent the highly structured component and which utilises an XML data type (SQL:2003, SQL:2006 (ISO/IEC 9075, part 14, XML-related specifications - SQL/XML), Krishnaprasad et al. 2005, Murthy et al. 2005, Singh et al. 2005, Rys 2005, Rys et al. 2005, Beyer et al. 2005, Pal et al. 2005 and 2006, Özcan et al. 2006 and Lacoude 2006) to represent the semi-structured data component. So, the main objective of these experiments is to test the relative performance of this hybrid model against the two base models, storage mapping and the use of XML as a data type. The strategy to achieve this objective is presented in the next section.

4.2.2 The Experiments' Strategy

To achieve the experiments' objective discussed in the previous section, it is necessary to determine appropriate metrics and analysis methods to compare performance, and establish appropriate data sets, query sets, scenarios and an operational environment within which meaningful comparisons can take place. It is also necessary to determine the significance and limitation of the results achieved in this way. Accordingly, the strategy to achieve the above objective is:

- Review the current SQL/XML standard, and the different implementations in the database industry, and choose which database management system to be used as a test bed for these experiments. Section 4.3.2 presents the results of this review and ends up with the chosen database management system.

- Determine exactly which XML storage approaches can be compared within the experiments, and present the rational for these choices as in section 4.3.3. Specifically, XML storage approaches must be selected which characterise the intrinsic properties of the hybrid approach, proposed in section 3.6, and the two approaches that it combines, i.e., storage mapping and XML as a type, such that valid and generalisable comparisons can be made.

- Review the current XML benchmarks and choose the most appropriate one to be used as the basis for the experiments. Section 4.3 reviews and analyses current

XML benchmarks and compares and contrasts between them. Then nominate the most appropriate one to be used in this research.

- Adapt the chosen XML benchmark technique to address the aim of the experiments, specifically to test partially-structured data. This includes the choice of the data set as well as the adaptation of the query set to match the partially structured document requirements. These adaptations are presented in section 4.4 which includes two parts; a section for the adaptation of the data set and another section for the adaptation of the query set.

- The method of comparison must be established. Specifically, metrics relating to performance and invariants must be identified and appropriate analysis methods must be designed, such that relative performances of the three approaches being compared can be evaluated across the range of relevant variable. Section 4.5 presents this method of comparison.

- The operational environment - within which the experiments are conducted – is described. In particular, consideration must be given to the hardware/software environment within which the experiments are executed and how its stability can be established, the way in which the chosen systems as well as the proposed system are implemented. This point is discussed in section 4.6.

- Finally, the experiments must be conducted and results analysed, including an analysis of their significance and limitation. This is presented in the following chapter.

The above steps outline the strategy used to conduct these experiments. Each step is elaborated in more detail in the following sections in this chapter. The analysis of the results takes place in the following chapter.

4.2.3 SQL/XML Standard

Since the emergence of XML as a new technology in 1998 (XML 1998: online), the international standard agency (ISO) started to incorporate this new technology into the well established SQL standard. The first appearance of the XML into the SQL standard was in the SQL:2003 (ISO/IEC 9075, part 14, XML-related specifications - SQL/XML). In this standard, the XML data model was based on the XML Infoset data model (as described in section 2.3.2.2). The XML data type was firstly introduced in

this standard to store an XML document as a column inside a relational table. The second emerging standard from ISO was recently published in SQL:2006 (ISO/IEC 9075, part 14, XML-related specifications - SQL/XML). One of the major enhancements was the change of the XML data model from the Infoset to XQuery 1.0 and XPath 2.0 data model (XDM, W3C Candidate Recommendation 11 July 2006) (Eisenberg and Melton 2004, Krishnaprasad et al. 2005). XQuery 1.0 and XPath 2.0 data model was not considered mature enough when the SQL:2003 was published.

Every major database management system vendor has followed suit, by incorporating XML into their DBMS products, for example Oracle (Krishnaprasad et al. 2005, Murthy et al. 2005), Microsoft SQL Server (Pal et al. 2005 and 2006, Rys 2005, Rys et al. 2005 and Lacoude 2006), IBM DB2 (Beyer et al. 2005 and Özcan et al. 2006) and Sybase (Singh et al. 2005). However, as the XML standard is still emerging, the implementation is still far from standardised within the different database management systems. There is no way of knowing just how well each DBMS will process XML data as each database vendor has taken different approaches to represent and manage XML data inside their respective database management systems.

In order to run the experiments, there is a need to choose an existing DBMS that supports XML data fields. Due to the time limitations, repeating the experiments with other DBMS is beyond the scope of this project. In particular, the aim of the experiments is to establish the impact of the trade off, as the ratio of highly structured to semi-structured data varies, between database complexity and increased size which are a consequence of storage mapping approaches, and the relative structural simplicity and compactness which is a consequence of XML as a type approach. The analysis of the results - as it will be discussed later in more detail - would focus on the relative performance rather than the absolute performance as far as possible. Therefore, it was decided that one DBMS which supports an XML data type would be selected to provide a common test bed for experimental comparison approaches, to establish the relative performance between these different approaches. Accordingly, the choice of DBMS was seen as arbitrary, and a decision was made to use MS SQL Server 2005 (Pal et al. 2005 and 2006, Rys 2005, Rys et al. 2005 and Lacoude 2006) database management system, because it is the available DBMS in the Department of Computer Science, University of Sheffield, it is widely used internationally and is representative example of SQL/XML technology.

4.2.4 Storage Model Used in the Experiment

The three XML storage models used in the experiments are the hybrid approach described in section 3.6 and the two approaches it combines, i.e.:

- The Structure Mapping Approach: in this case, the whole partially structured XML document is mapped to an equivalent relational structure (Shanmugasundaram et al. 1999 and 2001,Shimura et al. 1999, Florescu and Kossmann 1999, Schmidst et al. 2000, Klettke and Mayer 2001, Yoshikawa and Amagasa 2001, Kudrass 2002, Kudrass and Conrad 2002, Runapongsa and Patel 2002, Tian et al. 2002, Amer-Yahia and Srivastava 2002, Bohannon et al. 2002, Kuckelberg and Krieger 2003, Han et al. 2003, Harding et al. 2003, Leonov and Khusnutdinov 2004, Pal et al. 2004, Lu et al. 2006, Na and Lee 2005, Balmin and Papakonstantinou 2005, Chaudhuri et al. 2005, Qin et al. 2005, Pardede et al. 2005, 2006 and Pal et al. 2006).
- To store XML as an XML data type: the whole partially structured XML document is stored as an XML data type instance (SQL:2003, SQL:2006 (ISO/IEC 9075, part 14, XML-related specifications - SQL/XML), Krishnaprasad et al. 2005, Murthy et al. 2005, Singh et al. 2005, Beyer et al. 2005, Rys 2005, Rys et al. 2005, Pal et al. 2005 and 2006, Özcan et al. 2006 and Lacoude 2006).
- The proposed hybrid approach: in this case, the highly-structured part of the document is stored as an equivalent relational structure while the semi-structured part is stored as an XML data type instance (as described in section 3.6).

The following subsections briefly describe each of these models and present the rational for these choices.

4.2.4.1 Structured Mapping Approach

The first base storage model is the one using structured mapping approach (Document Dependent Approach) (Shanmugasundaram et al. (1999), Klettke and Mayer (2001), Kudrass and Conrad (2002), Runapongsa and Patel (2002) and Amer-Yahia et al. (2004)) (see section 3.5.1 for more detail). In this approach, the XML document is mapped to an equivalent relational structure. Thus, the hierarchical structure of an XML

document is represented as a collection of tables, one of which represents the document itself and the other tables represent the document's nested tagged elements. The lowest-level tagged elements are represented as table columns. To implement this approach, four steps must be followed (Atay et al. 2004, 2007)

1. Schema mapping: build the appropriate relational structure to host XML data.
2. Data mapping: inserting XML data into the target structure.
3. Query mapping: translate XML queries into SQL queries.
4. Reverse data mapping: publish XML data from relational data.

This approach is suitable for highly structured documents (Lu et al. 2006) and specifically for storing a large number of XML documents that conform to a limited number of document structures, (typically defined as DTDs or XML Schema) and which are static over time (Yoshikawa and Amagasa 2001).

However, if storage models in this category are to be applied to partially structured data, they must also be able to represent the semi-structured components of large collections of XML documents, which is problematic. Specifically, the problems for the semi-structured part of the document to be stored using the structure mapping approach relates to the complexity of the generated schema, the inherent ambiguities and contentions that must be resolved and the inflexibility of this resulting relational schema in situations where the structure of the XML document collection is volatile (as discussed in 3.5.1)

The above problems derive from the fact that, in some respects, XML has more expressive power than the relational model (Yoshikawa and Amagasa 2001). This is due to the hierarchical and ordered nature of XML data compared to the simple, flat and unordered nature of relational data (Atay et al. 2007). Therefore, preserving an XML document's semantic content when mapping it to a relational representation is problematic. For example, the relational model cannot directly model the tree structure of the semi-structured part, such as the hierarchies of elements (for example, if it is un-limited recursive elements) as explained by Pasila (2002). Also, XML includes structural directives, such as the Or ('|') choice in an XML document, which cannot be mapped naturally in a relational data model (Yoshikawa and Amagasa 2001). Furthermore, it is hard to force the semi-structured part to adhere to an explicitly specified rigid schema because of its irregularity and because it can evolve rapidly and

data elements may change their data type. Thus, it may be difficult to decide in advance on a single correct schema that can match all the possible variations of the semi-structured part of the document (McHugh et al. 1997). For example, part of the data could hold an author's first name and last name while another part could have the author's full name in one field; this can yield redundancy in storing the data and an ambiguous situation when querying it. So, using such a model can restrict the full flexibility of the semi-structured part. Also, a small change of the structure in part of the document could lead to a series of complex changes in the equivalent relational structure. For example, if the author element in the document changed from one author (this means one field inside a record) to multiple authors (a new table to hold these multiple authors' names must be added to the equivalent relational structure).

The important issue is how schema mapping can be executed. Basically, there are three approaches to map XML documents to a relational presentation:

- From the XML data itself, by using data mining techniques to extract the structure from within a collection of XML documents, such as STORED system (Deutsch et al. 1999), see section 3.5.3.
- From DTD (or XML Schema), by analysing the DTD structure and converting it to a relational structure. Examples of this approach include the basic inlining technique, shared inlining, hybrid inlining (Shanmugasundaram et al. 1999), the new inlining (which is an improvement to the shared inlining technique, Lu et al. 2003), DOM-based approach (which is based on the new inlining technique, Atay et al. 2004 and 2007) and mapping DTD to relational with semantics constrains (such as functional dependences, domain constrains, choice constrains, reference constrains and cardinality constrains, Lv and Yan 2006). A similar approach is to map XML Schema to ER diagrams (Penna et al 2006), and then map the ER diagram to a relational structure.

Using a manual approach by analysing the XML document and/or its DTD, if it exists, and creating an equivalent relational structure. This approach follows the same normal procedure for database design and therefore it is a subjective approach. Two database designers can produce two different and valid schemas for the same XML document. The other important factor in this approach is the query workload; this could lead to some changes in the schema design, for example, by de-normalising parts of the

schema, rather than fully normalising it, or by using alternative modelling capabilities such as sub-types (Elmasri and Navathe 2006). The first two automated approaches can be adapted manually later on to put the final touches to the automated, generated, equivalent relational schema.

This research uses the manual approach. This suits the partially-structured document collection better since it gives more control and flexibility over categorizing the document into highly-structured and semi-structured parts. The XML document used and its DTD are fully analysed to produce such a schema to host XML data into the relational structure.

4.2.4.2 Store the Whole XML Document as an XML Data Type

As presented in section 4.2.3, storing XML documents as an XML data type (SQL:2003, SQL:2006 (ISO/IEC 9075, part 14, XML-related specifications - SQL/XML), Eisenberg and Melton 2004, Krishnaprasad et al. 2005) is an emerging approach. It is promising but its performance is yet to be tested. This is because this feature is still emerging and recently released in new software versions releases (such as MS SQL Server 2005 which released in December 2005).

MS SQL Server 2005 provides not only the storage of XML inside an XML data type, but it implements other parts from the SQL/XML standard. It supports indices for XML data inside XML data type as well as using XQuery to query XML data (Pal et al. 2005 and 2006, Rys 2005, Rys et al. 2005 and Lacoude 2006).

So, storing an XML document as a new XML data type is used as the second base storage model in the experiment, including the utilization of indices provided by the DBMS to improve query performance. In this approach, the structured part of the document is ignored and the whole document is stored as a semi-structured document. That is to say, the document is dealt with as if it is 100% semi-structured.

4.2.4.3 The Proposed Hybrid Approach

The proposed hybrid model is a combination of the above two base models. The idea behind this model is to take the advantage of both the two base systems, as described in 3.6. Mapping the structured part into a relational structure allows the use of relational query optimisation for the highly-structured part of the data. Mapping the semi-

structured part to XML data type allows more flexibility when dealing with this part of the document.

The aim of the experiments is to test the query performance of the proposed models against the two base models, and to determine weather this hybrid model has any advantages/disadvantages compared with its two base models. This entails the benchmarking of the performance of the proposed model, to compare with the other data models. Instead of creating a benchmark from scratch, one of the available benchmark techniques in the XML data management research field was adapted to support the research aim. This is considered appropriate given that the aim of existing benchmarks is to compare the relative performance of different system to store and query XML documents. The chosen benchmark was adapted in view of the specific features of the experiments. This adaptation is explained in full in section 4.4. In the next section, the different benchmark systems are explained, followed by a comparison of these systems and the selection of the benchmark system chosen for the experiments.

4.3 XML Benchmarking

In the field of database benchmarks, there are a number of benchmarking techniques used to measure the performance of a specific application or domain. The Transaction Processing Performance Council (http://www.tpc.org) provides the database industry with a number of these domain-specific benchmarking such as TPC-C for online transaction processing, TPC-H, TPC-R for decision support systems and TPC-W for transactional web e-commerce benchmarking. Database researchers provide a number of other benchmarks such as the Wisconsin benchmark (DeWitt 1993) and the OO7 Benchmark which represents a comprehensive test of OODBMS (Carey et al. 1993).

XML DBMS Benchmarking - as a domain specific benchmarking - receives a lot of attention in the XML data management research field because of the need to compare the performance of different XML data management systems. This is due to the increase usage of XML documents and the variety of different models and techniques used to store and query XML databases and therefore the need to compare their performances. Although one of the primary uses of XML is in web e-commerce transactions, TPC-W, as a transactional web e-commerce benchmark, was not sufficient by itself to benchmark XML databases because of the complexity of XML DBMS activities many

of which are not covered in what TCP-W measures, i.e., the number of web interactions processed per second.

XML benchmarking can be divided into two main categories; the first is application benchmarking, in which the benchmark is designed to measure the overall performance of any XML database. There are four major XML benchmarks in this category in the XML data management research field. They are XMach-1 (Böhme and Rahm 2001, 2002), XMark (Schmidt et al. 2002), X007 (Brassan et al. 2002), and XBench (Yao et al. 2002, 2003, 2004). The second category is the micro benchmark. This class is designed to test basic query system components such as selections, joins and aggregations inside the XML DBMS. Testing these individual components leads to the ability to isolate any problem and improving such components leads to an overall improvement of the XML DBMS performance. The Michigan Benchmark (Runapongsa et al. 2002a, 2006) is an example of this class.

The following subsections describe each of these benchmarks in more detail followed by a comparison of them all.

4.3.1 XMach-1

Böhme and Rahm proposed XMach-1 in 2001. It is a multi-user benchmark for XML data management system. XMach-1 is based on a web application scenario. This system uses a multiple documents data set. The majority of documents in this scenario are document-centric (such as books or essays) as well as a minority of data-centric document which contains meta data about documents such as document Id, URL, insert and update times. XMach-1 consists of eight retrieval queries as well as three update operations (one insert, one delete and one update operation).

4.3.2 XMark

Schmidt et al. (2002) proposed XMark. It is a single-user benchmark for an XML data management system. XMark is based on an internet auction web site. So, it uses a single big XML document. The majority of the elements inside this single XML document are data-centric which cover the data of the items, the bids and so on. While the minority cover document-centric which are the textual description of items (which is a good example of a partially structured document). XMark consists of twenty different

retrieval queries which cover more aspects than XMach. It does not have any update queries.

4.3.3 XOO7

Brassan et al. (2002) developed XOO7 based on the OO7 object oriented database management system benchmark (Carey et al. 1993) with some minor changes to both data structure and additional operation types to cover XML usage patterns. XOO7 is a single-user benchmark for an XML data management system. XOO7 is not based on a specific application domain; it rather focuses on a single XML document with a majority of the components represented as complex objects using component-of relationship. So, the majority of the elements inside this single XML document are data-centric, while the minority are document-centric, represented by elements with mixed content (yet again another good example of a partially structured document). XOO7 consists of twenty three different retrieval queries and it does not have any update queries.

4.3.4 XBench

Yao et al. (2002, 2003, 2004) developed a family of XML benchmarks (XBench). It is a single-user benchmark for XML data management systems. What really makes XBench different from the others, is that it does not consider only one type of application, it covers four different types by classifying the application into two different classes (Data-Centric (DC) and Text Centric (TC)) and also classifying the data into two different classes (Single document (SD) and multiple documents (MD)). Using these two classifications will produce four different designs (TC/SD, TC/MD, DC/SD, DC/MD). The second feature which makes XBench unique among the others is that it covers all the XQuery (Online 2003) features. So, XBench deals equally between the different classes of XML documents.

4.3.5 The Michigan Benchmark

Runapongsa et al. (2002a, 2006) developed the Michigan Benchmark as a micro benchmark. It is currently the only one in this category of benchmarking in the XML database research field. This type of benchmarks is designed to test individual system characteristics. And by doing that, this could allow the system designer to identify the parts of the system that needs more attention and more development, for example, basic

query components such as selections, joins and aggregations inside the XML DBMS. The queries are categorised into six main categories (Selection, Value-Based Join, Pointer-Based Join, Aggregation and Update Queries, Aggregation, Update) with a number of different queries in each category.

4.3.6 A Comparison between Different XML Benchmarking Techniques

The following table is a comparison between these five XML benchmarks. It is compiled based on two different comparisons in Yao et al. (2002a) and Böhme and Rahm (2003).

	XMach	XMark	X007	XBench	Michigan
Scope	Application	Application	Application	Application	Micro
Users	Multi-user	Single-User	Single-User	Single-User	Single-User
DB Size	16KB * No of documents	10MB – 10 GB	4MB – 10 GB	Various, depending on domain	50MB * 10n where n=1,2,3,4
Document Environment	Multiple Documents	Single Document	Single Document	Both	Single Document
DTD Support	Multiple, same structure	One	One	Multiple	One
XML Schema Support	No	No	No	Yes	Yes
No of Queries	8	20	23	20	49
No of Update Operations	3	-	-	-	7
All XQuery Features	No	No	No	Yes	N/A since it tests the functionality of the core query language

Table 4.1 A Comparison between Different XML Benchmarking Techniques

For the purpose of this research, the Michigan Benchmark is excluded, since the main research aim is not to develop a new query engine, and therefore the micro benchmark is not applicable. XBench is the only one that supports both single document and multiple document scenarios. It is also support all the different XQuery use cases. The following table shows in details the XQuery use cases (Yao et al. 2003).

Query Functionality	XMach-1	XMark	XOO7	XBench
Exact Math				
Shallow	-	-	Q1	Q1
Deep	Q1	Q1	-	Q2
Function Application	Q3,Q7	Q18,Q20	Q3,Q7,Q15	Q3
Ordered Access				
Relative	Q8	Q2-Q3	Q2	Q4
Absolute	-	Q4	Q17-Q18	Q5
Quantifier				
Existential	Q8	-	Q14	Q6
Universal	-	-	-	Q7
Regular Path Expressions				
Unknown Element	Q2	Q15-Q16	-	Q8
Unknown Sub path	Q4-Q5	Q6-Q7	-	Q9
Sorting				
By String	Q8	Q19	-	Q10
By Non-string	-	-	Q8	Q11
Document Construction				
Structure Preserving	-	Q13	Q16	Q12
Structure Transforming	-	Q10	Q6, Q9, Q12,Q16	Q13
Irregularity				
Missing elements	-	-	-	Q14
Empty (Null) Values	-	Q17	-	Q15
Retrieve Individual Documents	Q1	-	-	Q16
Text Search				
Uni-gram Search	-	Q14	-	Q17
N-gram Search	Q2	-	Q5	Q18
Reference and Joins	Q1-Q2, Q6	Q8-Q9, Q11-Q12	Q10-Q11, Q15, Q17-Q18	Q19
Data Type Cast	-	Q5	-	Q20

Table 4.2 XQuery Use Cases

The application benchmark system which is used in this research is the XBench. This is mainly because it covers all the XQuery use cases as in (XQuery use case, online) as XQuery becomes the *de facto* query language for XML. Also, XBench provides the flexibility of dealing with different document scenarios in four different ways (TC/SD, TC/MD, DC/SD, DC/MD) compared with other benchmarks which deals with only one scenario.

Applying XBench is not directly applicable to the research problem since it involves partially-structured data. This is mainly because of the features of partially-structured data that needs to be tested when designing the experiment (for example, dealing with differently structured parts inside the document). The next section shows how this benchmark can be adapted.

4.4 Adapted XBench

This section shows the adaptation of the XBench benchmark to support the aims of the experiment. Section 4.6.1 describes the data set proposed by XBench and then presents the data set used in the experiment. Following from that, section 4.6.2 describes the query set proposed by the XBench and its adaptation for testing partially-structured XML documents.

4.4.1 Data Set

As explained in section 4.3.4, XBench is based on four different scenarios. These scenarios are:

- Data-Centric – Single Document
- Data-Centric – Multiple Document
- Text-Centric – Single Document
- Text-Centric – Multiple Document

XML documents are generated using ToXGene (Barbosa et al. 2002). The following parameters characterise each generated XML document (Yao et al. 2002)

- Elements Types: presents the collection of elements' types to be used in the generated document.
- Tree structure of element types: shows the relationship between element types. For example if there is a parent/child relation between two elements.
- Distribution of children to elements: shows the probability distribution of child elements for each type (directly sub-element types).
- Distribution of element values to types: shows the probability distribution of values of each element type.

- Attribute names: present the collection of attribute names to be used in the generated document.
- Distribution of attribute values to name: shows the probability distribution of values of each attribute
- Distribution of attributes to elements: shows the probability distribution of the attributes to each element.

The minimum and maximum of each distribution parameter is defined in order to generate finite documents with specific distribution.

Rather than adopting the above approach of synthesising data sets, the experiments were conducted using the large XML document from DBLP (Digital Bibliography & Library Project http://dblp.uni-trier.de/xml/). The DBLP is bibliographic information specifically in computer science journals and proceedings. The characteristics of this bibliographic database can be found in Reuther et al. (2006) and Ley and Reuther (2006). The rational for using this data set in the experiments is:

- It consists of one XML document (so it suits the single document scenario adopted by this research, section 3.2.1)
- The data conforms to a DTD from which it was possible to derive the structure for all or part of the data set though analysis of this DTD. This allowed flexibility in varying interpretations of the document so as to simulate varying the ratios between semi-structured and highly-structured content instead of dealing with the whole data set as totally highly-structured, totally semi-structured or partially-structured. This point is discussed in more detail later in this section.
- It is a large document (More than 750,000 publications, 450,000 authors stored in 335 Megabyte as of September 2006). This means that the performance results are credible.
- It is widely used in XML database research as a model for a bibliography data (such as Elmacioglu and Lee 2005, Low et al. 2002, Ley 2002 and Reuther et al. 2006), thus allowing a possible comparability of the results.
- Finally, the use of "natural", rather than artificially synthesized data sets adds to the validity of the research.

There are two important points that shape the implementations used in the experiment. The ratio between semi-structured and highly-structured data parts inside the XML document and the use of typed and un-typed XML data fields.

Firstly, the structure of the DBLP XML document (its DTD is shown in Figure 4.1) consists of articles, in-proceedings, proceedings, books, in-collections, PhD Theses, Master Theses and WWW sites. The majority of publications are conference papers (in-proceedings) (about 60% of all the publication[1]) followed by articles in journals (about 37% of all the publication[2]). While the remaining represent the other forms of publication.

One aim of the experiments is to determine the effect of the relationship between the ratio of semi-structured to highly-structured data within an XML document and the query performance for a range of storage strategies. Accordingly, within the experiment, the structure of the DBLP data set is interpreted in two different dimensions to vary this ratio. The first dimension, which is referred to as the vertical dimension because of the conventional tabular representation of data in which schema elements and their instances are denoted as columns, concerns the ratio of semi-structured to structured components of the schema. This dimension can be seen as a schema dimension. The second, which can be called the horizontal dimension, is the ratio of semi-structured to structured data instances. This dimension can be seen as the data instances dimension.

In the first vertical scenario, only the document key is considered as highly-structured while the reminder of the data is stored in an XML data field, as such the remaining data is considered to be semi-structured. This is annotated as '***PSD***', a Partially-Structured model with only the 'Document key' only as highly-structured data. In the second vertical scenario, both 'document key' and 'author' data were treated as highly-structured while the rest of the data was considered as semi-structured. This is annotated as '***PSDA***', a Partially-Structured model with 'Document key' and 'Author' data as highly-structured. In the final vertical scenario, both 'document key' and 'title' data were treated as highly-structured while the rest of the data was considered as semi-structured. This is annotated as '***PSDT***', a Partially-Structured model with 'Document key' and 'Title' data

---

[1] Number of in-proceedings publications represent 60.28% of the total number of publications in this document. This does not means that it represents 60.28% of the storage size of the document, since each publication data is different in its storage size from the others, for example, a Phd Thesis has one author while an article could have one or more authors.
[2] Number of articles publications represent 37.15% of the total number of publications in this document.

as highly-structured data. In this way, the data was interpreted in different ways so as to explore the effect on performance of the ratio of structured to semi-structured data definitions within the schema.

In order to investigate the impact of the second "horizontal" scenario, all in-proceedings publications were treated as semi-structured data, while the rest of the publications were considered as highly-structured. This is annotated as ***'60%XW'***, as the in-proceedings publications represented approximately 60% of the data total number of publications (approximately 60% of the whole data set). In the second "horizontal" scenario, all the article publications were treated as semi-structured data while the remaining publications were considered as highly-structured. This is annotated as ***'37%XW'***, as the articles publications represented approximately 37% of the total number of publications (approximately 37% of the whole data set). In these two scenarios, the document key is stored in the relational table, which means the document key is considered as highly-structured while the remaining data (such as authors, titles, URL...) are considered as semi-structured. In this way, the same data was interpreted in different ways so as to explore the effect on performance of the ratio of structured to semi-structured data instances. Dividing the data in the horizontal dimension as 60% and 37% are a specific to the DBLP data set and it maybe not possible in another case to divide the data in the horizontal dimension with the same percentage. Nevertheless, partitioning the data in the horizontal dimension showed that in some cases there is an advantage to divide the data in a similar manner, possibly with different percentages as opposed to dealing with the data as totally highly-structured or totally semi-structured.

MS SQL Server 2005 can store XML data (either document or content fragments of the document) in two different ways, un-typed and typed XML data fields (MS SQL Server web site). In the un-typed scenario, there is no XML schema associated with the document, therefore SQL Server only checks that the document (or the content fragments) is well formatted, that is to say, it can accept any well formatted XML document or document extract. This scenario is useful when the schema is not known at the design stage. In the Typed XML data type, the XML data must conform to an XML schema defined beforehand. This associated schema is used to validate the data, perform more accurate type checks and to optimize both storage and query processing. The early results of the experiments showed that the un-typed XML field performance was extremely un-reliable. Therefore the experiments include only testing the typed XML

data field. More information about SQL Server can be found in (Pal et al. 2005 and 2006, Rys 2005, Rys et al. 2005 and Lacoude 2006).

Therefore, and based on the above discussion, three storage models are tested in this experiment with seven implementations as follows:

- Structured Mapping Approach: 100% of the data is mapped to a relational structure. This is annotated as ***100%R***.
- XML data field: 100% of the data is mapped to one typed XML data field. This is annotated as ***100%XW*** ('W' refers to that data With schema).
- Proposed hybrid model: with five implementations so as to vary the ratios, respectively, of semi-structured to structured data instances and schema. In the first implementation, the document key is treated as highly-structured and this model is annotated as ***PSD***. In the second implementation, the document key and author are treated as highly-structured and this model is annotated as ***PSDA***. In the third implementation, the document key and title are treated as highly-structured and this model is annotated as ***PSDT***. In the final two implementations, approximately 60% and 37% of the data are mapped as semi-structured data to a typed XML data field and the rest are mapped as highly-structured. These are annotated as ***60%XW*** and ***37%XW*** respectively.

The final important factor related to the data set is its size. The DBLP document is considered as a large XML document (335 Megabyte). To study the effect of the data set size, the experiments are conducted in three different databases; the first is on the whole data set (coded as ***DB3/3***), the second is on approximately two-thirds of the whole data set by deleting one third of the in-proceedings and one third of the articles (coded as ***DB2/3***). Finally, the third data set consists of approximately one third of the whole data set by deleting two thirds of the in-proceedings and two thirds of the articles publications (coded as ***DB1/3***). This allows the results to be compared over different data set sizes (approximately 33%, 66% and 100% of the original data set size).

```
<!ELEMENT dblp (article|inproceedings|proceedings|book|incollection|
                phdthesis|mastersthesis|www)*>
<!ENTITY % field
"author|editor|title|booktitle|pages|year|address|journal|volume|number|m
onth|
url|ee|cdrom|cite|publisher|note|crossref|isbn|series|school|chapter">
<!ELEMENT article        (%field;)*>
<!ATTLIST article        key CDATA #REQUIRED
                         reviewid CDATA #IMPLIED
                         rating CDATA #IMPLIED
                         mdate CDATA #IMPLIED>
<!ELEMENT inproceedings (%field;)*>
<!ATTLIST inproceedings key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>
<!ELEMENT proceedings    (%field;)*>
<!ATTLIST proceedings    key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>
<!ELEMENT book           (%field;)*>
<!ATTLIST book           key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>
<!ELEMENT incollection   (%field;)*>
<!ATTLIST incollection   key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>
<!ELEMENT phdthesis      (%field;)*>
<!ATTLIST phdthesis      key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>
<!ELEMENT mastersthesis (%field;)*>
<!ATTLIST mastersthesis key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>
<!ELEMENT www            (%field;)*>
<!ATTLIST www            key CDATA #REQUIRED
                         mdate CDATA #IMPLIED>
<!ELEMENT author     (#PCDATA)>
<!ELEMENT editor     (#PCDATA)>
<!ELEMENT address    (#PCDATA)>
<!ENTITY % titlecontents "#PCDATA|sub|sup|i|tt|ref">
<!ELEMENT title      (%titlecontents;)*>
<!ELEMENT booktitle (#PCDATA)>
<!ELEMENT pages      (#PCDATA)>
<!ELEMENT year       (#PCDATA)>
<!ELEMENT journal    (#PCDATA)>
<!ELEMENT volume     (#PCDATA)>
<!ELEMENT number     (#PCDATA)>
<!ELEMENT month      (#PCDATA)>
<!ELEMENT url        (#PCDATA)>
<!ELEMENT ee         (#PCDATA)>
<!ELEMENT cdrom      (#PCDATA)>
<!ELEMENT cite       (#PCDATA)>
<!ELEMENT school     (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ATTLIST publisher  href CDATA #IMPLIED>
<!ELEMENT note       (#PCDATA)>
<!ATTLIST cite        label CDATA #IMPLIED>
<!ELEMENT crossref   (#PCDATA)>
<!ELEMENT isbn       (#PCDATA)>
<!ELEMENT chapter    (#PCDATA)>
<!ELEMENT series     (#PCDATA)>
<!ATTLIST series     href CDATA #IMPLIED>
<!ELEMENT layout     ANY>
```

```
<!ATTLIST layout   logo CDATA #IMPLIED>
<!ELEMENT ref (#PCDATA)>
<!ATTLIST ref href CDATA #REQUIRED>
<!ELEMENT sup (%titlecontents;)*>
<!ELEMENT sub (%titlecontents;)*>
<!ELEMENT i   (%titlecontents;)*>
<!ELEMENT tt  (%titlecontents;)*>
```

Figure 4.1 DBLP DTD (http://dblp.uni-trier.de/xml/dblp.dtd)

Using an existing data set gives the experiments more credible results, because this is a real life data set. However, the disadvantage of using existing data (such as DBLP) is that, it does not provide a range of partially-structured data sets with different degrees of structuredness, as would be possible if the data sets were synthesised. Instead, since the separation between highly-structured and semi-structured data is not naturally defined inside the data set, it is defined artificially.

This point was taken into consideration when designing the above data set by having two ways to interpret the structuredness of the data (vertically and horizontally) as discussed earlier. It was also taken into consideration when designing the query set, as is explained in section 4.4.2.

In Appendix B, the full SQL script to create the tables used in the experiments is presented. The following subsections show the schema design for the above scenarios. These schemas are exactly the same for the three database sizes (coded ***DB1/3***, ***DB2/3*** and ***DB3/3***).

4.4.1.1 Schema Design for Structured Mapping Approach

In this model, 100% of the data is mapped to a relational structure. SQL is used to query this model. The relational schema representing this model consists of the following tables (with the A_ prefix):

A_Doc (DocId, DocTypeId, MDate, DocKey, ReviewId, Rating)

A_Doctype (DoctypeId, DocType)

A_Doc is the main table which contains all the publications. *DocKey* is a unique value for each document. There are eight different document types represented in the A_*DocType* table. The other tables in this schema are:

A_Address (Id, DocId, Address)

A_Author (Id, DocId, Author)

A_BookTitle (Id, DocId, BookTitle)

A_CDROM (Id, DocId, CDROM)

A_Chapter (Id, DocId, Chapter)

A_Cite (Id, DocId, Cite)

A_CrossRef (Id, DocId, CrossRef)

A_Editor (Id, DocId, Editor)

A_EE (Id, DocId, EE)

A_ISBN (Id, DocId, ISBN)

A_Journal (Id, DocId, Journal)

A_Month (Id, DocId, Month)

A_Note (Id, DocId, Note)

A_Number (Id, DocId, Number)

A_Pages (Id, DocId, Pages)

A_Publisher (Id, DocId, Publisher)

A_School (Id, DocId, School)

A_Series (Id, DocId, Series)

A_Title (Id, DocId, Title)

A_URL (Id, DocId, URL)

A_Volume (Id, DocId, Volume)

A_Year (Id, DocId, Year)

In all these tables, *DocId* is a foreign key linked to the main *A_Doc* table. All the possible indices are used to improve the query performance. For example, in the Author table, there are three indices on the Id, DocId and the Author fields. In the title table, there are three indices on the Id, DocId and the Title fields.

4.4.1.2 Schema Design for Using an XML Data Field

In this model, 100% of the data is mapped to an XML data field. The relational schema representing this model consists of one table (its name is *B_XMLDocumentWithSchema* in the ***100%XW***). The table structure is:

B_XMLDocument (DocId, XMLDoc)

This table contains one record only, where the whole XML document is loaded into the *XMLDoc* field. XQuery is used to query this model.

The indices used in this mode and supported by MS SQL Server 2005 (Pal et al. 2005 and 2006, Rys 2005, Rys et al. 2005 and Lacoude 2006) are:

- Primary XML index: a B+ tree and similar to a primary key in a relational table. It can be seen as a hidden table which contains PK, XML node values, XML node paths, node types, document order and other relevant information. This primary index must exist before creating any of the following secondary indices.
- Value Secondary XML index: a B+ tree on the value column of the primary index. This index is used for value-based queries.
- Path Secondary XML index: a B+ tree on the path column of the primary index. This index is used for path-based queries.
- Property Secondary XML index: It is a B+ tree built on PK, path and node value). This index is used when retrieving object properties (attributes).

All the above indices are used in this storage model as the experiments are concerned with the query performance, and using such indices can enhance this model's query performance.

4.4.1.3 Schema Design for the Proposed Model

There are five storage representations for the proposed model (as described in section 4.4.1). The first scenario defines the in-proceedings as the semi-structured part (for the systems ***60%XW***) with the prefix *C_* and with the postfix *WithSchema* for the typed XML data field. The second scenario defines the articles as the semi-structured part (for the system ***37%XW***) with the prefix *D_* and with the postfix *WithSchema* for the typed

XML data field. A combination of SQL and XQuery is used to query this model depending on the situation. This is described in the beginning of each query group.

These storage models have the same schema as the structured mapping approach except there is an extra field in the _DocWithSchema table which is an *XMLExtract:*

C_Doc (DocId, DocTypeId, MDate, DocKey, ReviewId, Rating, XMLExtract)

XMLExtract is an XML data type. It contains the in-proceedings element which represents semi-structured data in the *C_DocWithSchema* table and contains the articles element in the *D_DocWithSchema* table. All in-proceedings data are deleted from the other *C_* tables (*C_Address, C_Author ...)* since that data is stored as an XML data field in the *XMLExtract* field. The same procedure was applied for the articles data in the other *D_* tables.

All the indices used in the storage mapping approach are used in the above two scenarios. Adding to them, XML indices are used for the *XMLExtract* field. There is a primary index as well as value, path and property secondary indices on the two tables (C_DocWithSchema and D_DocWithSchema).

For the other three systems (namely ***PSD, PSDA*** and ***PSDT***), the queries were adapted to retrieve their data from the same tables (with prefix A_ and C_ or D_) as need arises. In that sense, they do not have their own table structures. For example, searching for in-proceedings' title in the case of ***PSDA***, *A_Author* table is used to search for an 'author'. Then if the document type is 'in-proceedings', table *C_DocWithSchema* is used to retrieve the 'title' data while if the document type is 'article', table *D_DocWithSchema* is used.

4.4.2 Query Set

In this section, the query set used by XBench is presented and then adapted to the purpose of this experiment. The previous section described, how the publications are classified inside the document (in-proceedings publications presented semi-structured data and the remaining publications presented highly-structured data in the *C_* schema, while the articles publications presented semi-structured data in the *D_* schema), the two types of publications that are used in the experiments are in-proceedings and articles (as both together represent 97.43 of all publications).

The queries were designed to query only the semi-structured part of the document. This is due to the fact that all the hybrid models use the same structure mapping approach to store highly-structured data that is used in the ***100%R*** model, and therefore will produce the same performance as the ***100%R*** model.

The following subsections show each query group within the XBench query set and how it has been adjusted to query the experiments' data set.

4.4.2.1 Exact Match

The queries in this group require string exact match with specified and possibly long path expressions, depending on the levels of predicates being queried in XML documents. Consequently, they can be shallow queries that match only at the top level of XML document trees (example query Q1), or deep queries that match the nested structure of XML document trees (query Q2).

This group is adapted as follow:

- *Q1*: Return in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables) that have the key value X.
- *Q2*: Return in-proceeding's titles in 'C_' tables (or an article's titles in 'D_' tables) that have the same author X.

4.4.2.2 Function application

The query in this group challenges the system with aggregate functions such as count, avg, max, min and sum. This group is adapted as follows:

- *Q3*: counts in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables) that have the same author X.

4.4.2.3 Ordered access

The queries in this group test the performance of the system when it preserves the document order during retrieval. This could be a relative order (Q4) based on the current matching position, or an absolute order (Q5), which is the order in the document. This group is adapted as follows:

- *Q4*: Return in-proceeding's titles in 'C_' tables (or an article's titles in 'D_' tables) that have the same author X. In-proceeding's titles in 'C_' tables (or

article's titles in 'D_' tables) that have the same author X ordered by the relative order in the original document.

- *Q5*: Return the first in-proceeding's title in 'C_' tables (or article's title in 'D_' tables) that has the same author X order by their absolute ordered in the original document.

4.4.2.4 Quantification

The queries in this group test the existentially (Q6) and universally (Q7) quantified queries. This group is adapted as follow:

- *Q6*: Return in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables) that exist among their authors for two specific authors (author X and author Y)
- *Q7*: Return in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables) that have exactly two authors (author X and author Y)

4.4.2.5 Path expressions

The queries in this group involve path expressions: Q8 queries data where one element name in its path is unknown. Q9 queries data where multiple consecutive element names in its path are unknown. This group is adapted as follow:

- *Q8*: Return in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables) that contain word XYZ among their title.
- *Q9*: Return in-proceeding's authors in 'C_' tables (or article's authors in 'D_' tables) for a specific publication that has a key value X. The (//) is used inside this query to direct the query to look for any path inside the document.

4.4.2.6 Sorting

Even though the generic data type of element content in XML documents is string, users may cast the string type to other types. Therefore, the queries in this group test sorting both string types (Q10) and non-string types (Q11). This group is adapted as follows:

- *Q10*: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors sorted by title for specific author.
- *Q11*: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors sorted by publication date for specific author.

4.4.2.7 Document construction

Structure is important in many XML documents. However, some systems experience difficulties in even preserving the document's original structure. This class of queries tests the performance of the system in preserving the structure (Q12) and in transforming the structure (Q13). This group is adapted as follows:

- ***Q12***: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors for specific document preserving the original document structure.
- ***Q13***: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors for specific document transforming the original document structure to another structure.

4.4.2.8 Irregular data

Irregularity is common in XML documents. This class of queries tests missing elements (Q14) and empty (null) value (Q15). Since there are no empty elements in the data set used in this experiment, Q15 tests that the year equals a specific number instead of it equalling null. This group is adapted as follows:

- ***Q14***: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors where '*ee*' element is missing.
- ***Q15***: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors where the '*year*' element equals a specific number.

4.4.2.9 Retrieval of individual documents

The query in this group tests an essential function of an XML DBMS to retrieve individual XML documents (or an XML extract in this case) while preserving the contents of those documents. This group is adapted as follows:

- ***Q16***: Retrieve in-proceeding data in 'C_' tables (or article data in 'D_' tables) that has a key value X keeping its original structure.

4.4.2.10 Text search

These queries test the information retrieval capabilities of systems. Two cases are tested: uni-gram search (Q17) where the query contains one particular word and bi-gram and n-gram search (Q18) where multiple words are involved. This group is adapted as follows:

- *Q17*: Search for the word XYZ in any field in the in-proceeding data in 'C_' tables (or article data in 'D_' tables).
- *Q18*: Search for the phrase XX YY ZZ in any field in the in-proceeding data in 'C_' tables (or article data in 'D_' tables).

4.4.2.11 References and Joins

Data-centric documents usually have references to identify the relationship between related data, even among different XML documents. Sometimes users want to combine separate information using join-by values. The original version of this query tests the references and joins. Since the data set consists of a single XML document, and as this query group requires joining two documents, it is executed in two phases, in the first phase it retrieves an author of a specified article (or in- proceeding) then using this author name, another query is executed to retrieve all the articles he/she wrote. This group is adapted as follow:

- *Q19*: Retrieve the first author for in-proceeding data in 'C_' tables (or article data in 'D_' tables) that has a key value X. Using this author; retrieve all his publication's titles.

4.4.2.12 Datatype casting

The element values in XML documents are of a string type, but sometimes there is a need to cast them into other data types. This group is adapted as follows:

- *Q20*: returns all in-proceedings' titles 'C_' tables (or articles' titles in 'D_' tables) where their title's length is longer than a specific random size.

In this section, the different queries that are used in the experiment are described and adapted toward the aim of testing partially-structured data. The following section presents the performance metrics for the experiments.

4.5 Performance Metrics

As the experiments' aim is to test the relative performances between different storage models, each execution of an experiment records the following measures:

- Execution time: for each query, how many milliseconds are needed to get the result
- CPU busy time: the time (in milliseconds) that is consumed by the CPU to run this query
- IO busy time: the time (in milliseconds) that is consumed in IO related operations to run this query
- IO read: number of disk reads by SQL Server to run this query
- IO write: number of disk writes by SQL Server to run this query

The following figure shows a template to run each of the experiment's queries

```
Declare             @Clock DATETIME,
                    @CPU BIGINT,
                    @IO BIGINT,
                    @IOr BIGINT,
                    @IOw BIGINT,
                    @ExecutionTime BIGINT
SELECT              @Clock = GetDate(),
                    @CPU = @@CPU_BUSY,
                    @IO = @@IO_Busy,
                    @IOr = @@TOTAL_Read,
                    @IOw = @@Total_Write
/* Q1A, The Query syntacs will be executed here For Storage System
100%R*/
Select              @ExecutionTime = DateDiff(ms, @Clock , GetDate()),
                    @CPU = (@@CPU_Busy - @CPU) * @@TimeTicks,
                    @IOr = @@TOTAL_Read - @IOr,
                    @IOw = @@Total_write - @IOw,
                    @IO= (@@IO_Busy-@IO ) * @@TimeTicks
If @CPU >= 0 Print 'CPU:' + str(@cpu /1000.0 , 8, 0) + ' millisecond'
If @IO >= 0  Print 'IO Busy:' + str(@IO /1000.0, 8, 0) + '
millisecond'
If @IOr >= 0 Print 'IO reads:' + str(@IOr, 8, 0)
If @IOw >= 0 Print 'IO write:' + str(@IOw, 8, 0)
Print 'Execution time:' + str(@ExecutionTime,8,0)+ ' millisecond'
/* The following Stored Procedure will store these results for Q1A,
for the system code 100%R */
Execute StoreResults  'Q1', '100%R', @ExecutionTime, @CPU, @IO, @IOr,
@IOw
```

Figure 4.2 Query Template

The final line shows the *StoreQueryResults* stored procedure which stores all the above data for each query run in the following table:

Results (Id, QueryRunDate, QueryCode, SystemCode, ExecutionTime, CPU, IO, IOr, IOw)

The QueryRunDate field stores the system's current date and time when this query is executed. Each query runs for twenty times as a minimum. The twenty runs were chosen initially because it is often suggested that this is the minimum sample size necessary for any conclusions based on standard deviations to have much meaning. Two runs are excluded (the maximum and minimum execution time, the maximum was always the first cold run) and then the average of the remaining runs was computed. Standard deviations were calculated for each query and for each storage model, to identify any instability. As the results will be shown in the next chapter, each experiment produced a small standard deviation which reassured that the choice of the twenty runs for each query is sufficient. A target value for the standard deviation was set in advance, that it should not exceed 50% of the mean value. The maximum and the minimum were excluded so as not to skew the results and to eliminate any circumstantial running errors. They might represent anomalous cases, as illustrated by the fact that the maximum was always the first cold run, because of the additional overhead of loading indexes into the system's cache.

The aim of the experiment is to test the relative performance and therefore only the execution time is taken into consideration when analysing the results. Other data is available for further analysis if needed. The analysis of the results is discussed in the following chapter and the full results are presented in appendix C.

4.6 Experimental Operational Environment

This section presents the experimental operational environment within which the experiments were conducted by showing the hardware and the software used.

The experiment runs on a single machine environment. The machine specifications are:

- Intel ® Pentium ® Core™ Duo processor T2250, 1.73 Ghz
- 1536 MB Ram
- 120 GB Hard Disk Drive.
- Microsoft Windows XP Professional 5.1.2600 service pack 2

The software environment is based on Microsoft SQL Server 2005 (Pal et al. 2005 and 2006, Rys 2005, Rys et al. 2005 and Lacoude 2006). This database management system was released on December 2005. Its main features are:

- A Relational Database management system
- A new XML data type, which support the storage of XML documents naturally as an XML data type according to the SQL:2003 standard.
- Includes XML indexing and full-text XML search
- Supports XQuery and XPath.

MS SQL Server's language (Transact-SQL) is used as the main language to access the data and execute the experiment's queries.

4.7 Conclusion

This chapter introduces a series of experiments to evaluate the research hypothesis. These experiments are designed to compare the relative performance of the proposed hybrid model against the two base models it combines. These two models are: mapping the whole XML document to an equivalent relational structure (structure mapping approach) and mapping the whole XML document as an XML data type. The following chapter presents and discusses the experiments' results.

Chapter 5 Experiment Results and Analysis

5.1 Introduction

The goal of the previous chapter was to design a series of experiments to evaluate the research hypothesis. The detailed design of these experiments was discussed. In this chapter, the results of the experiments are discussed. Firstly, Section 5.2 briefly discusses the experiments' environment. This is followed by the results for each of the twenty queries in section 5.3. The overall analysis of these results is presented in section 5.4. The experiments' limitations are discussed in section 5.5 and this is followed by the findings and conclusion in section 5.6.

5.2 Experiments' Environment

This section recaps briefly on the experiments' environment, which includes the data set, the query set, performance metrics and the experiments' operational environment. The detailed discussion of these issues was presented in chapter four.

5.2.1 Data Set

These experiments were conducted using the data set of the DBLP (Digital Bibliography & Library Project http://dblp.uni-trier.de/xml/). The DBLP contains bibliographic information specifically from computer science journals and proceedings. It consists of more than 750,000 publications, 450,000 authors and was stored in 335 Megabyte as of September 2006. The properties of this data set were explained in section 4.4.1.

The first two storage strategies used in the experiments were ***100%R*** which represents mapping the entire DBLP data to the relational data model (see section 4.4.1.1) and the ***100%XW*** which represents mapping the entire document into one XML data field (see section 4.4.1.2).

The third storage strategy used in the experiments was the proposed model (see section 4.4.1.3) with five different implementation scenarios to vary the ratio between the semi-structured and the highly-structured parts in two different dimensions. The first dimension, referred to as the vertical dimension because of the conventional tabular representation of data in which schema elements and their instances are denoted as columns, concerns the ratio of semi-structured to structured components of the schema.

The second dimension, referred to as the horizontal dimension, is the ratio of semi-structured to structured data instances.

In the first vertical scenario, only the document key is considered highly-structured while the remaining data is stored in the XML data field, so the remaining data is considered to be semi-structured. This was annotated as '***PSD***' a Partially-Structured model with the 'Document key' only considered highly-structured data. In the second scenario, both 'document key' and 'author' data were treated as highly-structured while the rest of the data was considered to be semi-structured. This was annotated as '***PSDA***' a Partially-Structured model with 'Document key' and 'Author' data as highly-structured. In the final scenario, both 'document key' and 'title' data were treated as highly-structured while the rest of the data was considered to be semi-structured. This was annotated as '***PSDT***' a Partially-Structured model with 'Document key' and 'Title' data considered highly-structured data.

In order to investigate the impact of the second "horizontal" scenario, all in-proceedings publications were treated as semi-structured data, while the rest of the publications were considered as highly-structured. This was annotated as '***60%XW***', as the in-proceedings publications represented approximately 60% of the total number of publications (approximately 60% of the whole data set). In the second "horizontal" scenario, all the article publications were treated as semi-structured data while the remaining publications were considered as highly-structured. This was annotated as '***37%XW***', as the articles publications represented approximately 37% of the total number of publications (approximately 37% of the whole data set). In these two scenarios, the document key is stored in the relational table, which means that the document key is considered highly-structured data while the remaining data (such as authors, titles, URL...etc) are considered to be semi-structured. As discussed in section 4.4.1, the 37% and 60% are specific to the DBLP data set. In other data sets, there is a possibility to use a different percentage to partition the data differently in the horizontal dimension.

Thus, through these strategies of different interpretation of the structuredness of the data set, the experiments were designed to evaluate the relative performance between these different storage strategies and their different implementations, as the ratio of semi-structured to structured data instances varies, and as the ratio of semi-structured to structure schema elements varies.

MS SQL Server (Pal et al. 2005 and 2006, Rys 2005 and Lacoude 2006) was used as the database management system in the experiments (as explained in section 4.2.3). MS SQL Server can store XML data (either as a document or content fragments of the document) in two different ways, i.e., un-typed and typed XML data fields (MS SQL Server web site). In the un-typed XML data field, data has no XML schema associated with the document. Therefore SQL Server only checks that the document (or the content fragments) is well formatted. In the typed XML data field, the XML data must conform to a pre-defined XML schema. This associated schema is used to validate the data and perform more accurate type checks and to optimise both storage and query processing. The typed XML field was coded as '*W*' (with Schema). (The details of the storage structures used by MS SQL Server for typed and un-typed XML data fields were discussed in section 4.4.1.).

Predictably, un-typed XML gave an extremely poor performance in almost all the query runs (for example, the average execution time for Q2 using the relational model was 148 milliseconds, for the un-typed 100%X it was 43097 milliseconds, for the un-typed 37%X it was 78440 milliseconds and for the un-typed 60%X it was 183616 milliseconds). This was most likely to be due to the lack of schema information, and consequently the database management system could not optimise the query process effectively. Therefore, the results of the un-typed scenario are not considered as a viable option for storing data of the type characterised by the data set, and are not presented or analysed in this chapter. In appendix C, the full results including the un-typed scenarios are presented.

To summarise, three storage models were tested in the experiments with seven implementations as follows:

- Structured Mapping Approach (see 4.4.1.1): 100% of the data was mapped to a relational structure. This was annotated as ***100%R***.
- XML data field (see 4.4.1.2): 100% of the data was mapped to one typed XML data field. This was annotated as ***100%XW***.
- Proposed hybrid model (see 4.4.1.3): with five implementations so as to vary the ratios of semi-structured to structured data instances and schema. In the first implementation, the document key was treated as highly-structured and this model was annotated as ***PSD***. In the second implementation, the document key

and author were treated as highly-structured and this model was annotated as ***PSDA***. In the third implementation, the document key and title were treated as highly-structured and this model was annotated as ***PSDT***. In the final two implementations approximately 60% and 37% of the data was mapped as semi-structured data as typed XML data field and the rest was mapped as highly-structured. These were annotated as ***60%XW*** and ***37%XW*** respectively (see Fig. 5.1).

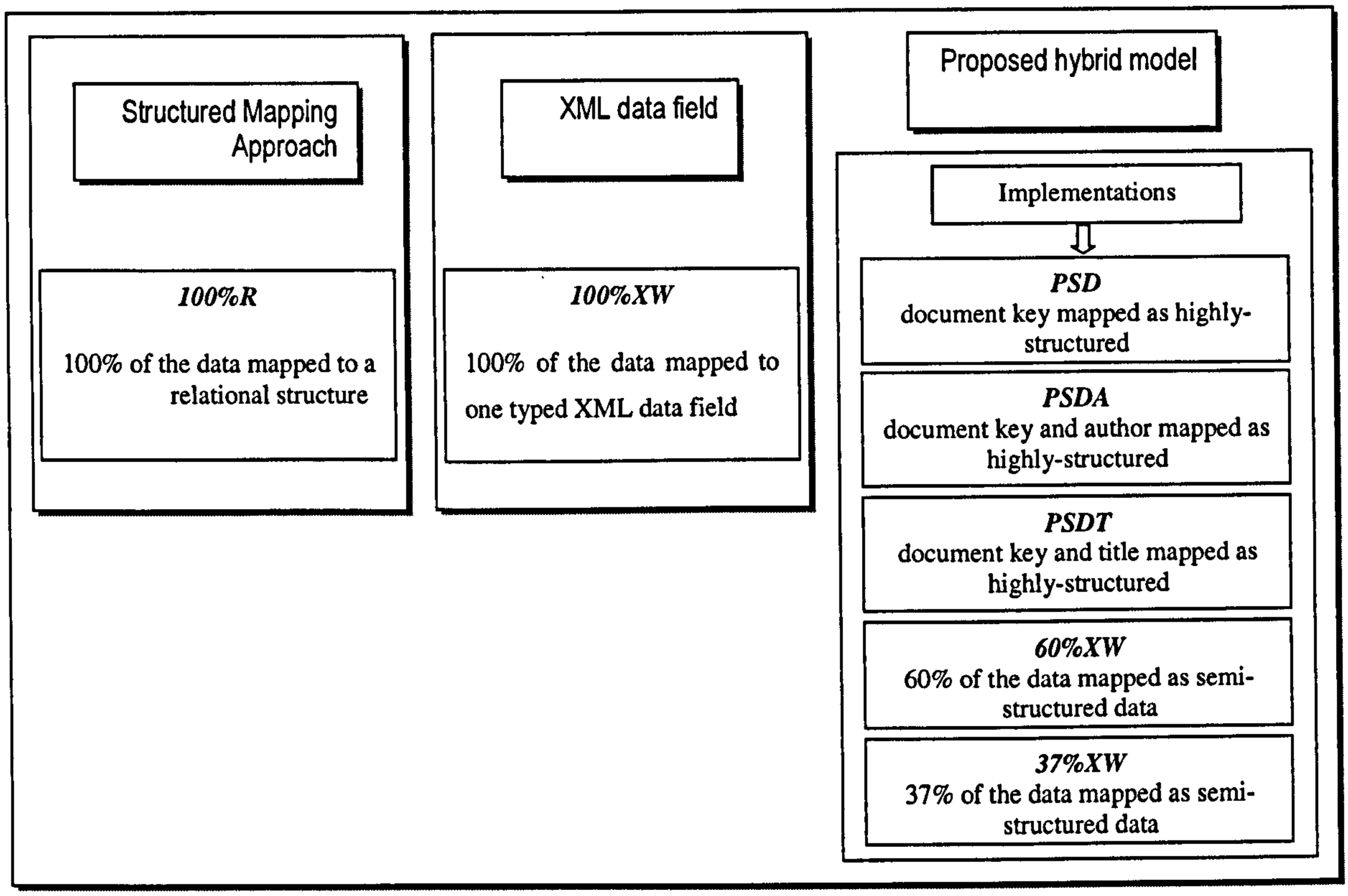


Figure 5.1 Storage models tested in the experiments

Three database sizes were used in the experiments so as to evaluate the impact of data set size, with respect to all of the above variants. The first database was the same as the initial DBLP document (335 Megabyte as of September 2006) and was annotated as ***DB3/3***. The second database was approximately two-thirds of the initial document size and was annotated as ***DB2/3*** and the final database was approximately one-third of the initial document size and was annotated as ***DB1/3***. The three databases had exactly the same data structure. The variation in the database size enables the database size scalability analysis which is explained later in this chapter (in section 5.3.2).

5.2.2 Query Set

The experiments' query set was based on the XBench XML benchmark (Yao et al. 2002). The query set consists of twenty queries (Q1, Q2 ... Q20) grouped as twelve different groups based on their functionality. These groups are: exact match, function application, ordered access, quantification, path expressions, sorting, document construction, irregular data, retrieval of individual documents, text search, references and joins and datatype casting. The main reason behind choosing XBench as the base query set was because it covers all the XQuery use cases as in XQuery use case (online). The full discussion and the rational behind this decision was discussed in detail in section 4.3. These twenty queries were re-formulated to access the data set used in the experiments and for each storage strategy employed. Each of these queries is further explained in section 5.3 in which the results are presented.

5.2.3 Performance Metrics

The main aim of the experiments was to establish the relationship between query performance, data structuredness with respect to schema and to data instances, storage strategy and database size for different queries grouped by their functionality. As explained in section 5.2.1, different storage strategies, database sizes and different interpretations of the structuredness of the data were planned when running these experiments. To test the query performance, the experiments were designed to measure the execution time for each query in milliseconds (as it was the minimum time period that can be measured using MS SQL Server). Each query was executed 20 times, two runs were excluded (the maximum and the minimum execution time, the maximum was always the first cold run). Performance was then computed as the average. Also, standard deviations were calculated for each query and for each storage model, to identify any instability. The detail of the performance metrics was discussed in section 4.5.

5.2.4 Operational Environment

The experiments were conducted in a single machine environment. The machine specifications were:

- Intel ® Pentium ® Core™ Duo processor T2250, 1.73 Ghz
- 1536 MB Ram

- 120 GB Hard Disk Drive.
- Microsoft Windows XP Professional 5.1.2600 service pack 2

The software environment was based on Microsoft SQL Server 2005 (Pal et al. 2005 and 2006, Rys 2005, Lacoude 2006). Transact-SQL (the SQL language used by MS SQL Server) was used as the main language to access the SQL Server and to execute the different queries. The detail of the experimental operational environment was discussed in section 4.6.

5.3 Experiments' Results

The results of the experiments are presented and discussed in this section. In appendix C, the full results including the un-typed scenarios are presented. The results are presented and analysed in three main groups:

1. In the first group 'Using Document Key Queries', the queries targeted semi-structured data using the 'document key' data. In this group, all the storage systems were tested apart from ***PSDA*** and ***PSDT***. The latter two models were excluded because these two systems will give results similar to ***PSD***, which is included in this group of experiments. Specifically, in these three models, the 'document key' is stored as highly-structured data and this group of queries select the data based on the 'document key' and neither the 'author' nor the 'title' data.
2. In the second group 'Using Author Queries', the queries targeted semi-structured data using the 'Author' data. In this group, all the storage systems were tested apart from ***PSDT***. This is because ***PSDT*** will give similar results as ***PSD***. In these two models, the queries will only benefit from the 'document key' as being highly-structured data and not from the 'title' data being highly-structured as in ***PSDT***.
3. In the final group 'Using Title Data Queries', the queries targeted semi-structured data using the 'Title' data. In this group, all the storage systems were tested apart from ***PSDA***. This is because ***PSDA*** will give similar results as ***PSD***, as these queries will not benefit from the 'Author' data as being highly-structured.

The results are sub-grouped within each of these main groups according to the query group functionality, using the grouping defined by XBench XML benchmarking (Yao et al. 2002). Each of these query groups is discussed in a separate subsection. Each subsection introduces the query group, and then presents the interpretation of this query group into meaningful queries to match the experiment data set by presenting the query semantics and syntax (SQL syntax for the relational storage model, XQuery syntax for the ***100%XW*** and the SQL/XQuery syntax for the proposed hybrid model). The results for each are presented graphically such that the relationship between of the performance for each representation and the database size is depicted. Accordingly, in the first graph, the x-axis represents the different storage strategies used in this experiment (***100%R***, ***100%X***, ***37%XW*** ...), and the y-axis represents the three database sizes used in this experiment (***DB1/3***, ***DB2/3*** and ***DB3/3***), while the z-axis represents the execution time in milliseconds. In the second graph, the x-axis represents the 'Data Instances Dimension'. It contains the different hybrid models in the horizontal dimension (***100%R***, ***37%XW***, ***60%XW*** and ***100%XW***). The y-axis represents the execution time. Each line on the graph represents a different database size (***DB1/3***, ***DB2/3*** and ***DB3/3***). It uses the same colour key as the first graph for the different database sizes. Finally, in the third graph, the x-axis represents the 'Schema Dimension'. It contains the different hybrid models in the vertical dimension (***100%R***, ***PSD***, ***PSDA***, ***PSDT*** and ***100%XW***). The y-axis, the data series and the database size colour are similar to graph two. These graphs are followed by a table showing the average execution time and the standard deviations for the different runs for this query. The graphical representation of each set of results is followed by an analysis of the results based on the storage strategy, the data structuredness and finally the different database sizes.

5.3.1 Using 'Document Key' Queries

The queries in this group access documents by specifying a unique document key value. Eight queries are included in this section. They are: shallow exact match (Q1), path expressions (Q9), document construction (Q12 for preserving the structure and Q13 for transforming the structure), irregular data (Q14 for missing elements and Q15 for empty (null) values), retrieval of individual documents (Q16) and references and joins (Q20). The following subsections discuss each of these queries in turn.

5.3.1.1 Exact Match (Shallow)

The queries in this group required exact string match with specified and possibly long path expressions, depending on the levels of predicates being queried in the XML documents. Consequently, they can be shallow queries that match only at the top level of the XML document trees (for example query Q1), or deep queries that match the nested structure of the XML document tree (query Q2, this query is discussed in section 5.3.2.1).

The queries in this group were re-formulated to query the data set used in the experiments, while retaining the functionally they were defined to characterise within the benchmark. Thus, the query within this group "Shallow Exact Match" was re-formulated to return in-proceeding's titles in the case of the ***60%XW*** or article's titles in the case of the ***37%XW***, to query the semi-structured part of the data. For the ***100%R***, ***100%X*** and ***PSD***, half of the runs returned in-proceedings' titles and the other half returned articles' titles, this is because in these three cases the data was: shredded in relational tables in the first case, stored as one XML data field in the second case and stored inside document fragments in the third case.

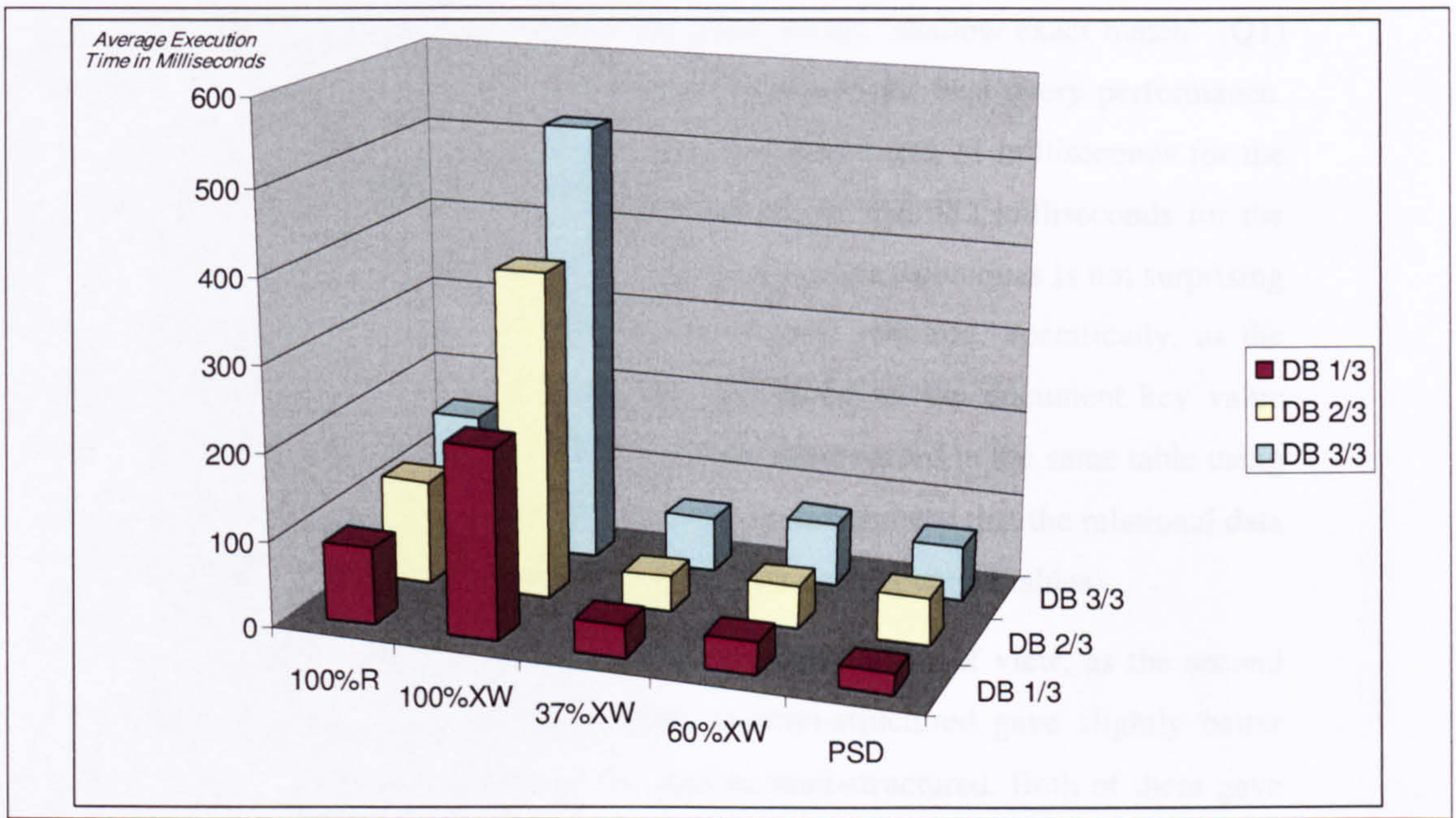
Also, it is necessary to express the re-formulated query in three different forms, so as to query data stored using the three different storage strategies. This is the case for all the queries and this is shortly explained in detail for the first query. As the following table presents, for the ***100%R***, the query is expressed in purely SQL syntax. For ***100%XW***, the query is expressed in XQuery format (using SQL syntax for MS SQL Server). Finally for the proposed model, the query syntax is expressed in a combined SQL and XQuery format. Further explanations of these different storage schemas are presented in section 4.4.1. The equivalence of these queries were established by checking the first couple of runs for each query and making sure that all these queries returned the same exact results.

The following paragraphs show a brief query description followed by a table showing the formulation of each query into SQL, XQuery and SQL/XQuery syntax.

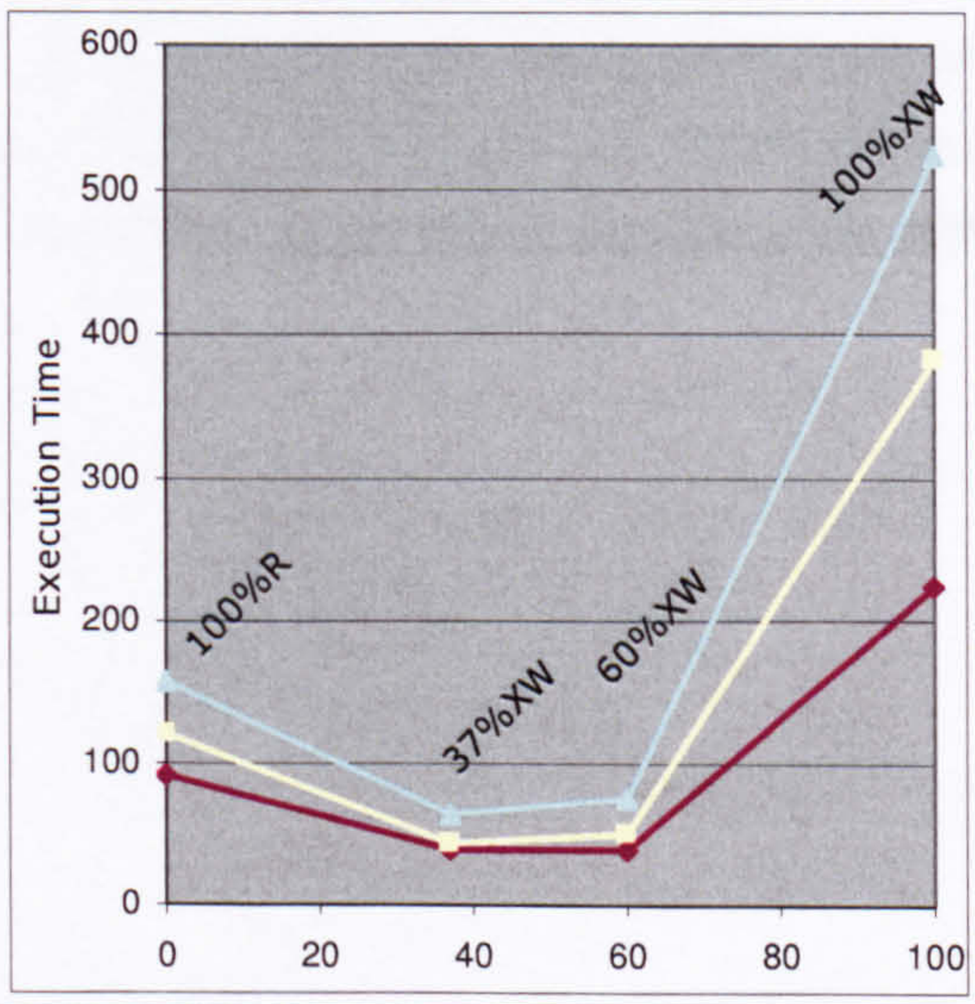
- ***Q1***: Return in-proceeding's titles (or article's titles) that has key value X.

SQL Syntax	`SELECT A_Title.Title AS Q01A100R FROM A_Title Inner Join A_Doc ON A_Title.DocId = A_Doc.DocId WHERE dockey = @RandomArticleDocKey For XML Auto;`
XQuery Syntax	`SELECT xmldoc.query('/dblp/article[@key="' + @RandomArticleDocKey + '"]/title') AS Q01A100XW FROM B_XMLDocument;`
SQL/XQuery Syntax	`SELECT xmlextract.query ('/inproceedings/title') AS Q01A60XW FROM C_Doc WHERE dockey = @RandomInProceedingsDocKey;`

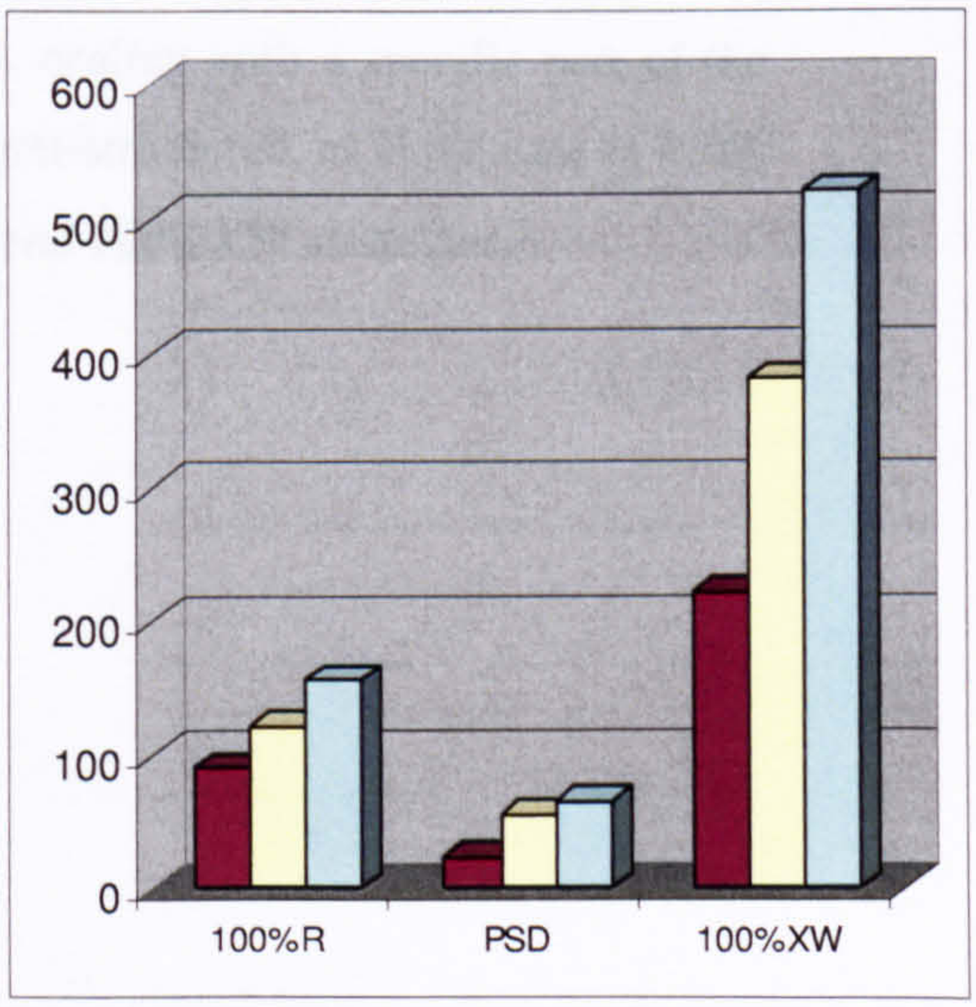
The following graphs show the results for this query:



Q1 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q1 – Graph 2: Data Instances Dimension



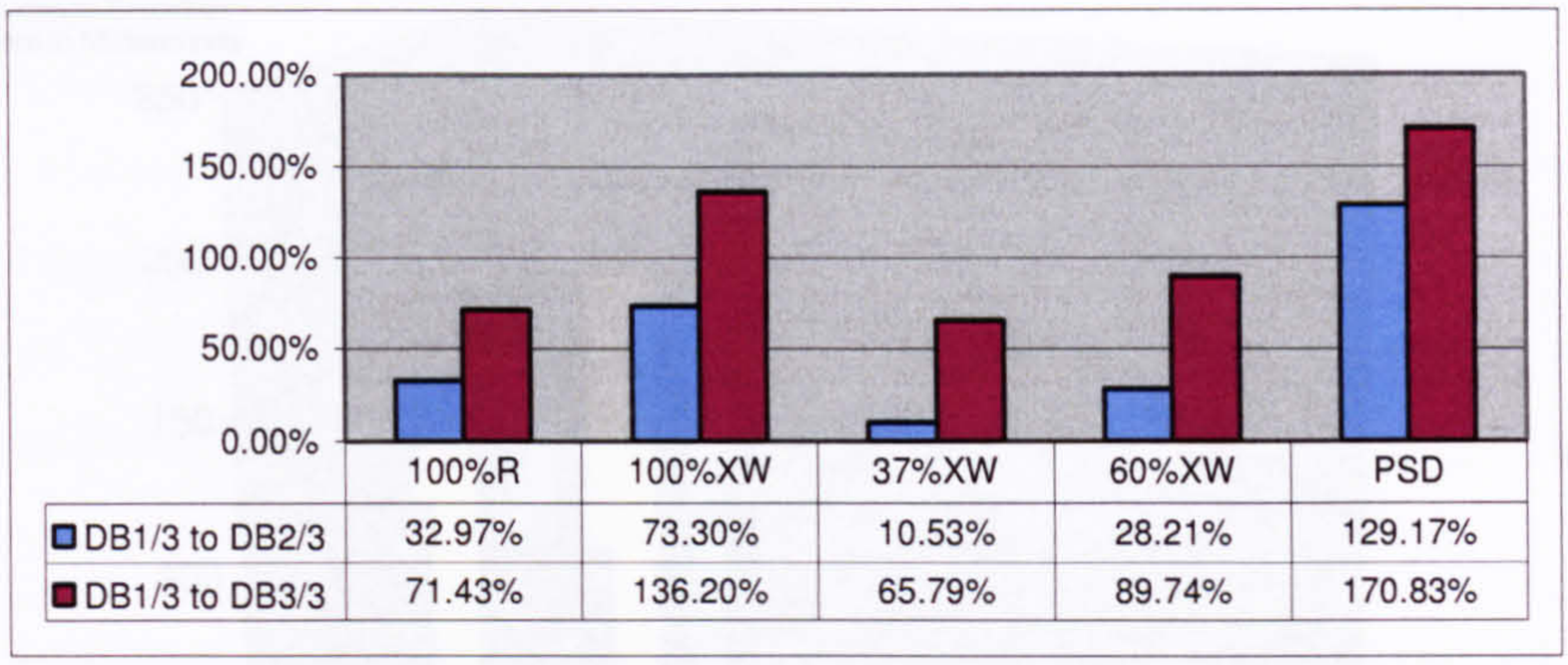
Q1 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	91	18.95	121	24.90	156	73.50
100%XW	221	26.54	383	64.06	522	100.41
37%XW	38	10.18	42	6.28	63	20.42
60%XW	39	9.42	50	8.83	74	13.65
PSD	24	8.19	55	8.27	65	30.92

Figure 5.2 Query 1: Shallow Exact Match

From the storage strategy point of view, the query group "shallow exact match" (Q1) and as the first graph shows, the hybrid models produced the best query performance. For example in the ***DB3/3***, it took on average between 63 and 74 milliseconds for the hybrid model, 156 milliseconds for the relational model and 522 milliseconds for the ***100%XW***. The relative performance of the different storage techniques is not surprising and can be explained in terms of the number of joins required. Specifically, as the SQL/XQuery syntax showed, it selected the data based on the document key value (which is a unique and indexed data field) from the same record in the same table using the 'document key' to access that record. The SQL syntax showed that the relational data model needed to do one join to get the data (from title and document tables).

From the data structuredness - data instances dimension point of view, as the second graph shows, dealing with 37% of the data as semi-structured gave slightly better performance than dealing with 60% of the data as semi-structured. Both of them gave better performances than ***100%R*** and ***100%XW***. From the data structuredness - schema dimension point of view, as the third graph shows, dealing with a specific part of the schema as highly-structured and the remaining as semi-structured, as is the case in ***PSD***, gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD
DB1/3 to DB2/3	32.97%	73.30%	10.53%	28.21%	129.17%
DB1/3 to DB3/3	71.43%	136.20%	65.79%	89.74%	170.83%

Figure 5.3 Query 1: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q1. The worst performance was for the hybrid model ***PSD.***

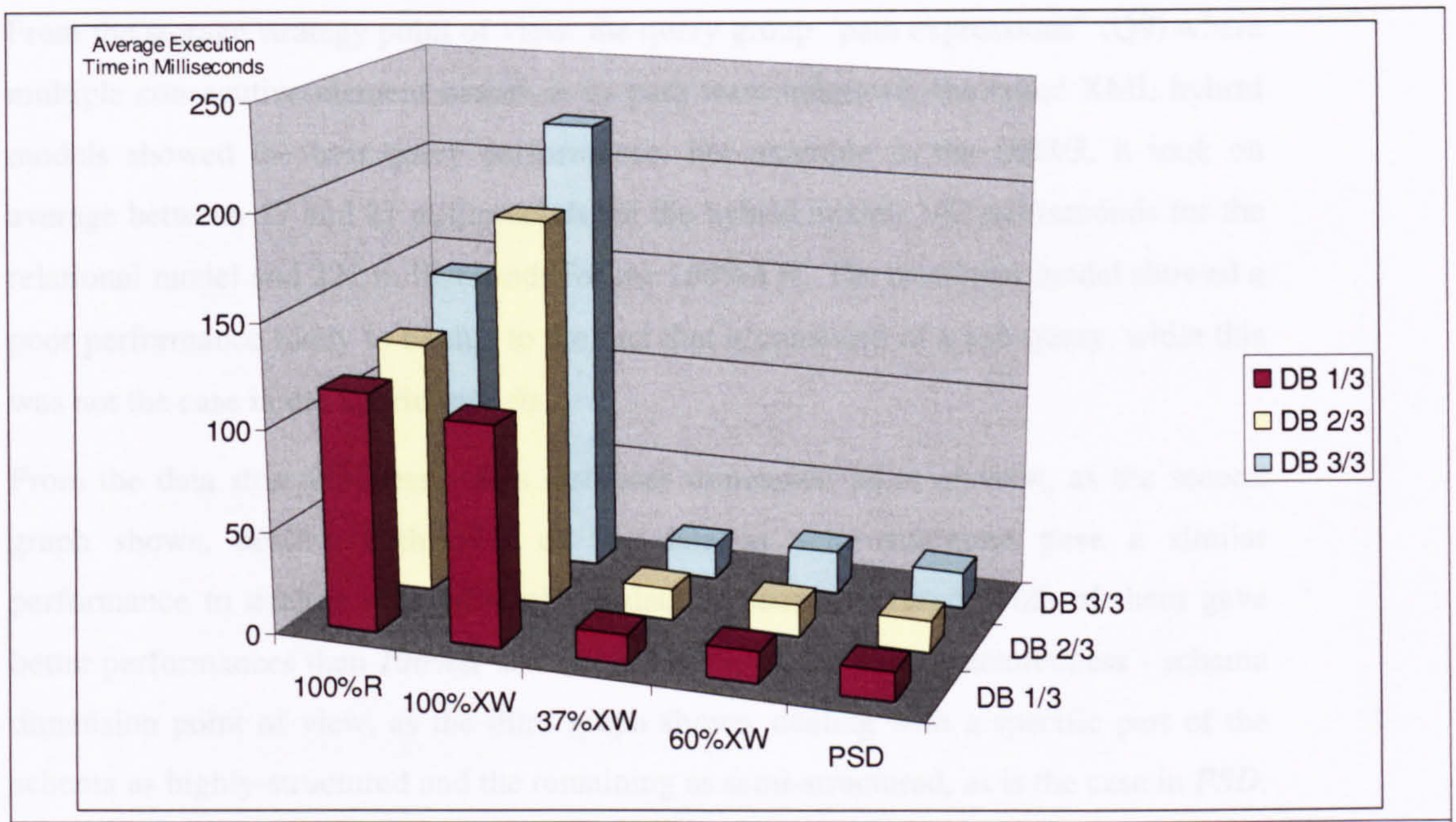
5.3.1.2 Path expressions

The queries in this group involved path expressions: Q8 queries data where one element name in its path was unknown (this query is discussed in section 5.3.3.1), Q9 queries data where multiple consecutive element names in its path were unknown.

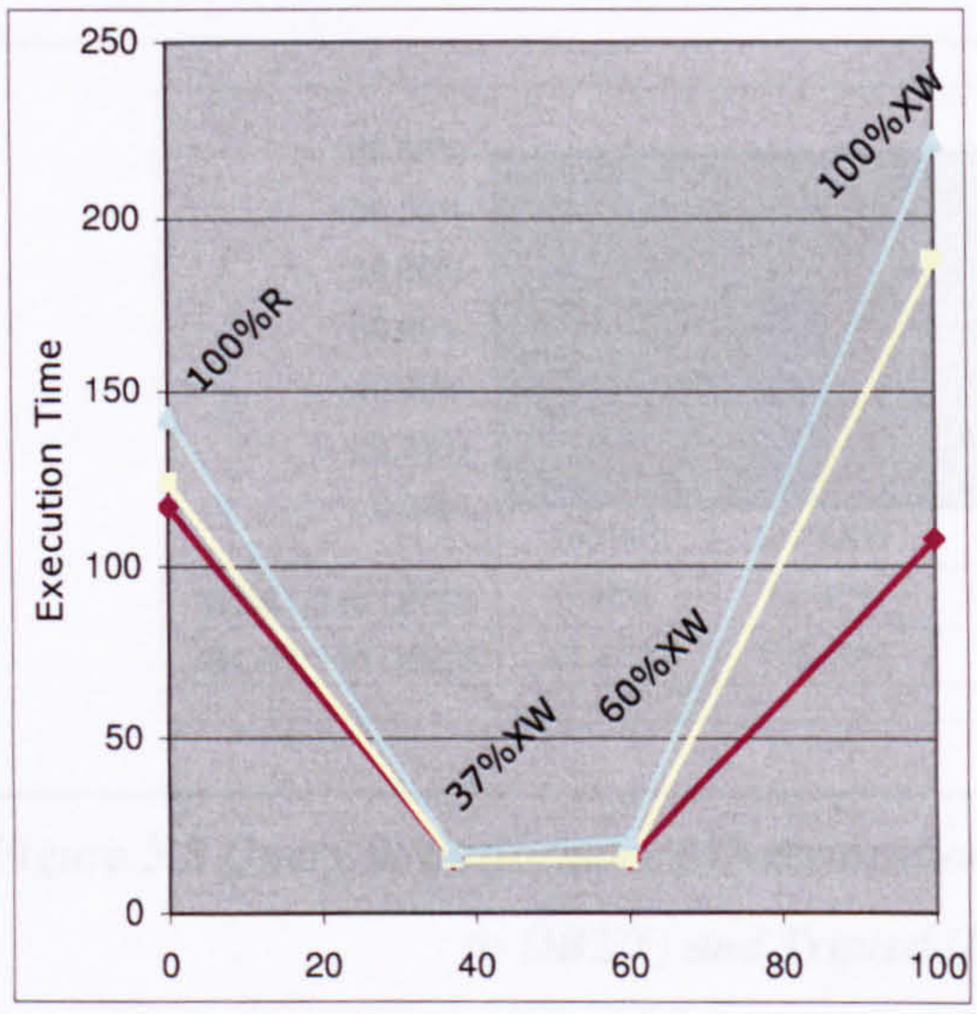
The query in this group was interpreted to match the data set used in the experiments as follows:

- ***Q9***: Return in-proceeding's authors (or article's authors) for specific publications that had key value X. The (//) was used inside XQuery syntax to direct the query to look for any path inside the document.

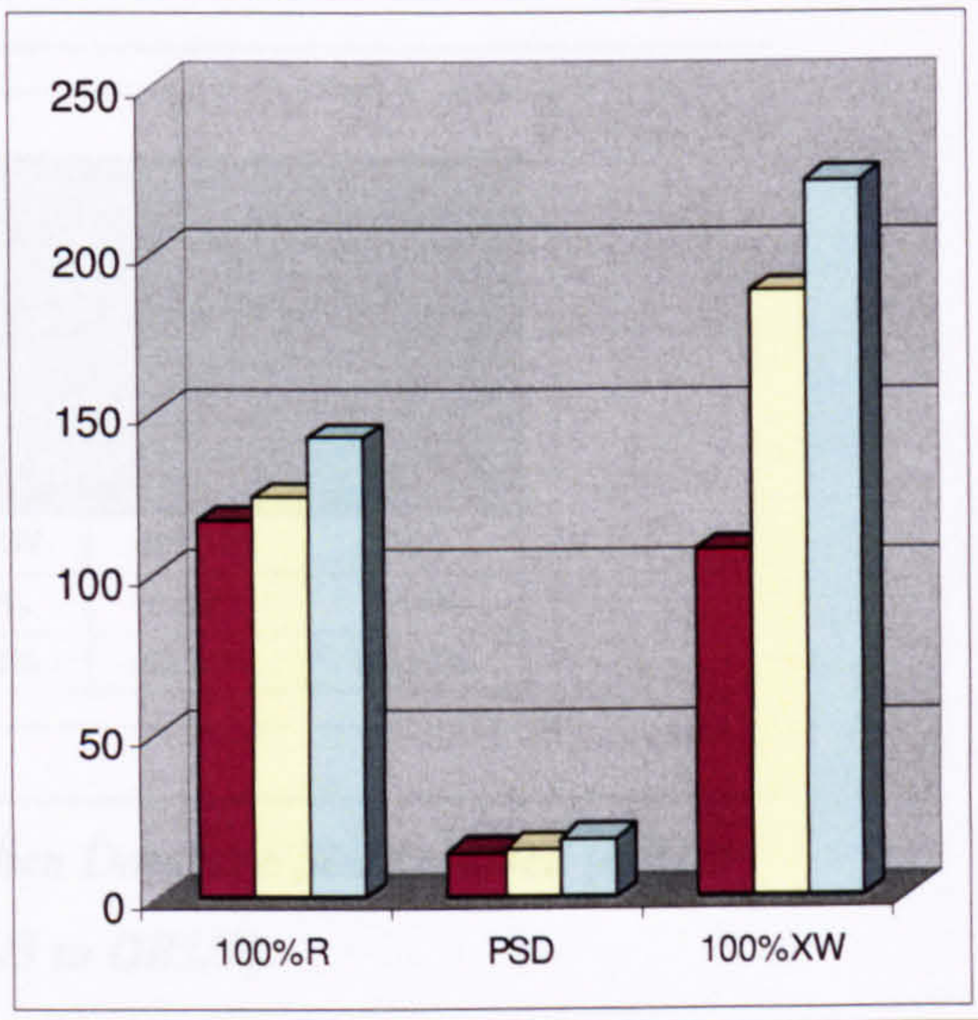
SQL Syntax	SELECT A_Author.Author AS Q09A100R FROM dbo.A_Author WHERE A_Author.docId in (SELECT DocId from A_Doc WHERE DocKey = @RandomArticleDocKey) For XML Auto;
XQuery Syntax	SELECT xmldoc.query('for $x in /dblp/article WHERE $x[@key="' + @RandomArticleDocKey + '"] return $x//author') AS Q09A100X From B_XMLDocument;
SQL/XQuery Syntax	SELECT xmlextract.query('for $x in /inproceedings return $x//author') AS Q09A60X From C_Doc WHERE dockey = @RandomInProceedingsDocKey



Q9 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q9 – Graph 2: Data Instances Dimension



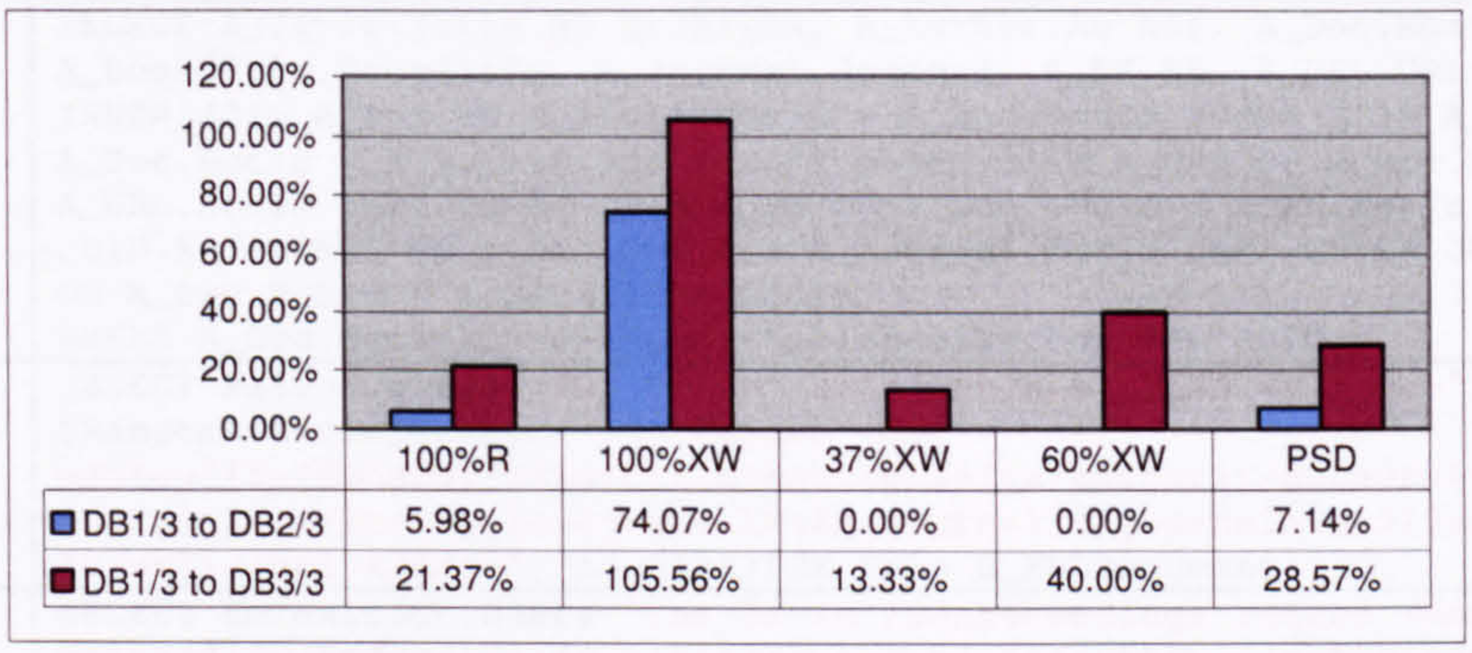
Q9 – Graph 3: Schema Dimension

	DB 1/3		DB 2/3		DB 3/3	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	117	8.46	124	15.04	142	13.11
100%XW	108	16.21	188	43.60	222	49.97
37%XW	15	1.46	15	2.28	17	0.98
60%XW	15	2.52	15	1.14	21	8.31
PSD	14	5.54	15	0.62	18	1.08

Figure 5.4 Query 9:Path expressions

From the storage strategy point of view, the query group "path expressions" (Q9) where multiple consecutive element names in its path were unknown, the typed XML hybrid models showed the best query performance. For example in the ***DB3/3***, it took on average between 17 and 21 milliseconds for the hybrid model, 142 milliseconds for the relational model and 222 milliseconds for the ***100%XW***. The relational model showed a poor performance likely to be due to the fact that it consisted of a sub-query, while this was not the case in the hybrid models.

From the data structuredness - data instances dimension point of view, as the second graph shows, dealing with 37% of the data as semi-structured gave a similar performance to dealing with 60% of the data as semi-structured. Both of them gave better performances than ***100%R*** and ***100%XW***. From the data structuredness - schema dimension point of view, as the third graph shows, dealing with a specific part of the schema as highly-structured and the remaining as semi-structured, as is the case in ***PSD***, gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD
DB1/3 to DB2/3	5.98%	74.07%	0.00%	0.00%	7.14%
DB1/3 to DB3/3	21.37%	105.56%	13.33%	40.00%	28.57%

Figure 5.5 Query 9: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure illustrates the performance deterioration when the database size doubled and tripled for Q9. The worst performance was for the model ***100%XW.***

5.3.1.3 Document construction

Structure is important in many XML documents. However, some systems experience difficulties in even preserving the document's original structure. This class of queries

tested the performance of the system in preserving the structure (Q12) and in transforming the structure (Q13). The queries in this group were interpreted to match the data set used in the experiments as follows:

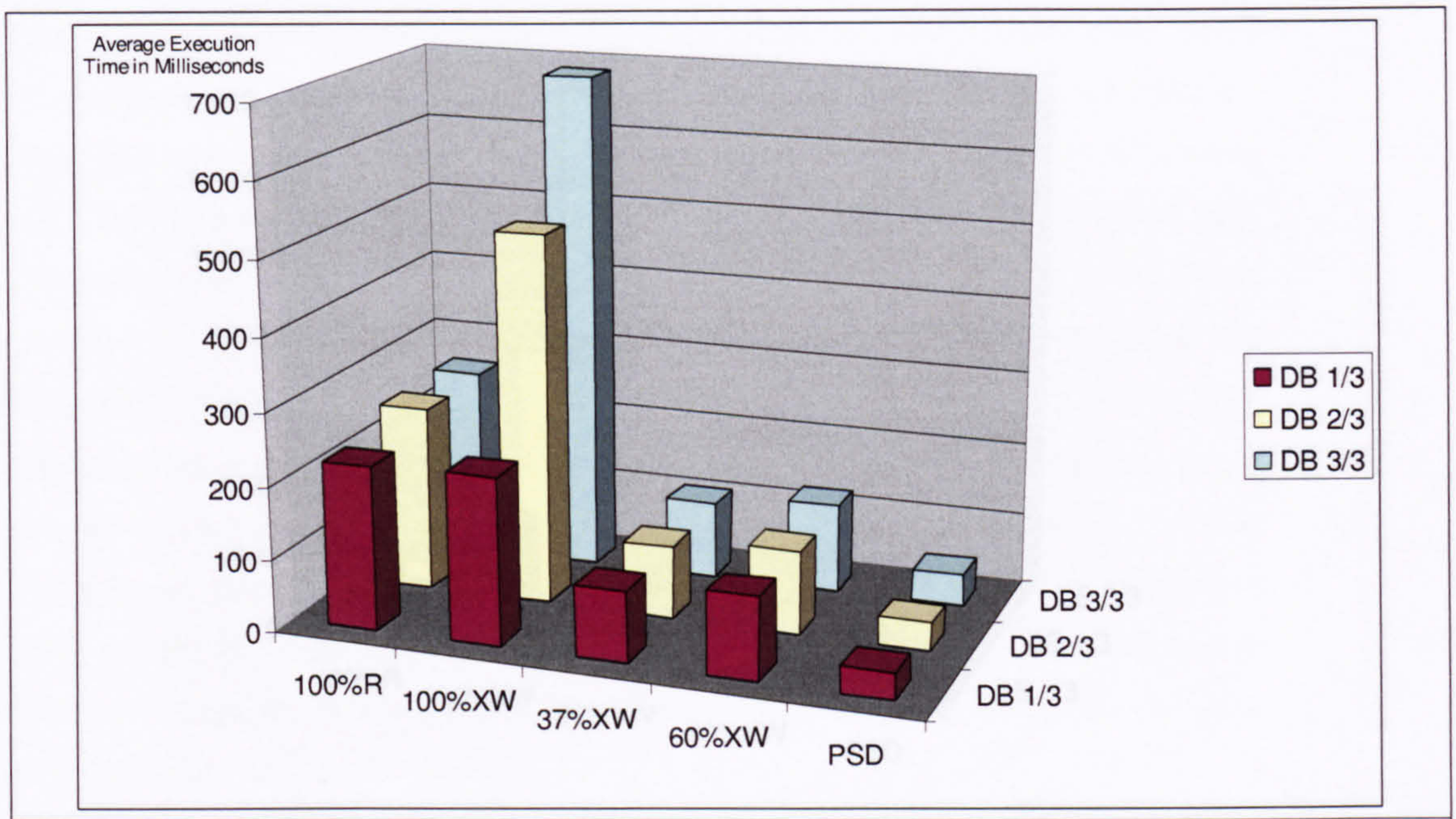
- ***Q12***: List all in-proceeding's titles (or article's titles), publication date, authors for specific document preserving the original document structure.

SQL Syntax	`SELECT dbo.A_Title.Title AS Q12A100R, dbo.A_Author.Author, dbo.A_Doc.MDate FROM dbo.A_Title INNER JOIN dbo.A_Doc ON dbo.A_Title.DocId = dbo.A_Doc.DocId INNER JOIN dbo.A_Author ON dbo.A_Doc.DocId = dbo.A_Author.DocId WHERE A_Doc.dockey = @RandomArticleDocKey For XML Auto;`
XQuery Syntax	`SELECT xmldoc.query('for $x in /dblp/article WHERE $x[@key="' + @RandomArticleDocKey + '"] return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q12A100X From B_XMLDocument;`
SQL/XQuery Syntax	`SELECT xmlextract.query('for $x in /inproceedings return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q12A60X From C_Doc WHERE dockey = @RandomInProceedingsDocKey`

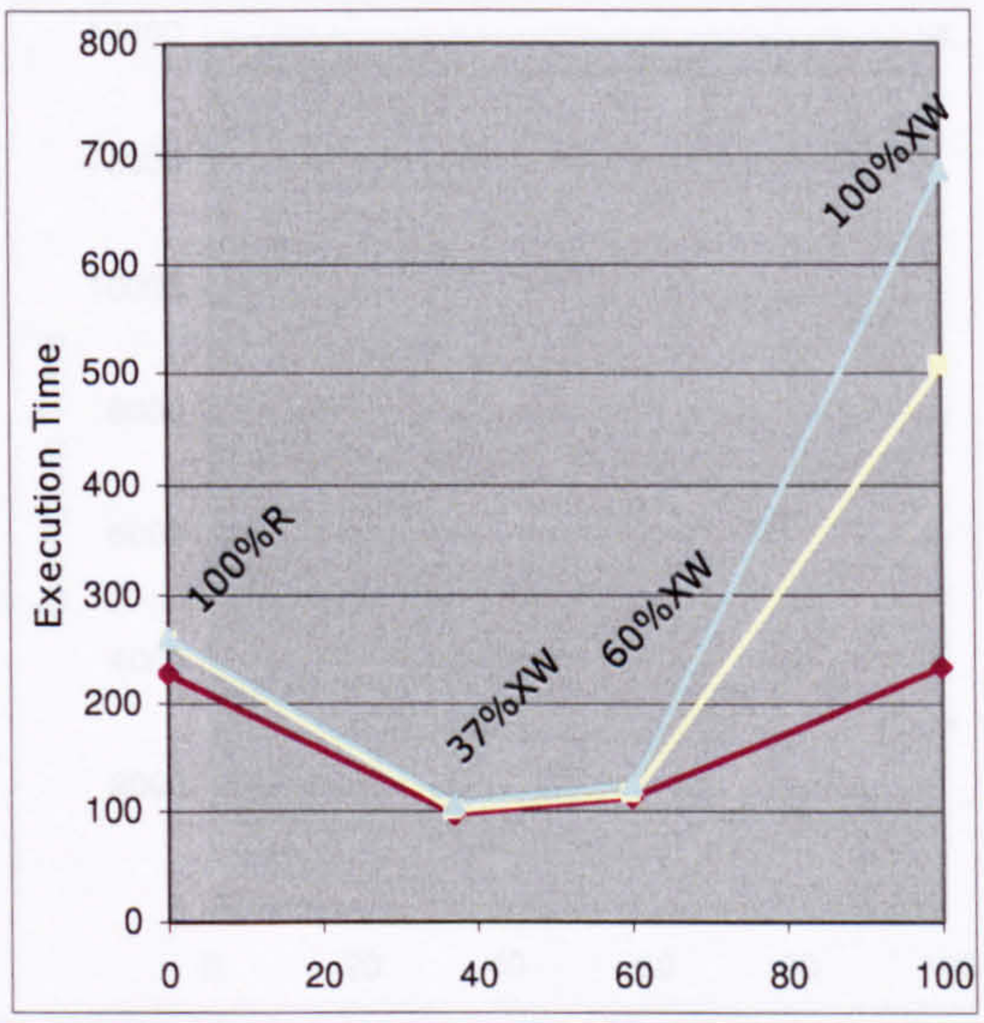
- ***Q13***: List all in-proceeding's titles (or article's titles), publication date, authors for specific document transforming the original document structure to another structure.

SQL Syntax	`SELECT A_Title.Title AS Q13A100R, A_Author.Author, A_Doc.MDate, A_BookTitle.BookTitle, A_Journal.Journal, A_EE.EE, A_URL.URL FROM A_Title INNER JOIN A_Doc ON A_Title.DocId = A_Doc.DocId INNER JOIN A_Author ON A_Doc.DocId = A_Author.DocId LEFT OUTER JOIN A_URL ON A_Doc.DocId = A_URL.DocId LEFT OUTER JOIN A_EE ON A_Doc.DocId = A_EE.DocId LEFT OUTER JOIN A_Journal ON A_Doc.DocId = A_Journal.DocId LEFT OUTER JOIN A_BookTitle ON A_Doc.DocId = A_BookTitle.DocId WHERE A_Doc.dockey = @RandomArticleDocKey For XML AUTO;`
XQuery Syntax	`SELECT xmldoc.query('for $x in /dblp/article WHERE $x[@key="' + @RandomArticleDocKey + '"] return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors><booktitle>{$x/booktitle}</booktitle><journal>{$x/journal}</journal><ee>{$x/ee}</ee><url>{$x/url}</url></doc>') AS Q13A100X From B_XMLDocument;`
SQL/XQuery Syntax	`SELECT xmlextract.query('for $x in /inproceedings return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors><booktitle>{$x/booktitle}</booktitle><journal>{$x/journal}</journal><ee>{$x/ee}</ee><url>{$x/url}</url></doc>') AS Q13A60X From C_Doc WHERE dockey = @RandomInProceedingsDocKey`

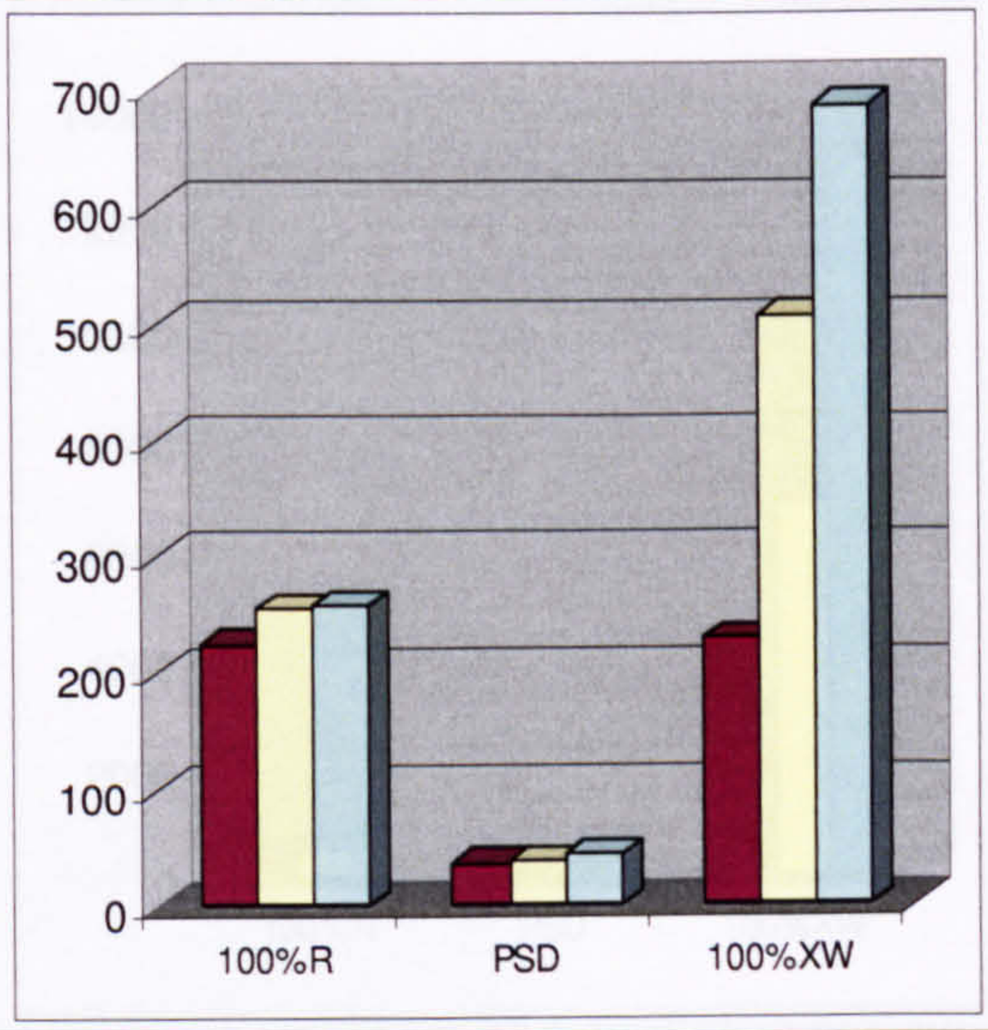
The following graphs show the results for these queries



Q12 – Graph 1: Database Size vs. Storage Strategy vs. Performance



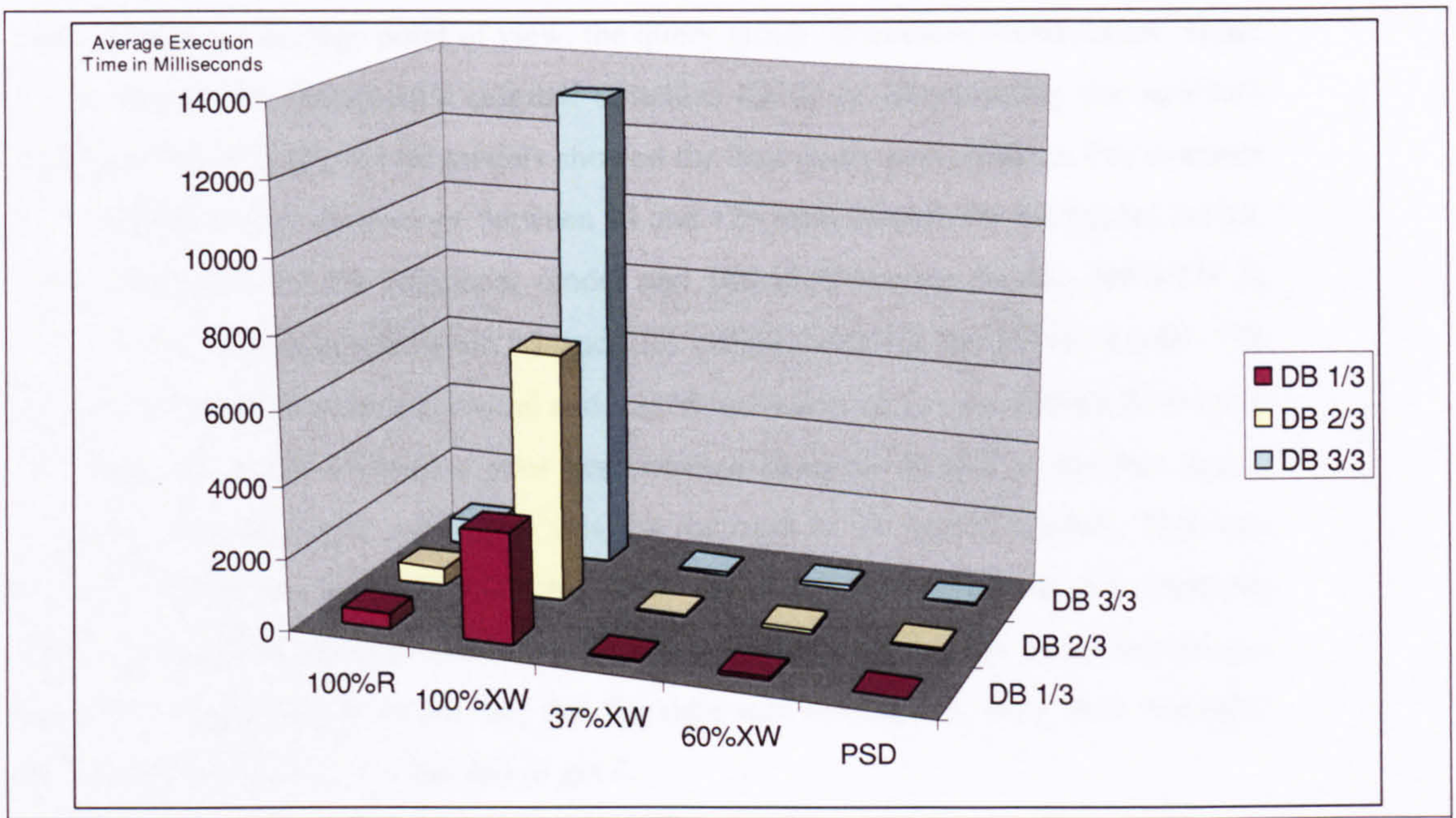
Q12 – Graph 2: Data Instances Dimension



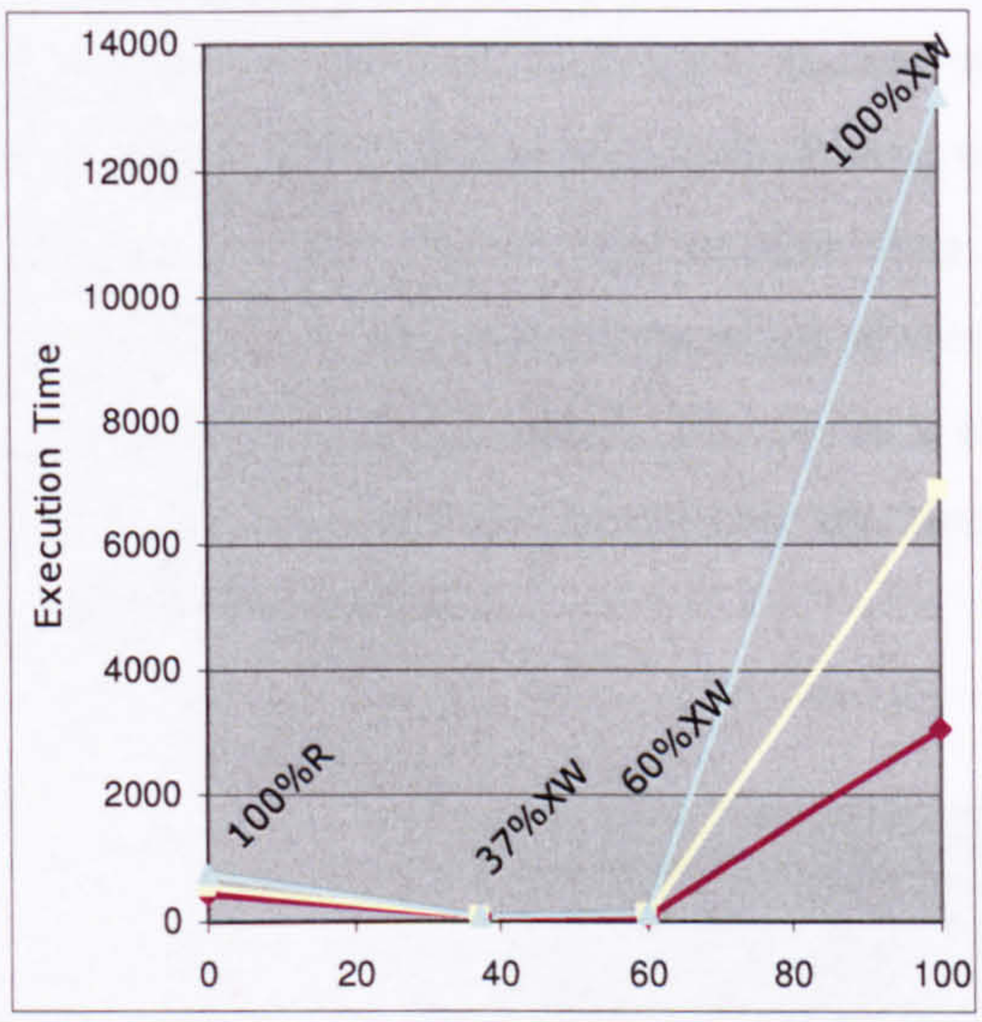
Q12 – Graph 2: Data Instances Dimension

Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	227	9.81	256	90.74	259	47.26
100%XW	231	57.75	507	51.37	686	52.76
37%XW	98	35.84	100	34.52	107	19.20
60%XW	112	14.11	115	9.37	124	21.87
PSD	36	6.68	37	8.17	44	13.61

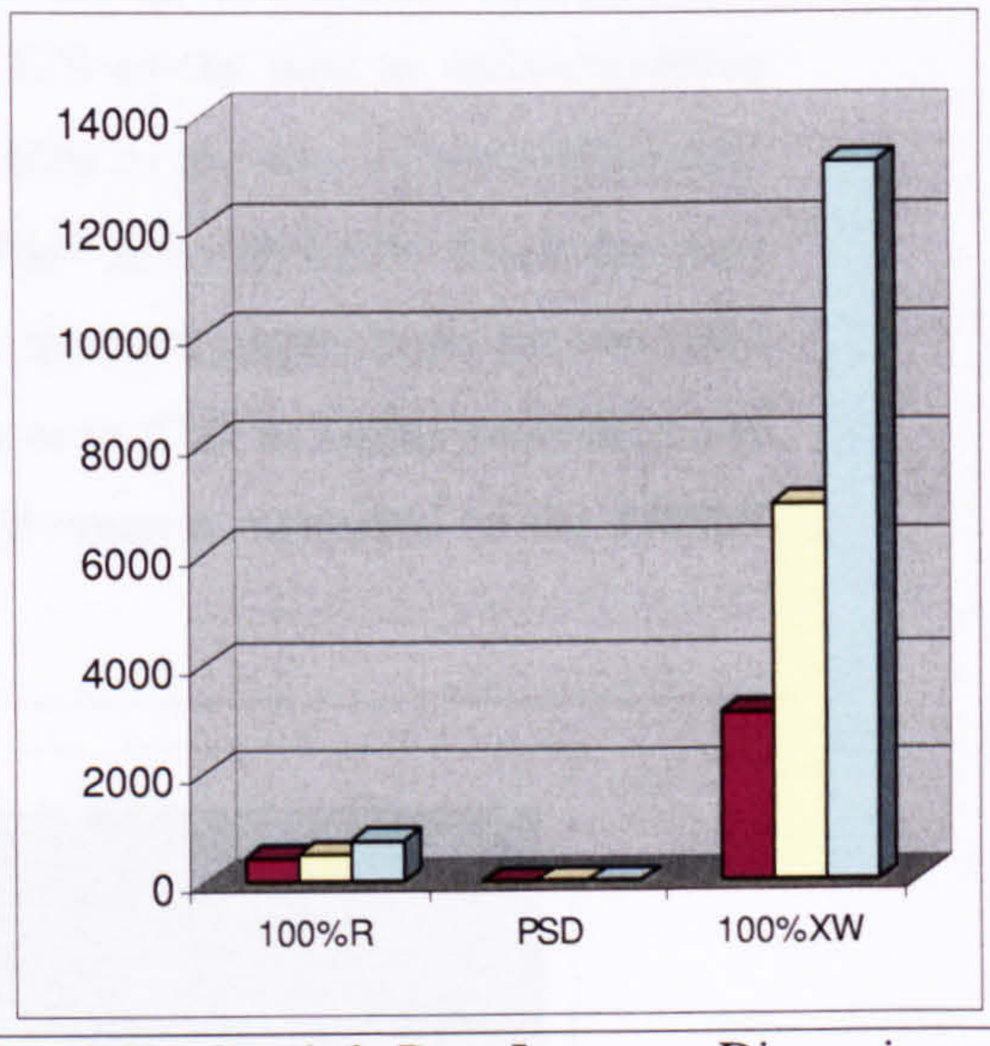
Figure 5.6 Query 12: Document Construction - Structure Preserving



Q13 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q13 – Graph 2: Data Instances Dimension



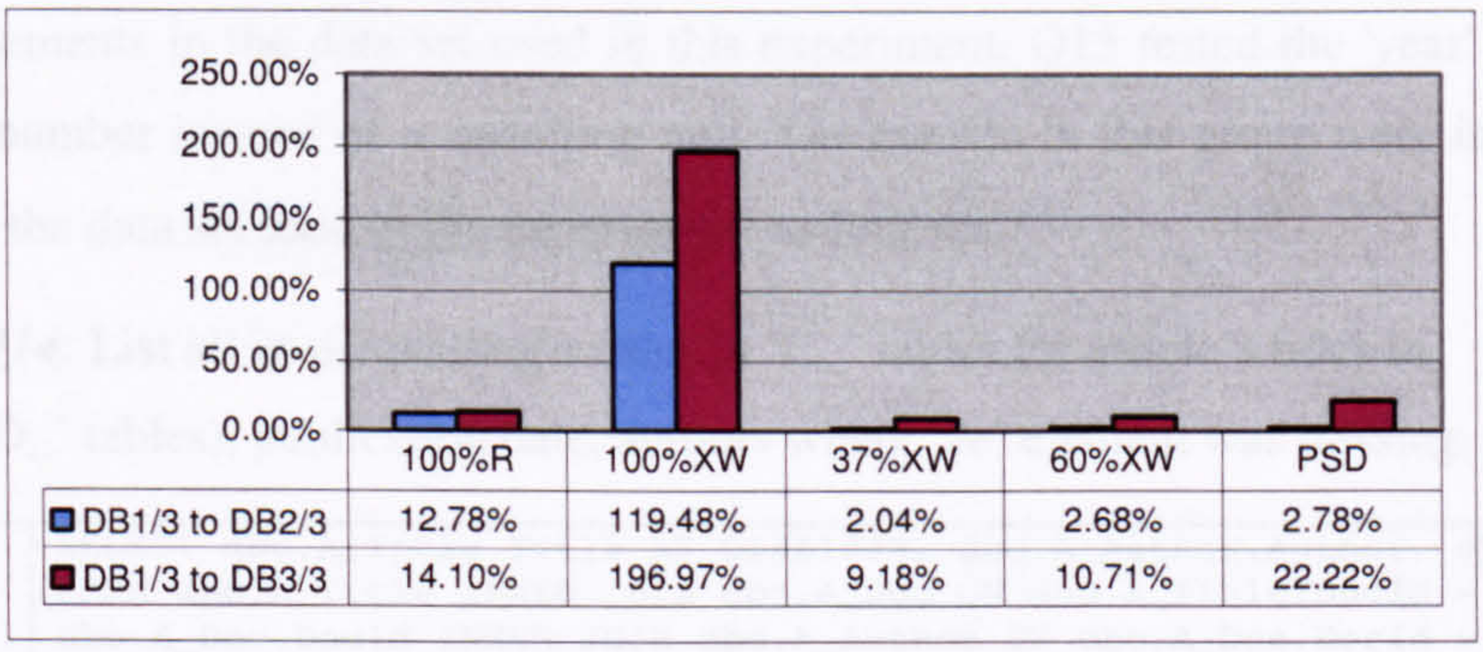
Q13 – Graph 2: Data Instances Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	420	36.06	515	24.18	739	78.69
100%XW	3082	992.17	6882	1419.22	13138	2761.02
37%XW	55	18.45	57	7.12	79	32.56
60%XW	72	23.67	80	5.85	102	36.87
PSD	12	1.57	13	1.17	15	1.21

Figure 5.7 Query 13: Document Construction - Structure Transforming

From the storage strategy point of view, the query group "document construction" either for preserving the document's original structure (Q12) or transforming the structure (Q13), the typed XML hybrid models showed the best query performance. For example in the ***DB3/3***, it took on average between 44 and 124 milliseconds for the hybrid model, 259 milliseconds for the relational model and 686 milliseconds for the ***100%XW*** in Q12. It took on average between 15 and 102 milliseconds for the hybrid model, 739 milliseconds for the relational model and 13138 milliseconds for the ***100%XW*** in Q13. The relational model showed a poor performance likely to be due to the fact that it consisted of a sub-query, while this was not the case in the hybrid models. This was probably due to the data being selected from one table and returned as it is from the XML extract using the document key. The poor performance of the relational storage model was expected due to the fact that the data was shredded in more than one table and more than one join was needed to get it.

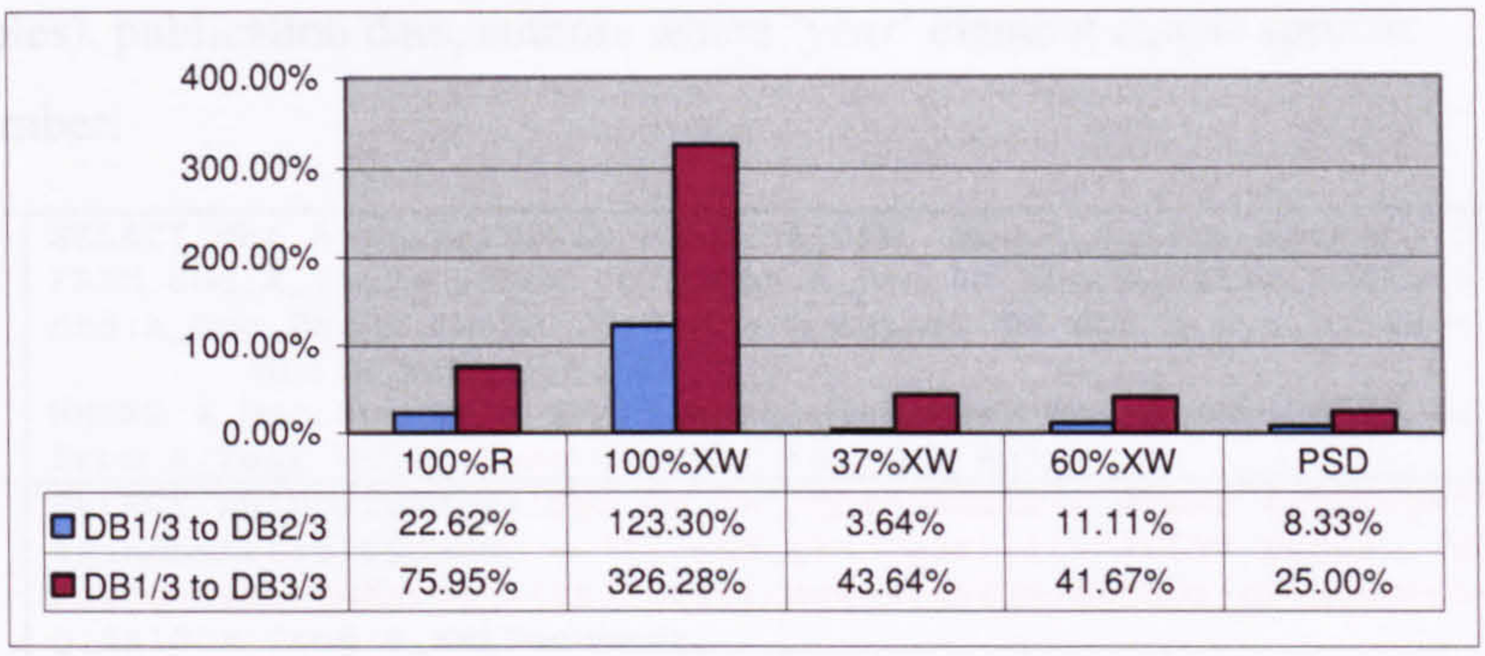
From the data structuredness - data instances dimension point of view, as the second graph shows for both Q12 and Q13, dealing with 37% of the data as semi-structured gave slightly better performance than dealing with 60% of the data as semi-structured. Both of them gave better performances than ***100%R*** and ***100%XW***. From the data structuredness - schema dimension point of view, as the third graph shows for both Q12 and Q13, dealing with a specific part of the schema as in ***PSD*** as highly-structured and the remaining as semi-structured gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD
DB1/3 to DB2/3	12.78%	119.48%	2.04%	2.68%	2.78%
DB1/3 to DB3/3	14.10%	196.97%	9.18%	10.71%	22.22%

Figure 5.8 Query 12: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q12. The worst performance was for the model ***100%XW.***



	100%R	100%XW	37%XW	60%XW	PSD
DB1/3 to DB2/3	22.62%	123.30%	3.64%	11.11%	8.33%
DB1/3 to DB3/3	75.95%	326.28%	43.64%	41.67%	25.00%

Figure 5.9 Query 13: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q13. The worst performance was for the model ***100%XW.***

5.3.1.4 Irregular data

Irregularity of schema is a common feature in XML databases. This class of queries tested missing elements (Q14) and empty (null) values (Q15). Since there were no empty elements in the data set used in this experiment, Q15 tested the 'year' data as a specific number instead of it equalling null. The queries in this group were interpreted to match the data set used in the experiments as follows:

- ***Q14***: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors where '*ee*' element was missing.

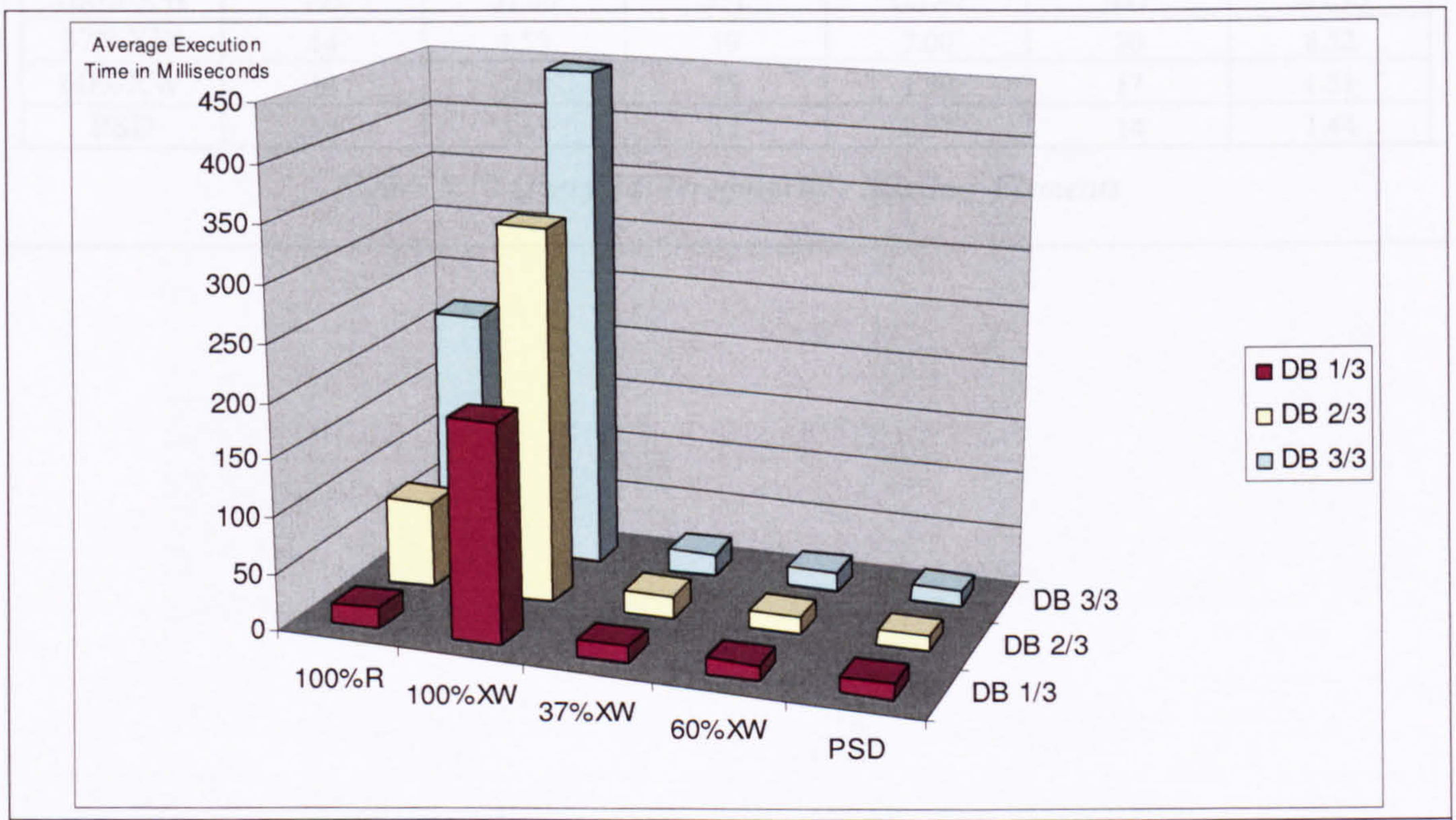
SQL Syntax	SELECT dbo.A_Title.Title AS Q14A100R, dbo.A_Author.Author, dbo.A_Doc.MDate FROM dbo.A_Title INNER JOIN dbo.A_Doc ON dbo.A_Title.DocId = dbo.A_Doc.DocId INNER JOIN dbo.A_Author ON dbo.A_Doc.DocId = dbo.A_Author.DocId WHERE A_Doc.dockey = @RandomArticleDocKey And A_Doc.DocId not in (SELECT DocId from A_EE) For XML Auto;
XQuery Syntax	SELECT xmldoc.query('for $x in /dblp/article WHERE $x[@key="' + @RandomArticleDocKey + '"] and empty($x/ee) return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q14A100X From B_XMLDocument;

SQL/XQuery Syntax	SELECT xmlextract.query('for $x in /inproceedings return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q14A60X From C_Doc WHERE dockey = @RandomInProceedingsDocKey and XMLExtract.value ('empty(/inproceedings/ee)', 'bit') = 1

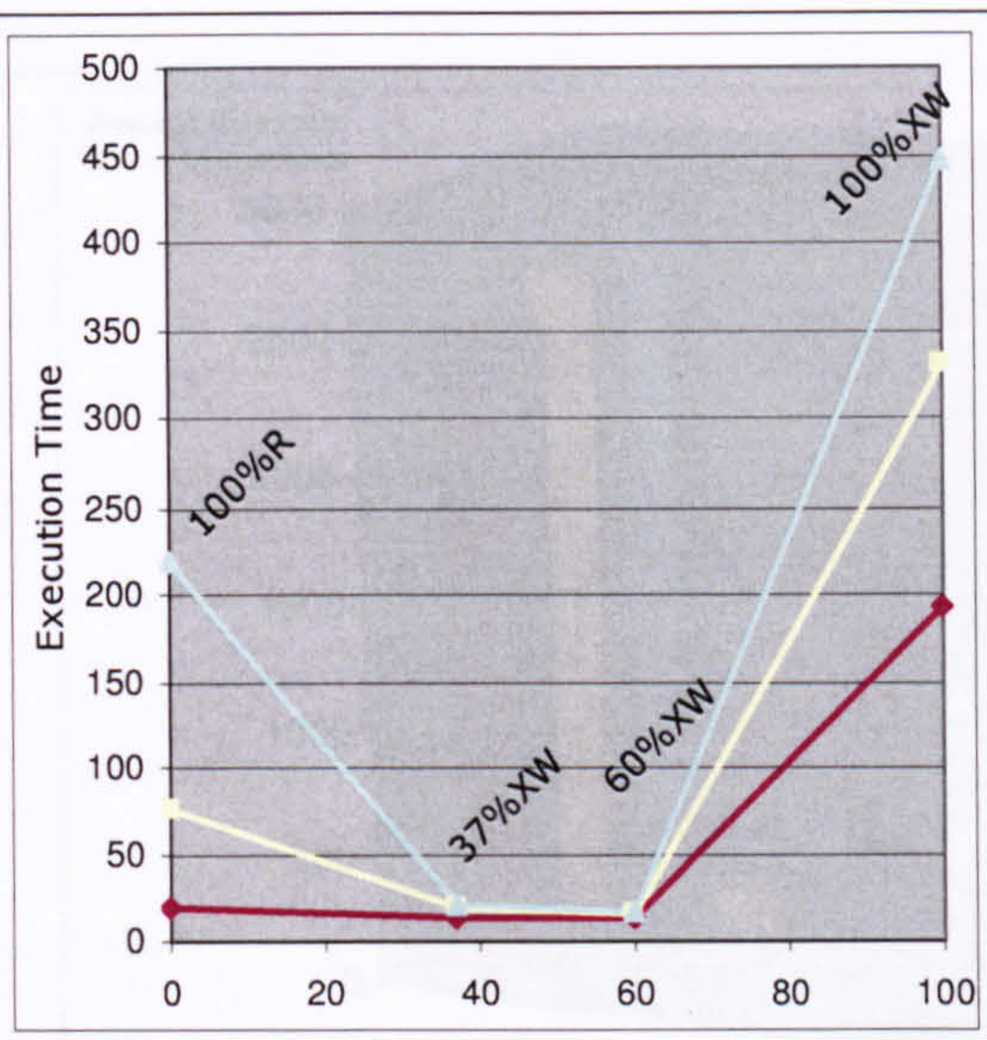
- ***Q15***: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors where '*year*' element equals specific number.

SQL Syntax	SELECT dbo.A_Title.Title AS Q15A100R, dbo.A_Author.Author, dbo.A_Doc.MDate FROM dbo.A_Title INNER JOIN dbo.A_Doc ON dbo.A_Title.DocId = dbo.A_Doc.DocId INNER JOIN dbo.A_Author ON dbo.A_Doc.DocId = dbo.A_Author.DocId WHERE A_Doc.dockey = @RandomArticleDocKey And A_Doc.DocId in (SELECT DocId from A_Year WHERE Year = 2003) For XML Auto;
XQuery Syntax	SELECT xmldoc.query('for $x in /dblp/article WHERE $x[@key="' + @RandomArticleDocKey + '"] and ($x/year)[1]="2003" return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q15A100X From B_XMLDocument;
SQL/XQuery Syntax	SELECT xmlextract.query('for $x in /inproceedings return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q15A60X From C_Doc WHERE dockey = @RandomInProceedingsDocKey and XMLExtract.value ('(/inproceedings/year)[1]', 'varchar(5)') = '2003'

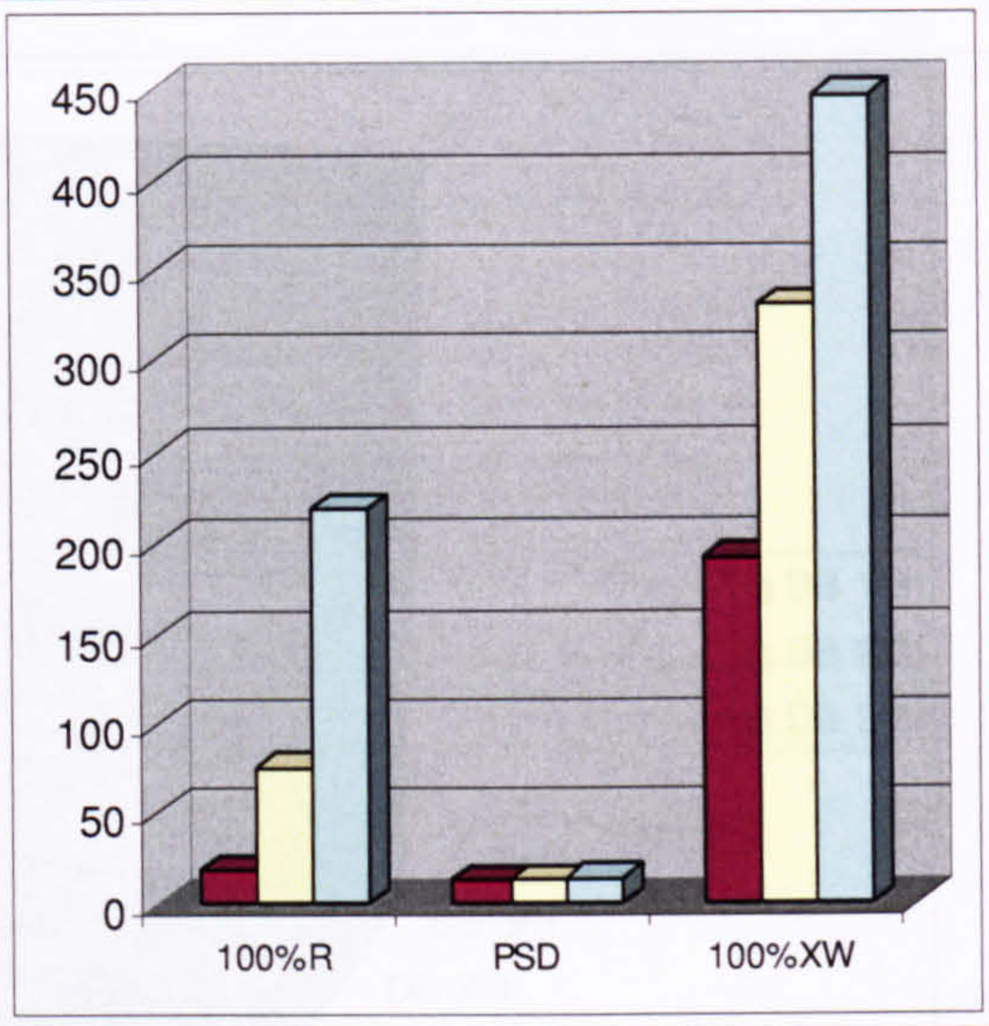
The following graphs show the results for these queries



Q14 – Graph 1: Database Size vs. Storage Strategy vs. Performance



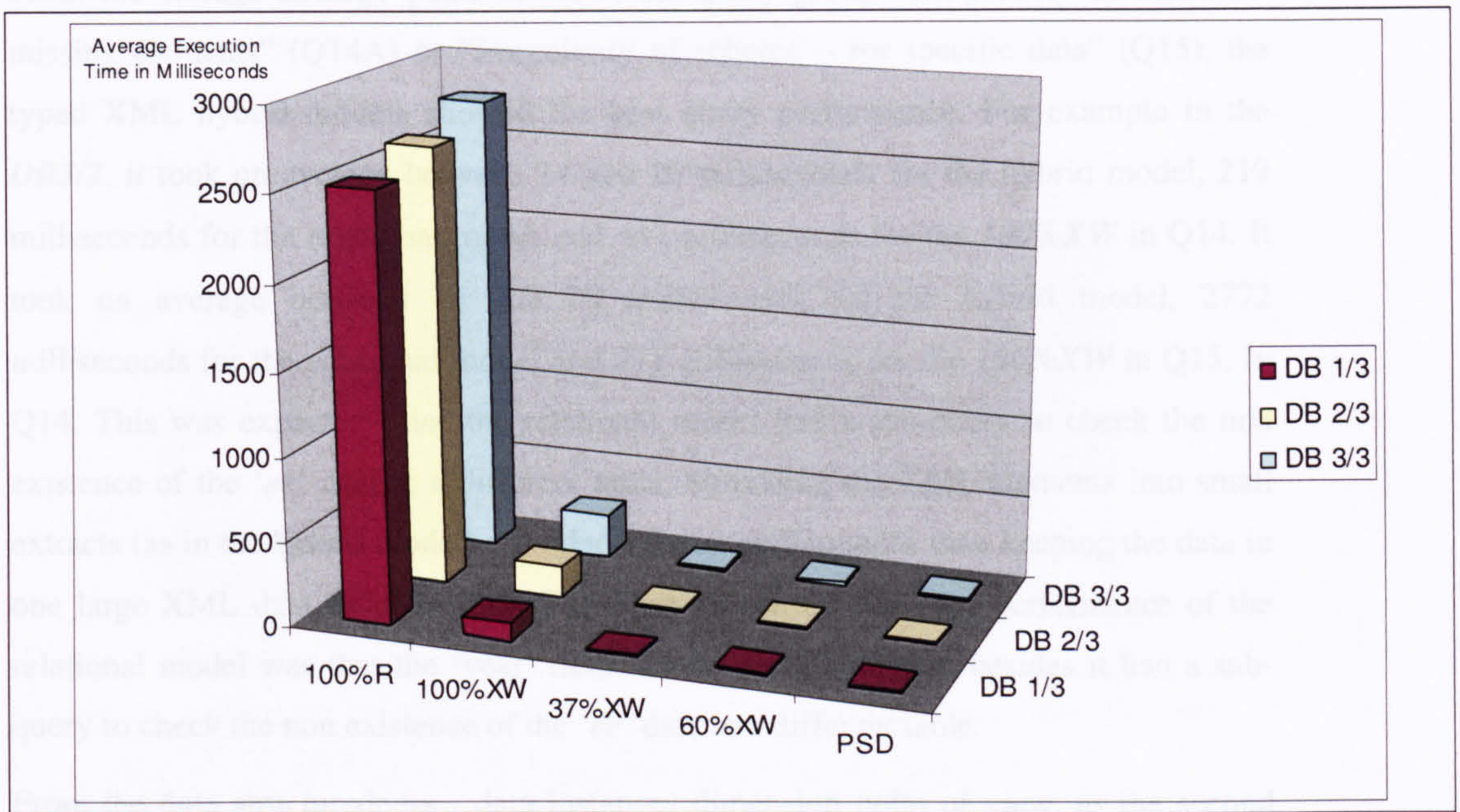
Q14 – Graph 2: Data Instances Dimension



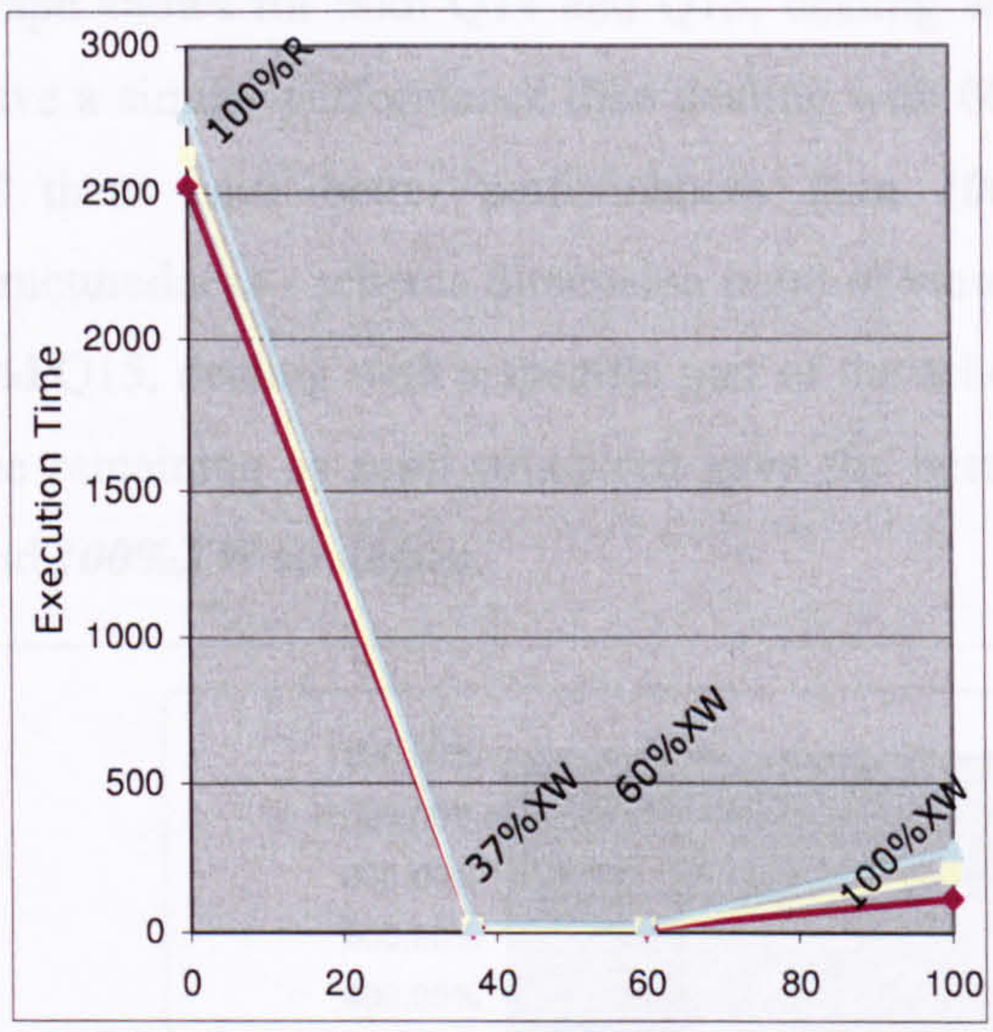
Q14 – Graph 2: Data Instances Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	19	7.53	76	13.50	219	23.33
100%XW	193	51.00	331	121.47	447	221.15
37%XW	14	1.55	19	7.00	20	8.52
60%XW	14	1.30	15	1.50	17	1.51
PSD	13	1.63	13	0.87	14	1.44

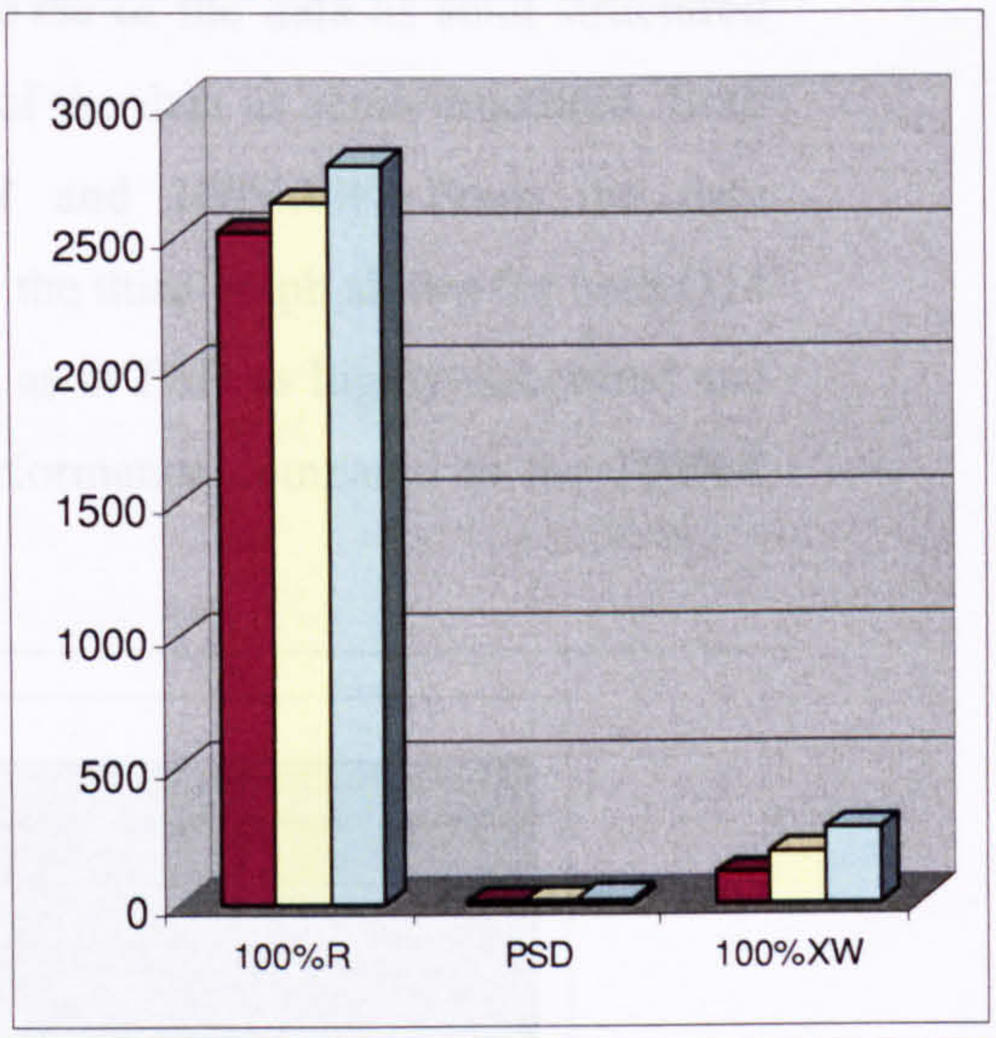
Figure 5.10 Query 14: Irregularity - Missing Elements



Q15 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q15 – Graph 2: Data Instances Dimension



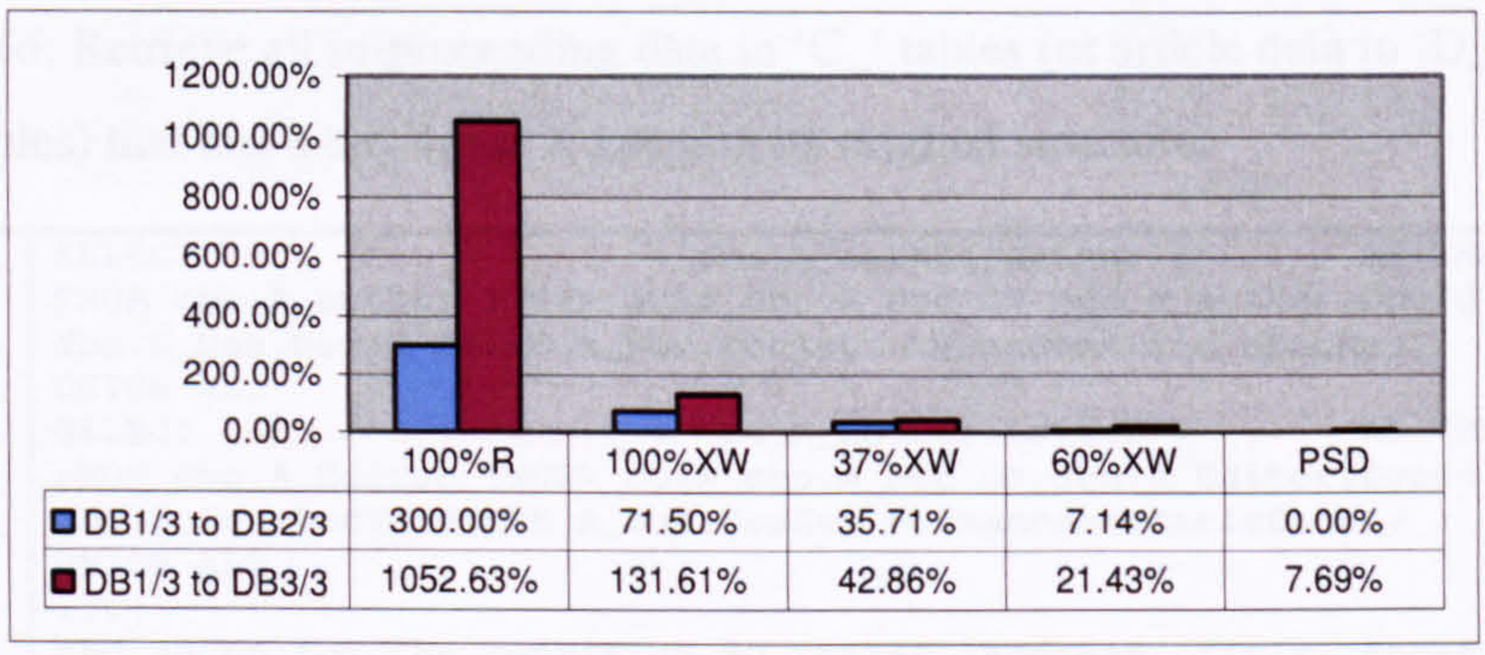
Q15 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	2519	144.27	2630	90.00	2772	27.85
100%XW	112	15.63	187	68.54	271	61.72
37%XW	15	1.39	15	1.71	20	7.60
60%XW	14	2.12	16	2.00	18	4.90
PSD	13	1.36	13	1.44	15	1.45

Figure 5.11 Query 15: Irregularity - Empty (Null) Values

From the storage strategy point of view, the query group "Irregularity of schema - missing elements" (Q14A) or "Irregularity of schema – for specific data" (Q15), the typed XML hybrid models showed the best query performance. For example in the ***DB3/3***, it took on average between 14 and 20 milliseconds for the hybrid model, 219 milliseconds for the relational model and 447 milliseconds for the ***100%XW*** in Q14. It took on average between 15 and 20 milliseconds for the hybrid model, 2772 milliseconds for the relational model and 271 milliseconds for the ***100%XW*** in Q15. In Q14, This was expected since the relational model had a sub-query to check the non existence of the '*ee*' data in a different table. Shredding the XML elements into small extracts (as in the hybrid model) provided a better performance than keeping the data in one large XML data field. In Q15, The main reason for the poor performance of the relational model was that the 'year' field did not have an index, besides it had a sub-query to check the non existence of the '*ee*' data in a different table.

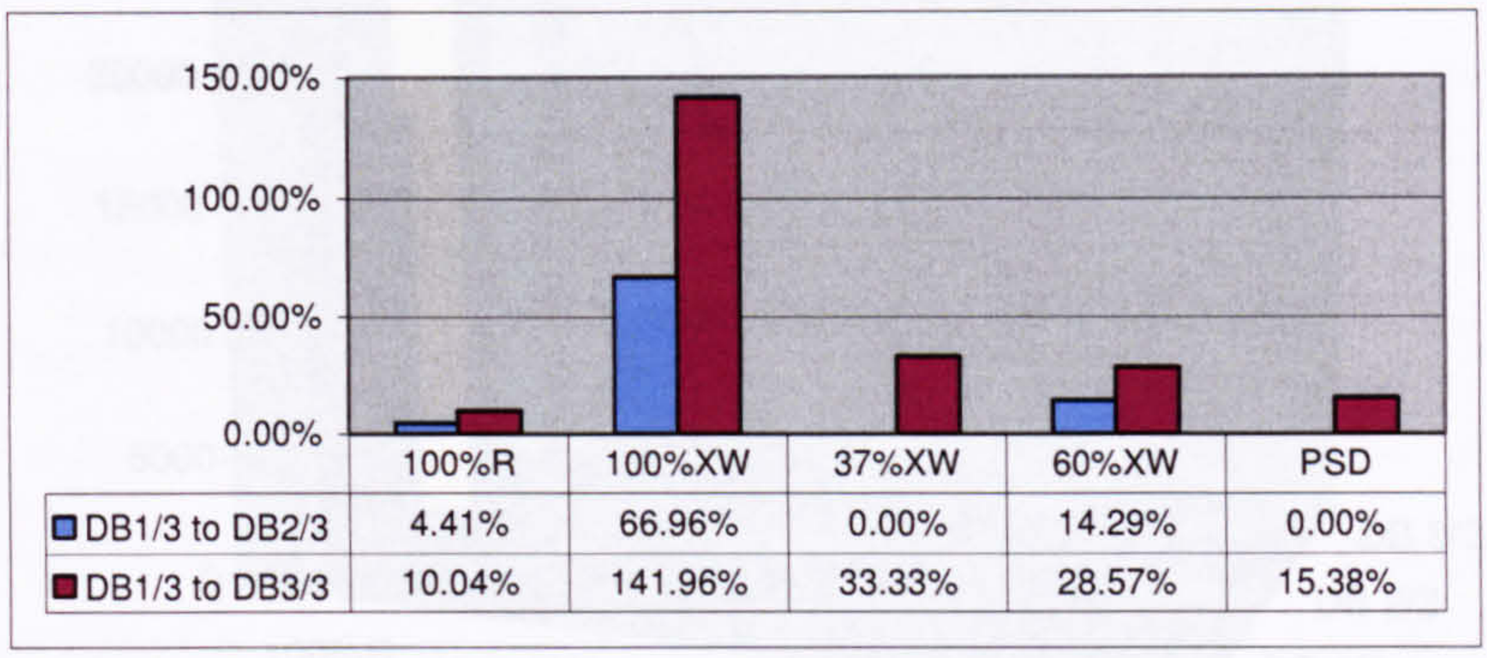
From the data structuredness - data instances dimension point of view, as the second graph shows for both Q14 and Q15, dealing with 37% of the data as semi-structured gave a similar performance than dealing with 60% of the data as semi-structured. Both of them gave better performances than ***100%R*** and ***100%XW***. From the data structuredness - schema dimension point of view, as the third graph shows for both Q14 and Q15, dealing with a specific part of the schema as in ***PSD*** as highly-structured and the remaining as semi-structured gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD
DB1/3 to DB2/3	300.00%	71.50%	35.71%	7.14%	0.00%
DB1/3 to DB3/3	1052.63%	131.61%	42.86%	21.43%	7.69%

Figure 5.12 Query 14: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure illustrates the performance deterioration when the database size doubled and tripled for Q14. The worst performance was for the model ***100%R.***



	100%R	100%XW	37%XW	60%XW	PSD
DB1/3 to DB2/3	4.41%	66.96%	0.00%	14.29%	0.00%
DB1/3 to DB3/3	10.04%	141.96%	33.33%	28.57%	15.38%

Figure 5.13 Query 15: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q15. The worst performance was for the model ***100%XW.***

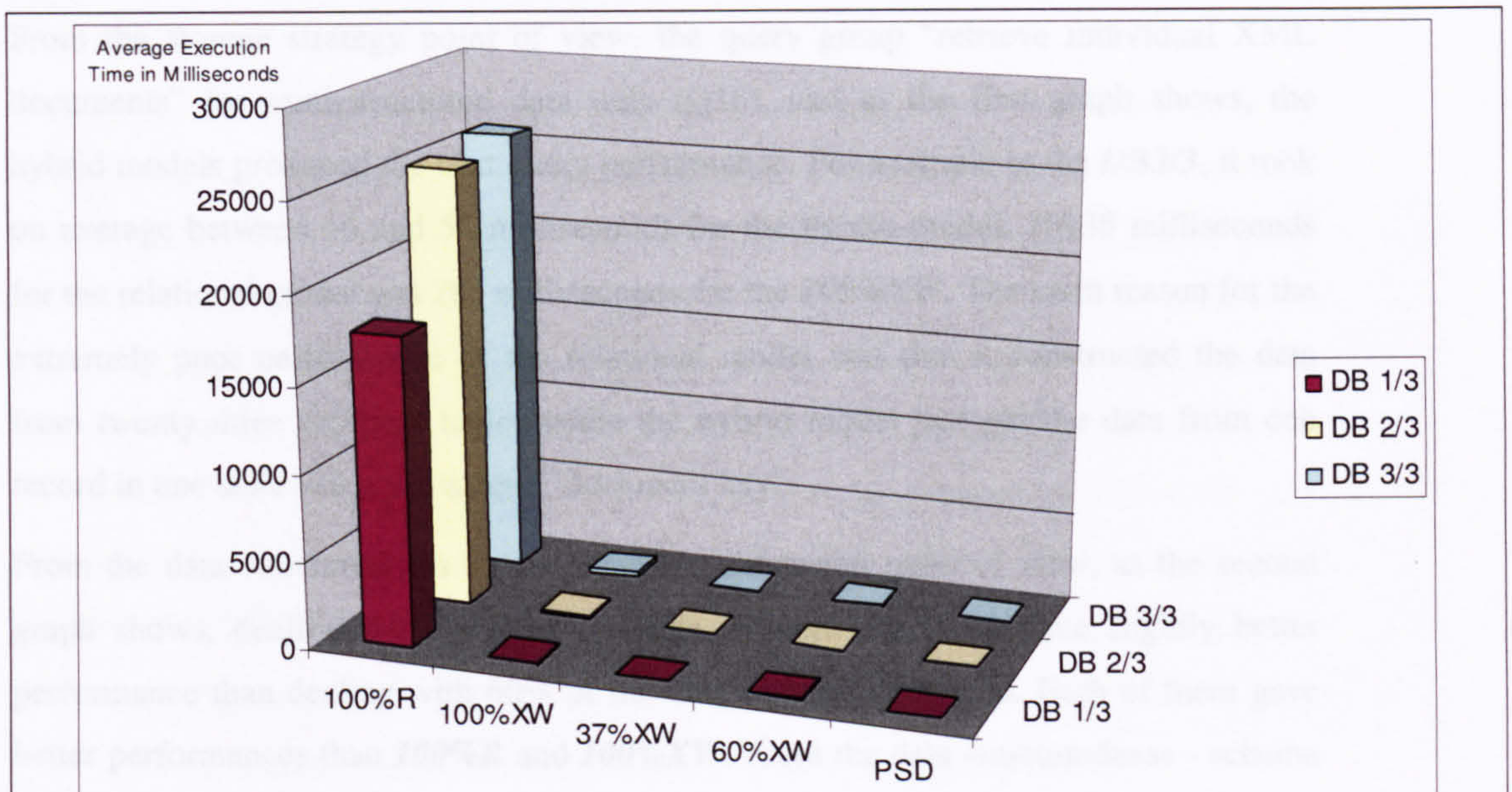
5.3.1.5 Retrieval of individual documents

The queries in this group tested an essential function to retrieve individual XML documents while preserving the contents of those documents. The queries in this group were interpreted to match the data set used in the experiments as follows:

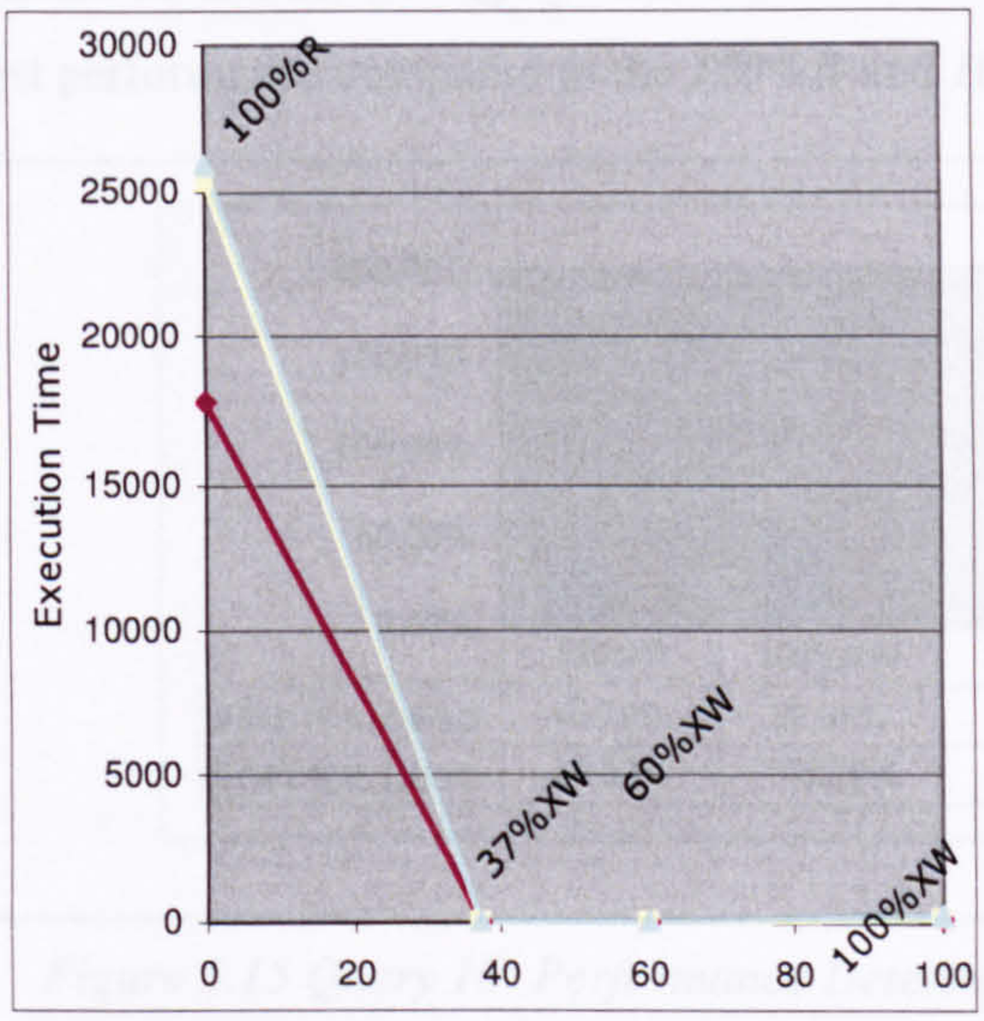
- ***Q16***: Retrieve all in-proceeding data in 'C_' tables (or article data in 'D_' tables) that had a key value X keeping its original structure.

SQL Syntax	SELECT cast ('<![CDATA[' + dbo.A_author.author + ']]>' AS XML) AS Q16A100R FROM dbo.A_author INNER JOIN dbo.A_Doc ON dbo.A_author.DocId = dbo.A_Doc.DocId WHERE A_Doc.DocKey = @RandomArticleDocKey UNION ALL SELECT cast ('<![CDATA[' + dbo.A_Editor.Editor + ']]>' AS XML) AS Q16A100R FROM dbo.A_Editor INNER JOIN dbo.A_Doc ON dbo.A_Editor.DocId = dbo.A_Doc.DocId WHERE A_Doc.DocKey = @RandomArticleDocKey UNION ALL ... And so on for the remaining 20 tables (Address, Title, Booktitle, Pages, Year, Journal, volume, month, URL, EE, CDRom, Cite, Publisher, CrossRef, ISBN, Series, School, Chapter, Number, Note)
XQuery Syntax	SELECT xmldoc.query('/dblp/article[@key="' + @RandomArticleDocKey + '"]') AS Q16A100X From B_XMLDocument;
SQL/XQuery Syntax	SELECT XMLExtract AS Q16A60X from C_Doc WHERE DocKey = @RandomInProceedingsDocKey;

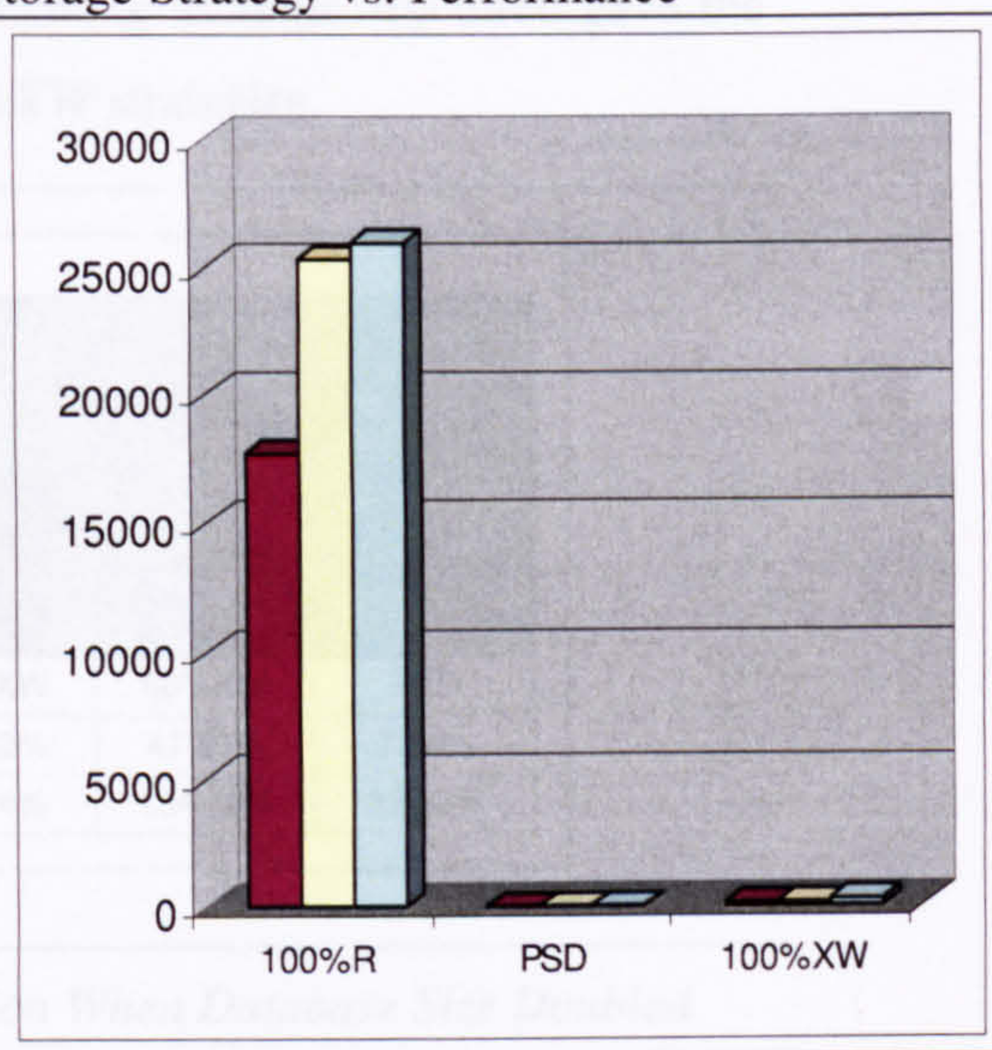
The following graphs show the results for this query



Q16 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q16 – Graph 2: Data Instances Dimension



Q16 – Graph 3: Schema Dimension

	DB 1/3		DB 2/3		DB 3/3	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	17721	570.33	25304	1042.22	25938	313.36
100%XW	107	40.36	163	36.46	263	107.13
37%XW	19	7.49	24	1.73	37	18.28
60%XW	23	8.78	34	2.08	54	16.60
PSD	13	1.00	14	1.56	16	1.38

Figure 5.14 Query 16: Retrieve Individual Documents

From the storage strategy point of view, the query group "retrieve individual XML documents" for semi-structured data only (Q16), and as the first graph shows, the hybrid models produced the best query performance. For example in the ***DB3/3***, it took on average between 16 and 54 milliseconds for the hybrid model, 25938 milliseconds for the relational model and 263 milliseconds for the ***100%XW***. The main reason for the extremely poor performance of the relational model was that it constructed the data from twenty three different tables while the hybrid model just got the data from one record in one table using the unique "document key".

From the data structuredness - data instances dimension point of view, as the second graph shows, dealing with 37% of the data as semi-structured gave slightly better performance than dealing with 60% of the data as semi-structured. Both of them gave better performances than ***100%R*** and ***100%XW***. From the data structuredness - schema dimension point of view, as the third graph shows, dealing with a specific part of the schema as in ***PSD*** as highly-structured and the remaining as semi-structured gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.

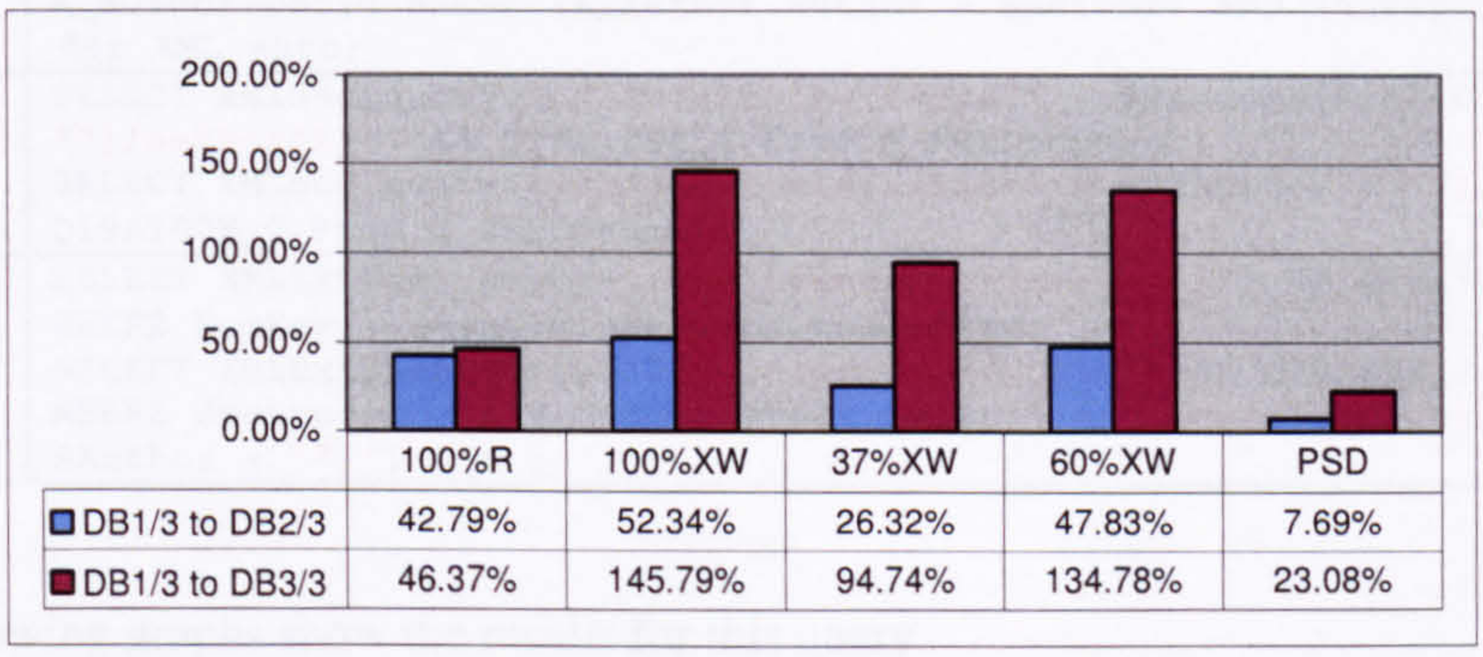


Figure 5.15 Query 16: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q16. The worst performance was for the model ***100%XW.***

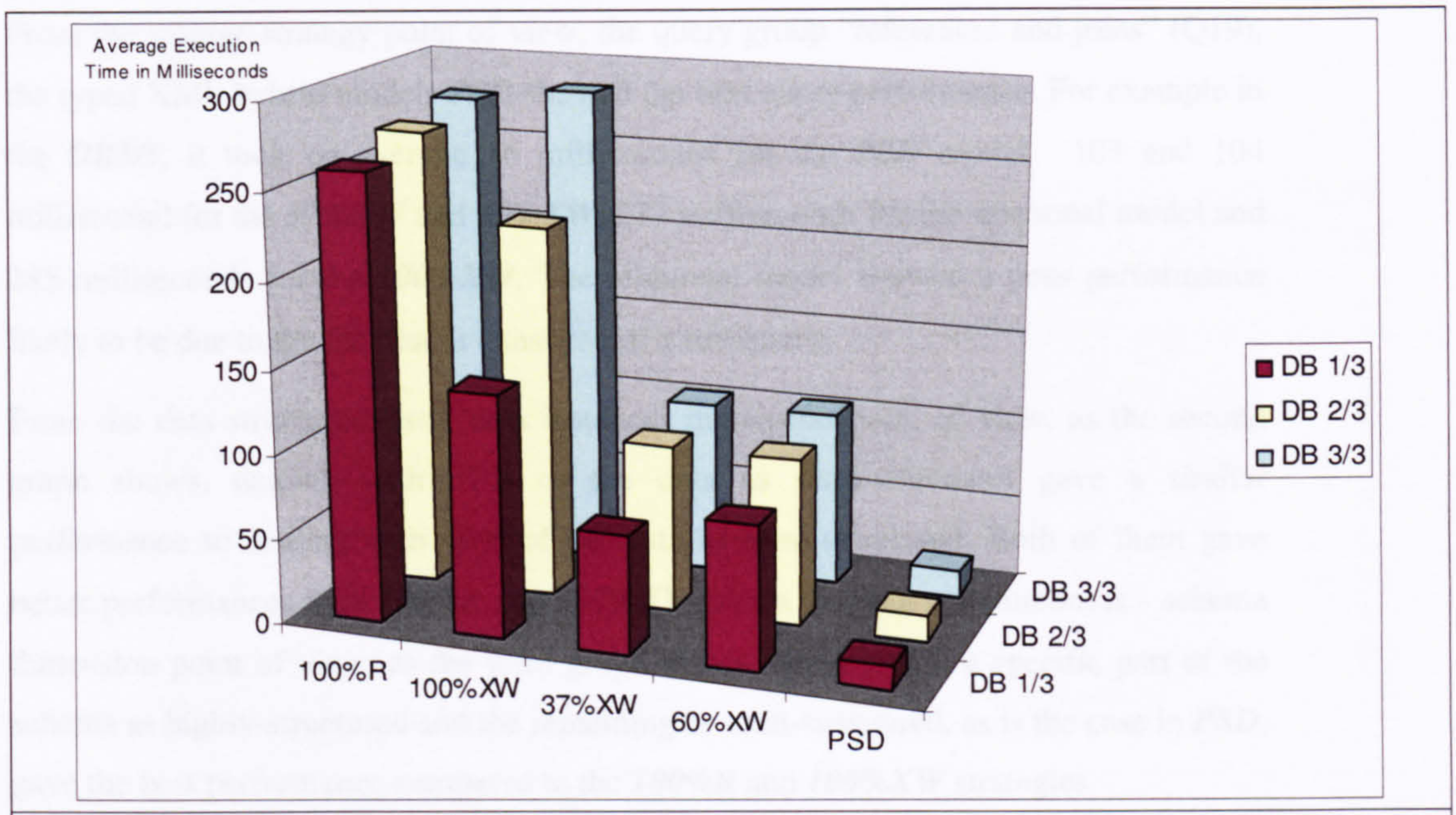
5.3.1.6 References and joins

Data-centric documents usually have references to identify the relationship between related data, even among different XML documents. Sometimes users want to combine separate information together using join by values. The original version of this query tests the references and joins. Since the data set consisted of a single XML document, and as this query required the join operation between two documents, it was executed in two steps. In the first step involved finding an author of a specified article (or in-proceeding), then in the second step, by using this author's name, another query was executed to retrieve all the articles he/she wrote. The queries in this group were interpreted to match the data set used in the experiments as follows:

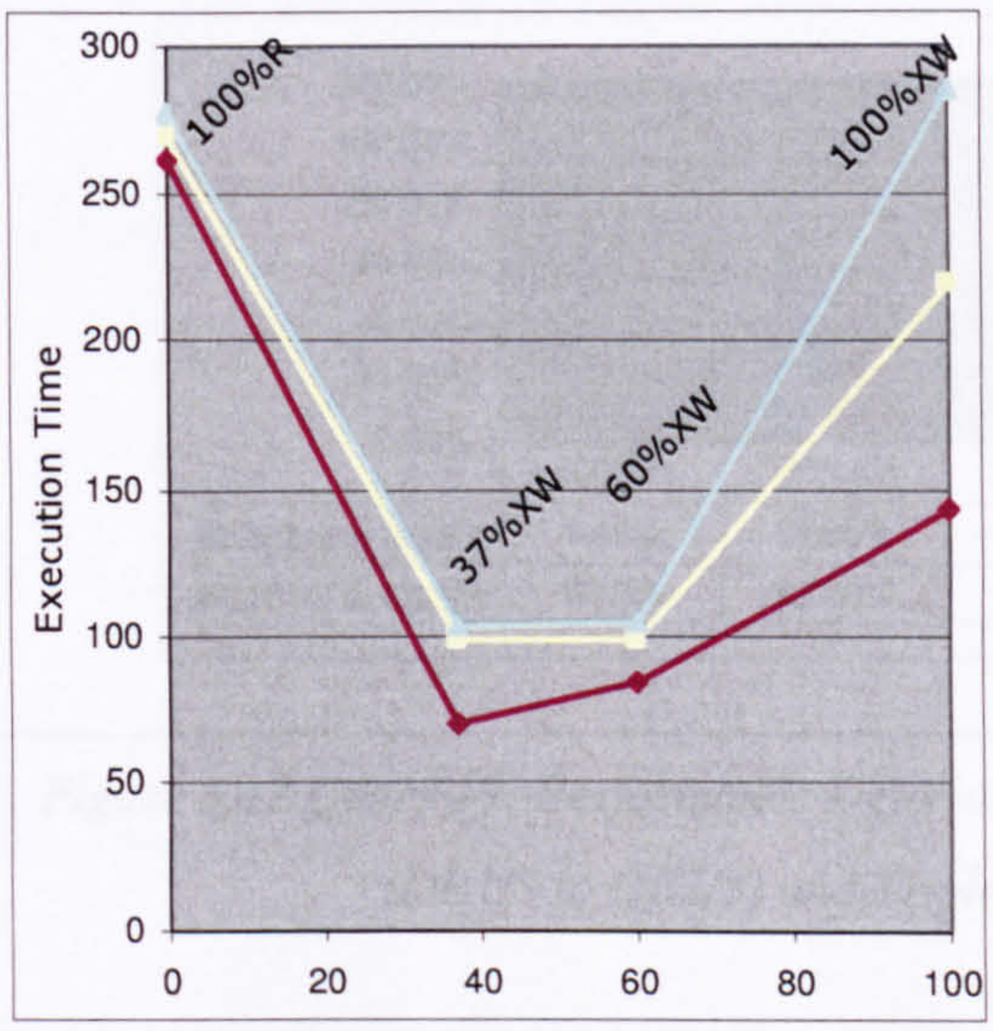
- ***Q19***: Retrieve the first author for in-proceeding data in 'C_' tables (or article data in 'D_' tables) that has a key value X. Using this author; retrieve all his publications' titles.

SQL Syntax	SELECT Top 1 @Author = Author from A_Author WHERE DocId in (SELECT DocId from A_Doc WHERE Dockey = @RandomArticleDocKey) SELECT A_Title.Title AS Q19A100R_2 FROM A_Title INNER JOIN A_Doc ON A_Title.DocId = A_Doc.DocId INNER JOIN A_Author ON A_Doc.DocId = A_Author.DocId WHERE (A_Author.Author = @Author) AND (A_Doc.DocTypeId = 1) for XML auto;
XQuery Syntax	SELECT xmldoc.query('/dblp/article[@key="' + @RandomArticleDocKey + '"]/author[1]') AS Q19A100X_1 From B_XMLDocument; SELECT XMLdoc.query('/dblp/article[author="N'+ @Author + '"]/title') AS Q19A100X_2 From B_XMLDocument
SQL/XQuery Syntax	SELECT XMLExtract.query('/inproceedings/author[1]') AS Q19A60X_1 From C_Doc WHERE DocKey = @RandomInProceedingsDocKey; SELECT XMLExtract.query('/inproceedings/title') AS Q19A60X_2 From C_Doc WHERE doctypeid = 2 and XMLExtract.exist('/inproceedings[author="N' + @Author + '"]') = 1 ;

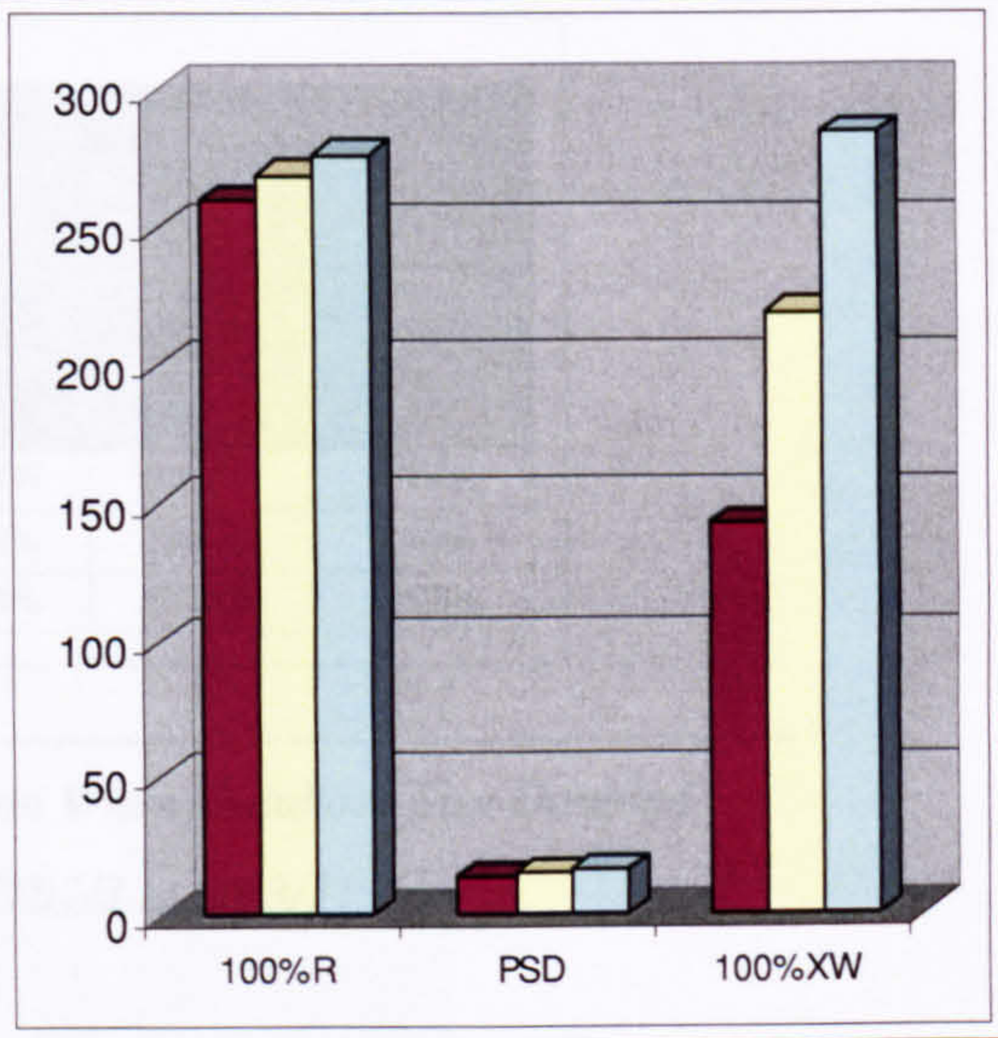
The following graphs show the results for this query



Q19 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q19 – Graph 2: Data Instances Dimension



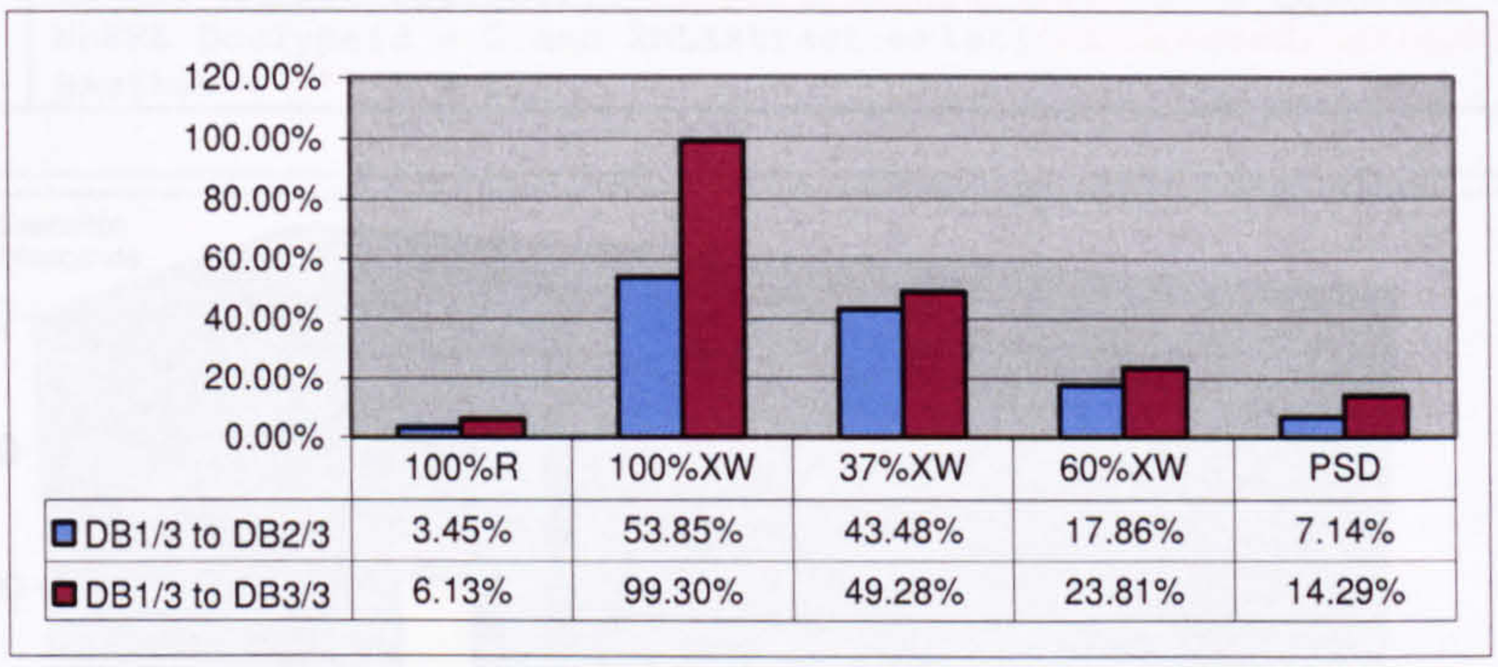
Q19 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	261	7.07	270	17.32	277	110.44
100%XW	143	52.88	220	39.47	285	59.26
37%XW	69	10.33	99	2.12	103	14.62
60%XW	84	10.74	99	11.09	104	7.30
PSD	14	1.52	15	1.29	16	1.83

Figure 5.16 Query 19: Reference and Joins

From the storage strategy point of view, the query group "references and joins" (Q19), the typed XML hybrid models ***PSD*** showed the best query performance. For example in the ***DB3/3***, it took on average 16 milliseconds for the ***PSD*** model, 103 and 104 millisecond for the ***37%XW*** and ***60%XW***, 277 milliseconds for the relational model and 285 milliseconds for the ***100%XW***. The relational model showed a poor performance likely to be due to the fact that it consisted of a sub-query.

From the data structuredness - data instances dimension point of view, as the second graph shows, dealing with 37% of the data as semi-structured gave a similar performance to dealing with 60% of the data as semi-structured. Both of them gave better performances than ***100%R*** and ***100%XW***. From the data structuredness - schema dimension point of view, as the third graph shows, dealing with a specific part of the schema as highly-structured and the remaining as semi-structured, as is the case in ***PSD***, gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD
DB1/3 to DB2/3	3.45%	53.85%	43.48%	17.86%	7.14%
DB1/3 to DB3/3	6.13%	99.30%	49.28%	23.81%	14.29%

Figure 5.17 Query 19: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q19. The worst performance was for the model ***100%XW.***

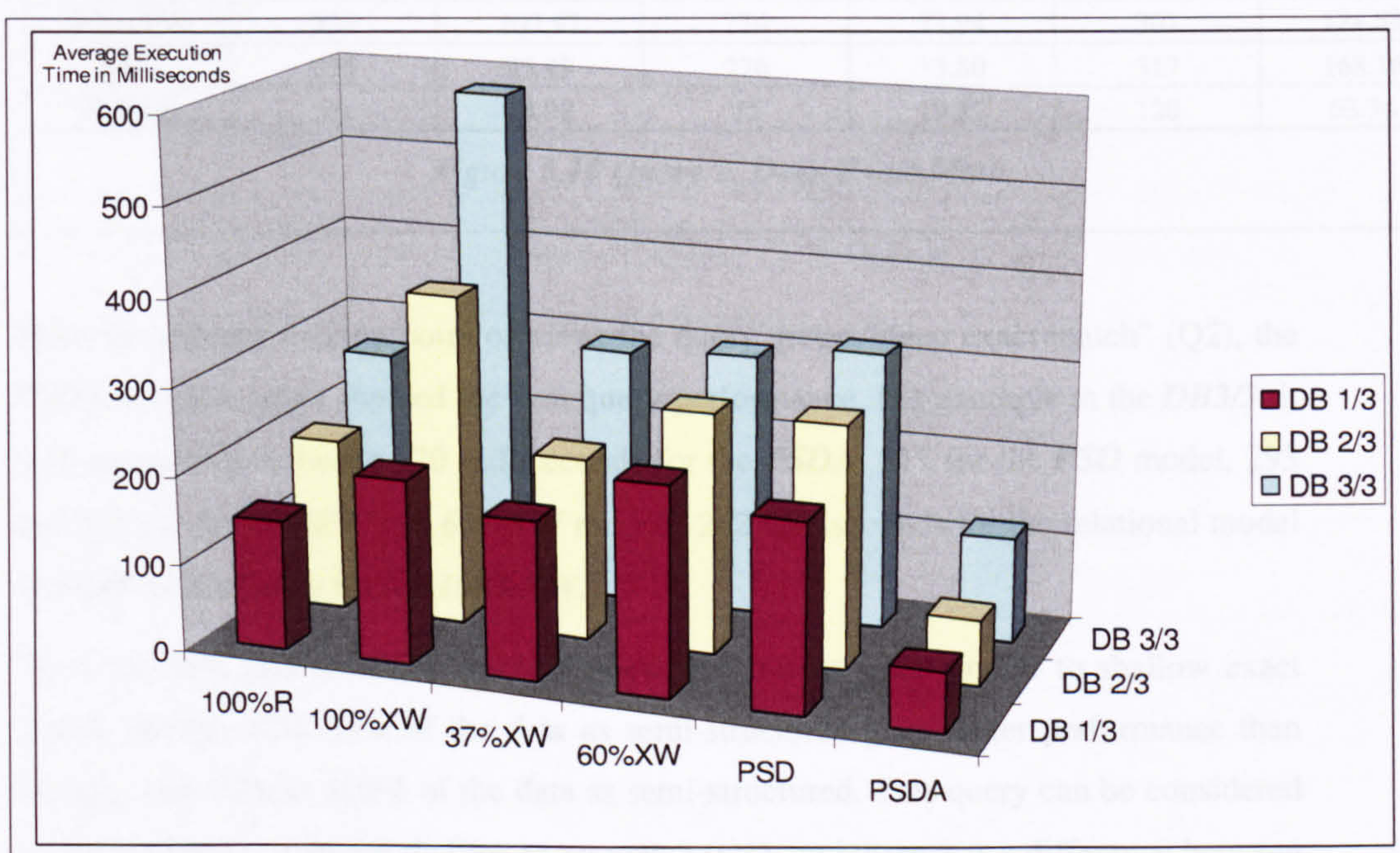
5.3.2 Using 'Author' Queries

5.3.2.1 Exact Match (Deep)

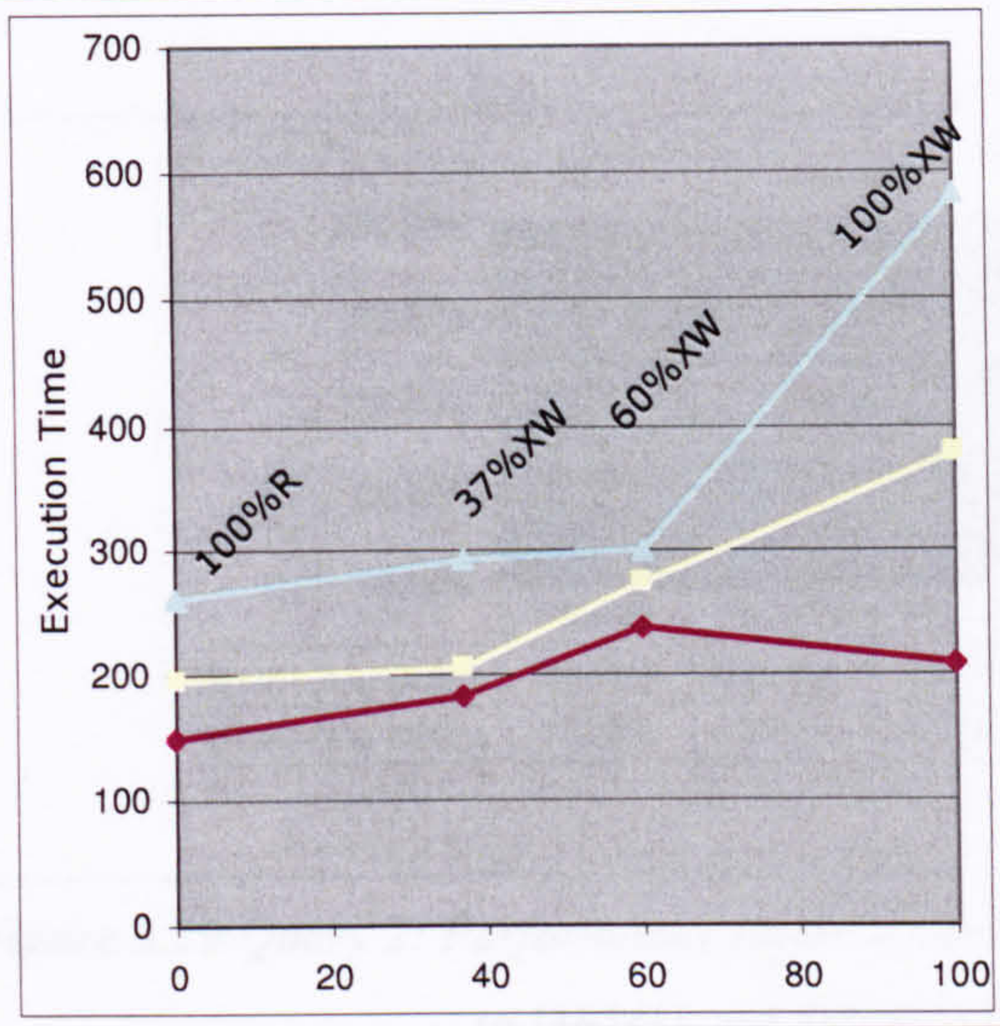
The queries in this group required exact string match with specified and possibly long path expressions, depending on the levels of predicates being queried in the XML documents. Consequently, they can be shallow queries that match only at the top level of XML document trees (example Q1, this query was presented in section 5.3.1.1), or deep queries that match the nested structure of an XML document tree (Q2).

- ***Q2***: Return in-proceeding's titles (or an article's titles) that has the same author Y.

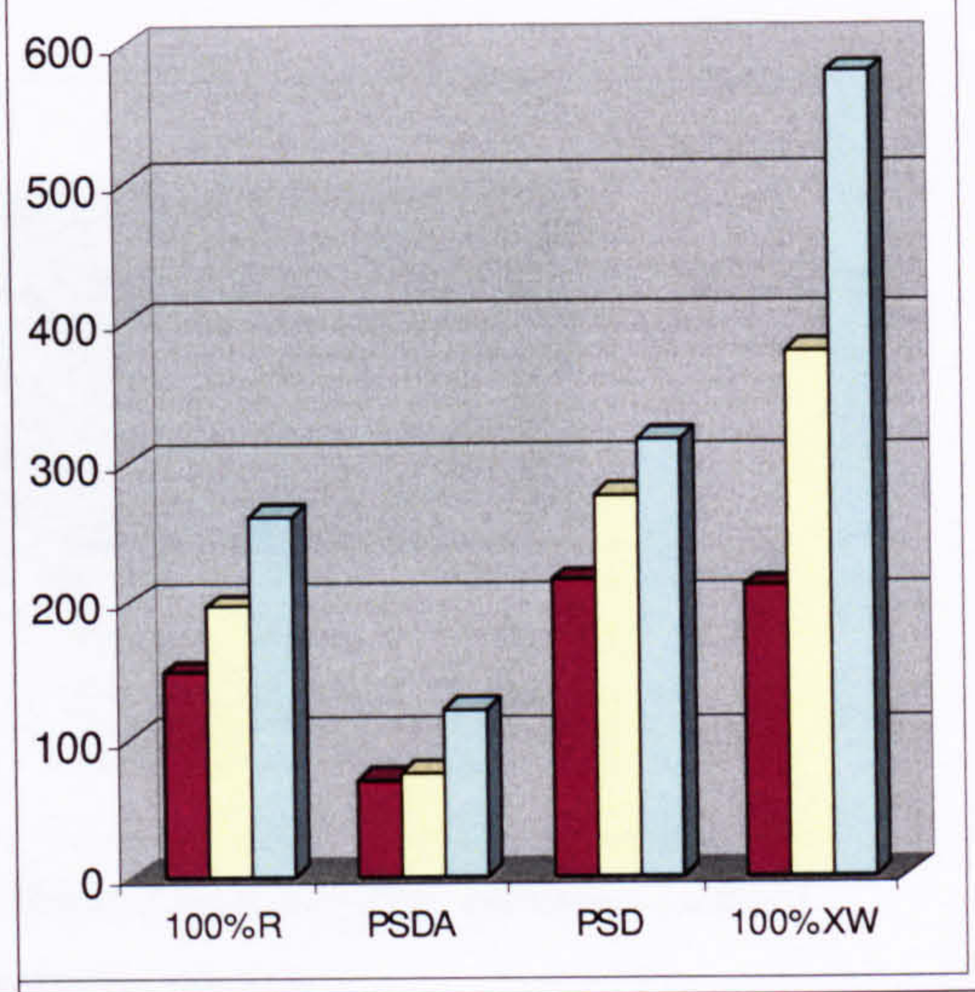
SQL Syntax	`SELECT A_Title.Title as Q02A100R FROM A_Title INNER JOIN A_Doc ON A_Title.DocId = A_Doc.DocId INNER JOIN A_Author ON A_Doc.DocId = A_Author.DocId WHERE (A_Author.Author = @Author) AND (A_Doc.DocTypeId = 1) for XML auto;`
XQuery Syntax	`SELECT XMLdoc.query('/dblp/article[author="'+ @Author + '"]/title') as Q02A100XW FROM B_XMLDocument;`
SQL/XQuery Syntax	`SELECT XMLExtract.query('/inproceedings/title') as Q02A60XW FROM C_Doc WHERE DocTypeId = 2 and XMLExtract.exist('/inproceedings[author="' + @Author + '"]') = 1;`



Q2 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q2 – Graph 2: Data Instances Dimension



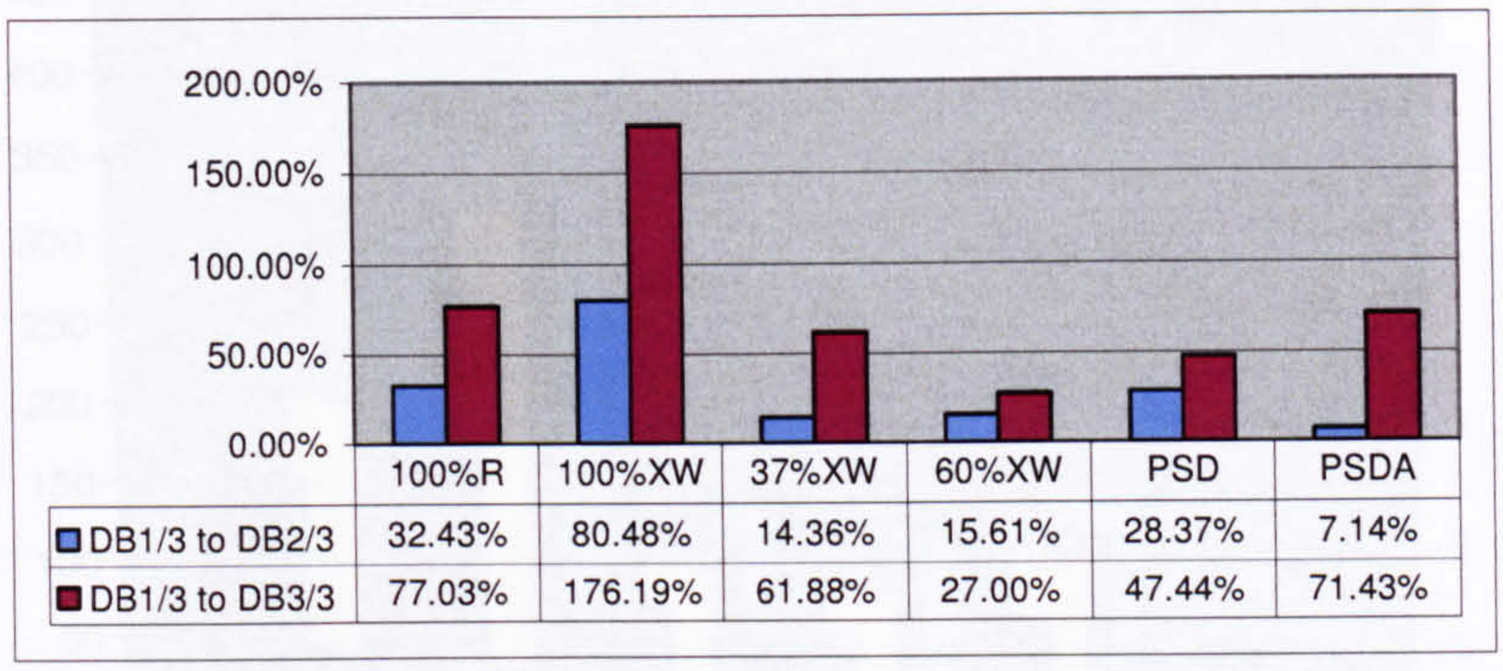
Q2 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	148	26.94	196	94.89	262	40.93
100%XW	210	59.34	379	90.90	580	144.30
37%XW	181	75.33	207	54.65	293	121.92
60%XW	237	102.97	274	73.94	301	124.83
PSD	215	35.99	276	13.80	317	168.16
PSDA	70	23.98	75	19.27	120	63.36

Figure 5.18 Query 2: Deep Exact Math

From the storage strategy point of view, the query group "deep exact match" (Q2), the ***PSDA*** hybrid models showed the best query performance. For example in the ***DB3/3***, it took on average between 120 milliseconds for the ***PSDA***, 317 for the ***PSD*** model, 293 and 301 for the ***37%XW*** and ***60%XW*** models, 262 milliseconds for the relational model and 580 milliseconds for the ***100%XW***.

From the data structuredness point of view, the results were similar to shallow exact match, dealing with 37% of the data as semi-structured gave better performance than dealing with 60% or 100% of the data as semi-structured. This query can be considered as more complex than the shallow exact match (Q1), and there was a difference between the performance of Q1 and Q2. The reason for that was probably due to the technique for selecting the data for this query, which involved retrieving the data from inside each XML extract versus using the document key value as in Q1.



	100%R	100%XW	37%XW	60%XW	PSD	PSDA
DB1/3 to DB2/3	32.43%	80.48%	14.36%	15.61%	28.37%	7.14%
DB1/3 to DB3/3	77.03%	176.19%	61.88%	27.00%	47.44%	71.43%

Figure 5.19 Query 2: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure illustrates the performance deterioration when the database size doubled and tripled for Q2. The worst performance was for the model ***100%XW.***

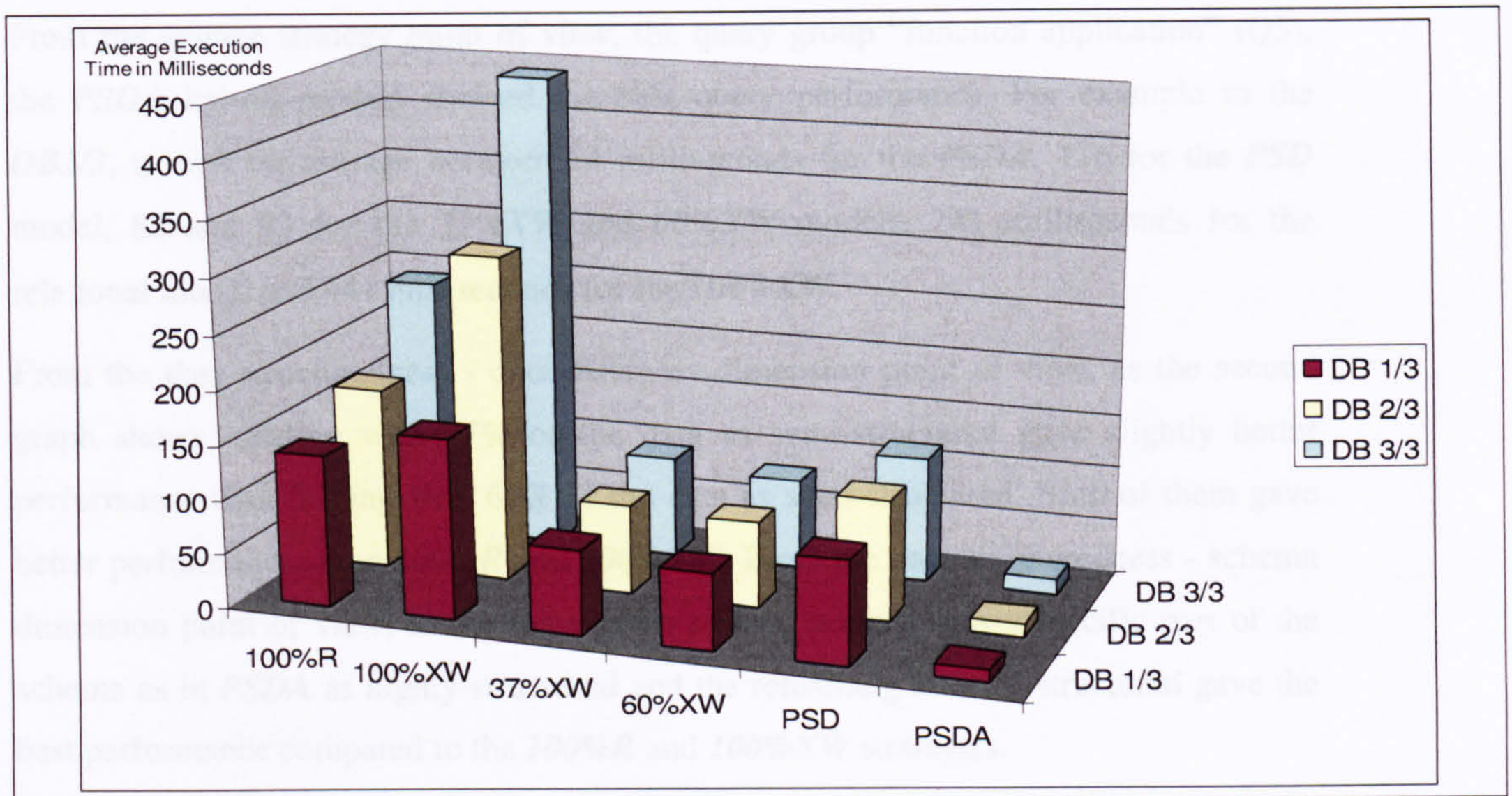
5.3.2.2 Function application

The queries in this group challenged the system with aggregate functions such as count, average, maximum, minimum and sum. The queries in this group were interpreted to match the data set used in the experiments as follows:

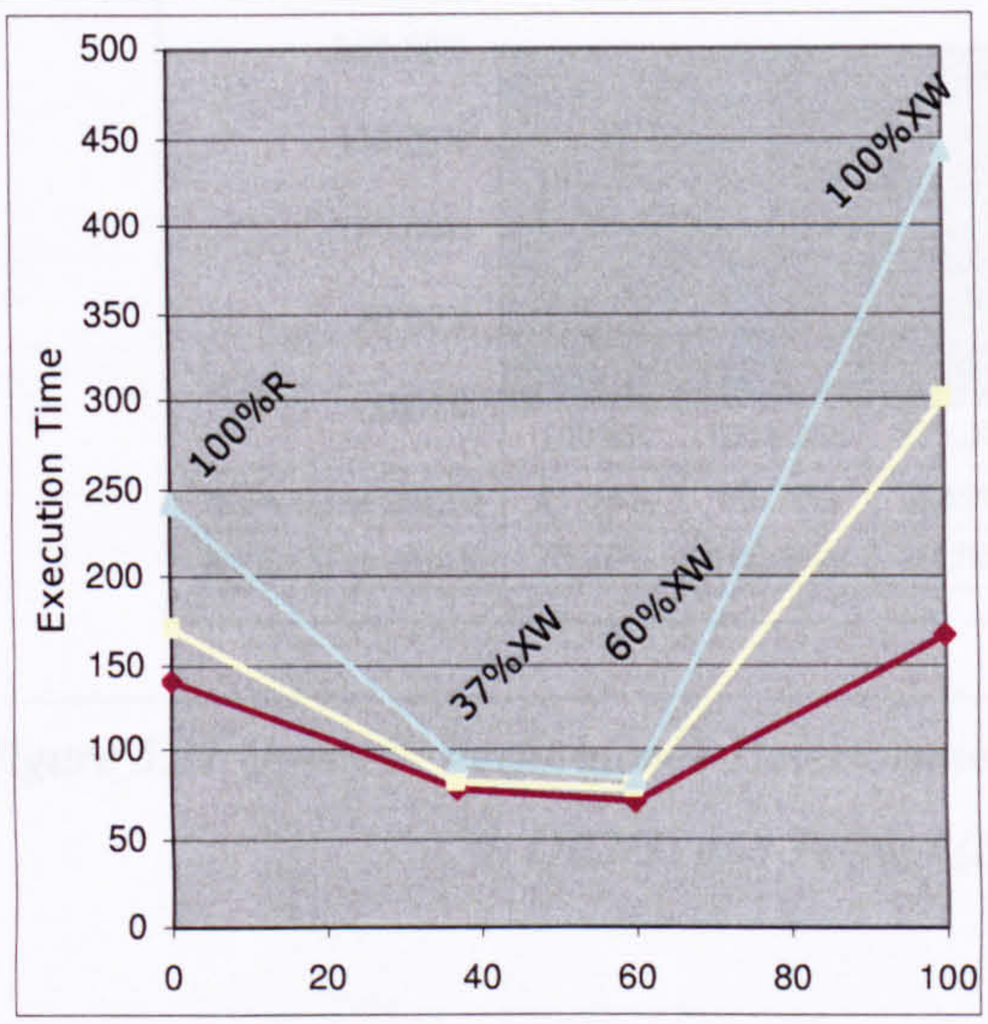
- ***Q3***: counts in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables) that have the same author X.

SQL Syntax	`SELECT COUNT(A_Doc.DocId) AS Q03A100R FROM A_Doc INNER JOIN A_Author ON A_Doc.DocId = A_Author.DocId WHERE (A_Author.Author = @Author) AND (A_Doc.DocTypeId = 1);`
XQuery Syntax	`SELECT XMLdoc.query('COUNT(/dblp/article[author="'+ @Author + '"]/title)') AS Q03A100XW FROM B_XMLDocument`
SQL/XQuery Syntax	`SELECT COUNT(XMLExtract.query('/inproceedings/title')) AS Q03A60X FROM C_doc WHERE doctypeid = 2 and XMLExtract.exist('/inproceedings[author="' + @Author + '"]') = 1;`

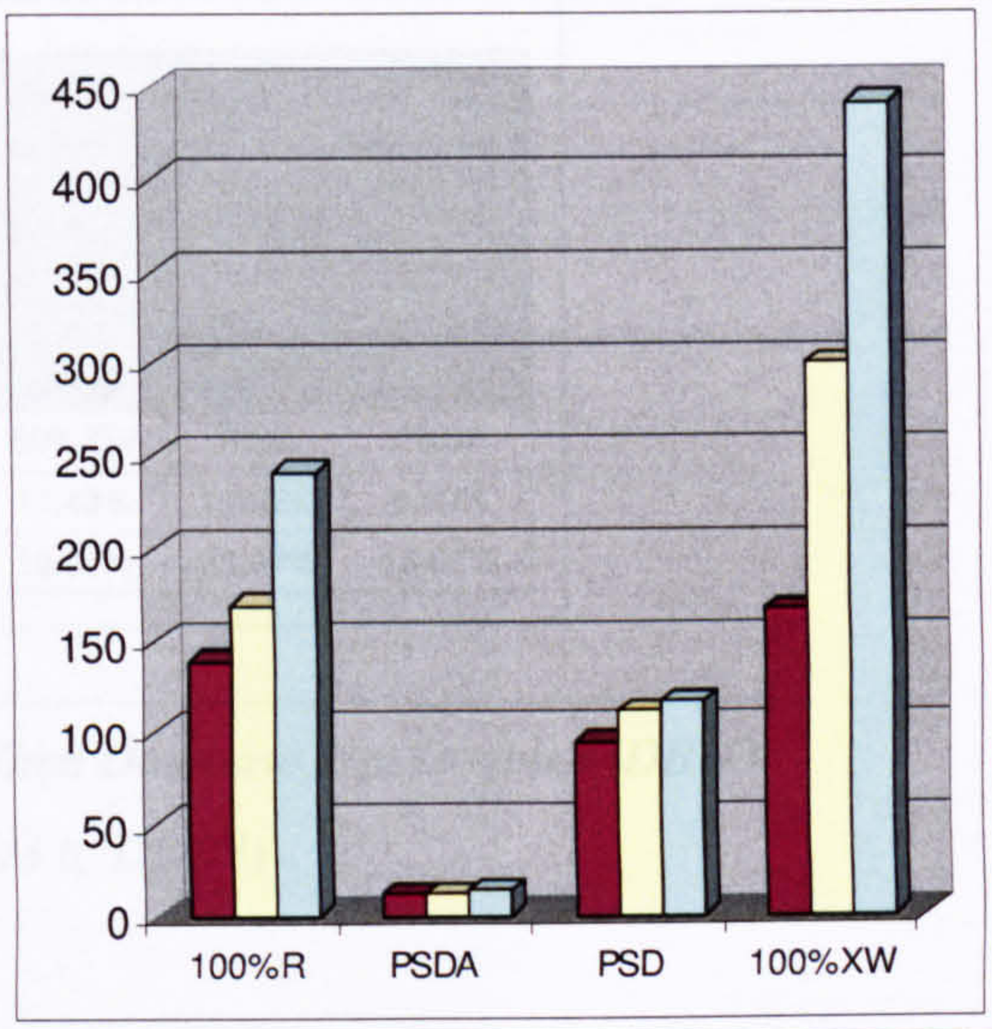
The following graphs show the results for this query



Q3 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q3 – Graph 2: Data Instances Dimension



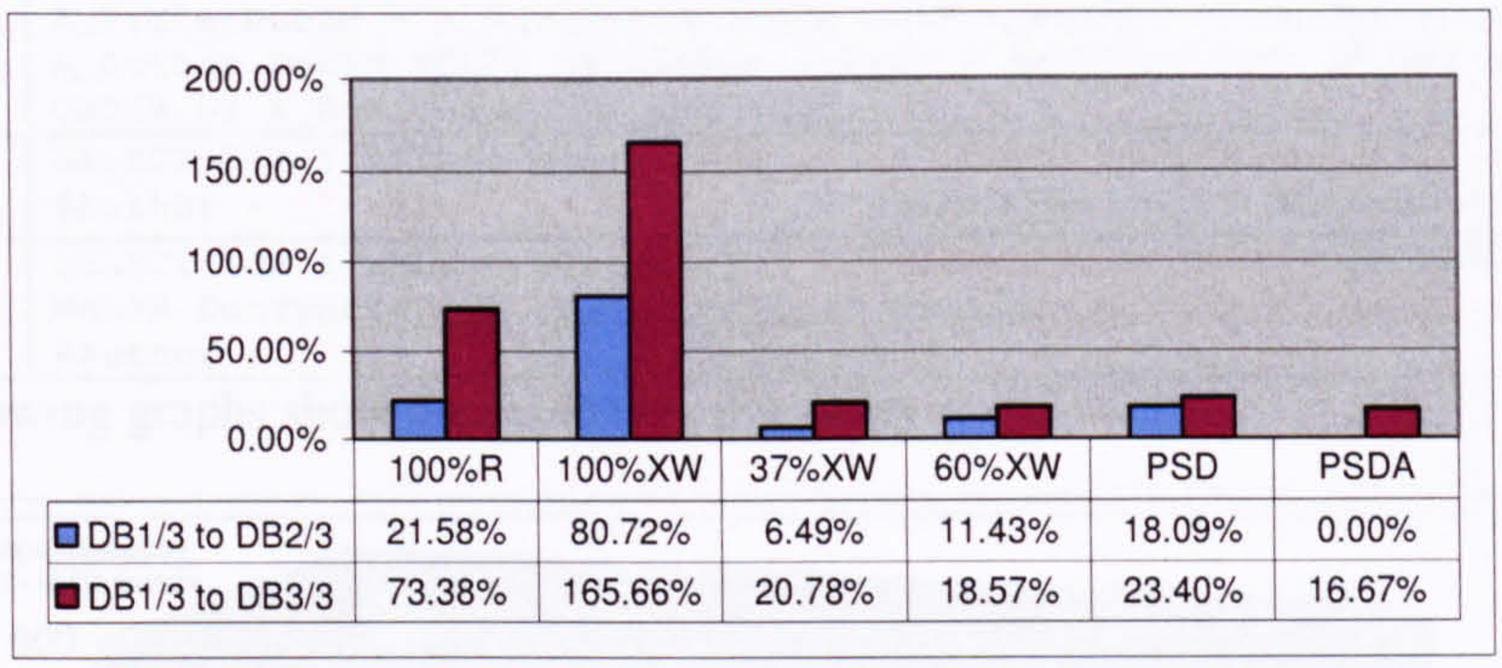
Q3 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	139	23.98	169	39.29	241	18.13
100%XW	166	63.49	300	66.31	441	96.34
37%XW	77	1.92	82	8.05	93	2.57
60%XW	70	8.07	78	5.63	83	9.10
PSD	94	17.81	111	23.09	116	3.25
PSDA	12	0.44	12	0.45	14	1.48

Figure 5.20 Query 3: Function Application

From the storage strategy point of view, the query group "function application" (Q3), the ***PSDA*** hybrid models showed the best query performance. For example in the ***DB3/3***, it took on average between 14 milliseconds for the ***PSDA***, 116 for the ***PSD*** model, 83 and 93 for the ***37%XW*** and ***60%XW*** models, 241 milliseconds for the relational model and 441 milliseconds for the ***100%XW***.

From the data structuredness - data instances dimension point of view, as the second graph shows, dealing with 37% of the data as semi-structured gave slightly better performance than dealing with 60% of the data as semi-structured. Both of them gave better performances than ***100%R*** and ***100%XW***. From the data structuredness - schema dimension point of view, as the third graph shows, dealing with a specific part of the schema as in ***PSDA*** as highly-structured and the remaining as semi-structured gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD	PSDA
DB1/3 to DB2/3	21.58%	80.72%	6.49%	11.43%	18.09%	0.00%
DB1/3 to DB3/3	73.38%	165.66%	20.78%	18.57%	23.40%	16.67%

Figure 5.21 Query 2: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure illustrates the performance deterioration when the database size doubled and tripled for Q2. The worst performance was for the model ***100%XW***.

5.3.2.3 Ordered access

The queries in this group test the performance of the system when it preserves the document order during retrieval. This could be in a relative order (Q4) based on the current matching position, or an absolute order (Q5). The queries in this group were interpreted to match the data set used in the experiments as follows:

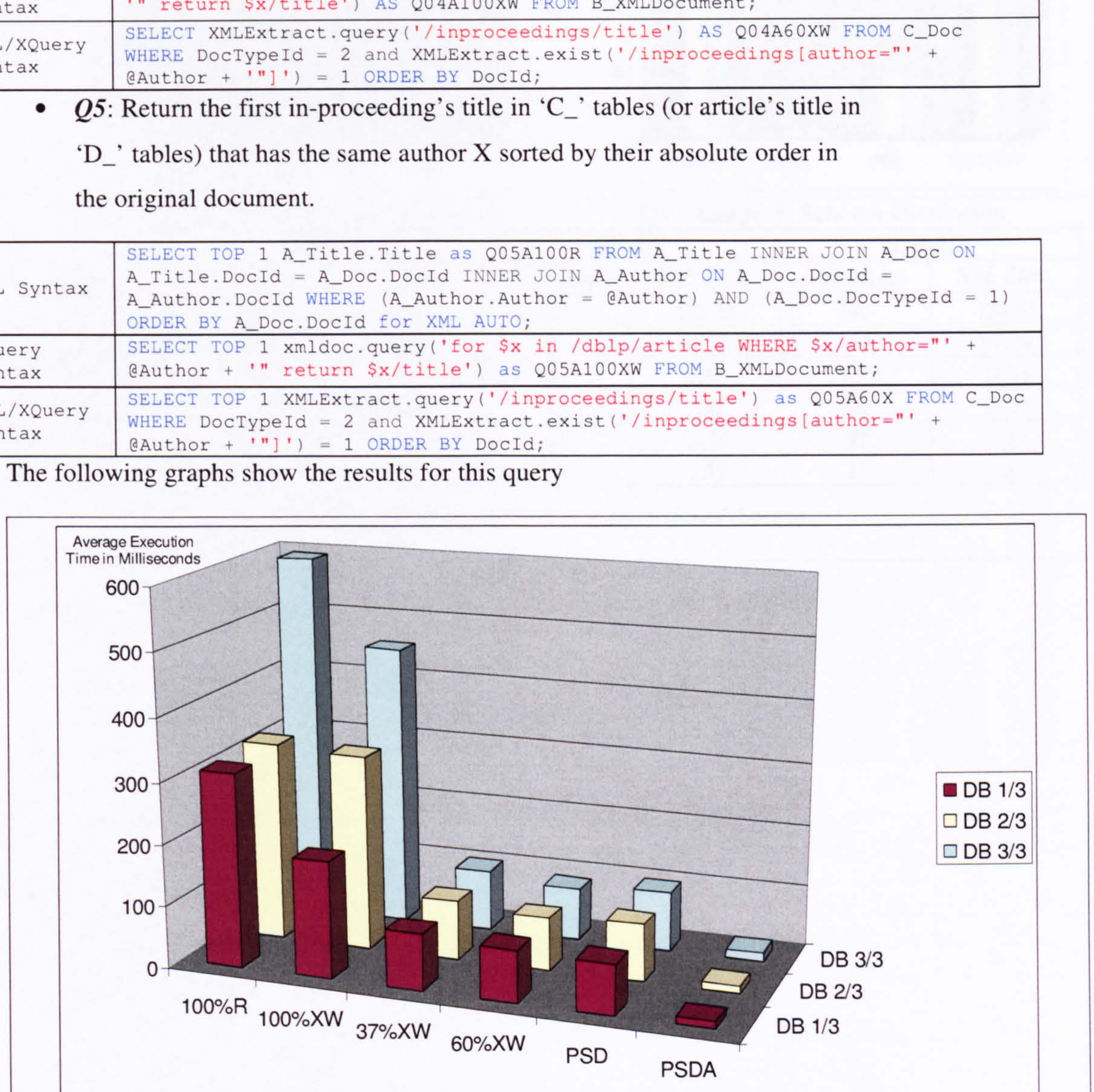
- ***Q4***: Return in-proceeding's titles in 'C_' tables (or an article's titles in 'D_' tables) that has the same author X. in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables) that have the same author X sorted by their relative order in the original document.

SQL Syntax	`SELECT A_Title.Title AS Q04A100R FROM A_Title INNER JOIN A_Doc ON A_Title.DocId = A_Doc.DocId INNER JOIN A_Author ON A_Doc.DocId = A_Author.DocId WHERE (A_Author.Author = @Author) AND (A_Doc.DocTypeId = 1) ORDER BY A_Doc.DocId for XML AUTO;`
XQuery Syntax	`SELECT xmldoc.query('for $x in /dblp/article WHERE $x/author="' + @Author + '" return $x/title') AS Q04A100XW FROM B_XMLDocument;`
SQL/XQuery Syntax	`SELECT XMLExtract.query('/inproceedings/title') AS Q04A60XW FROM C_Doc WHERE DocTypeId = 2 and XMLExtract.exist('/inproceedings[author="' + @Author + '"]') = 1 ORDER BY DocId;`

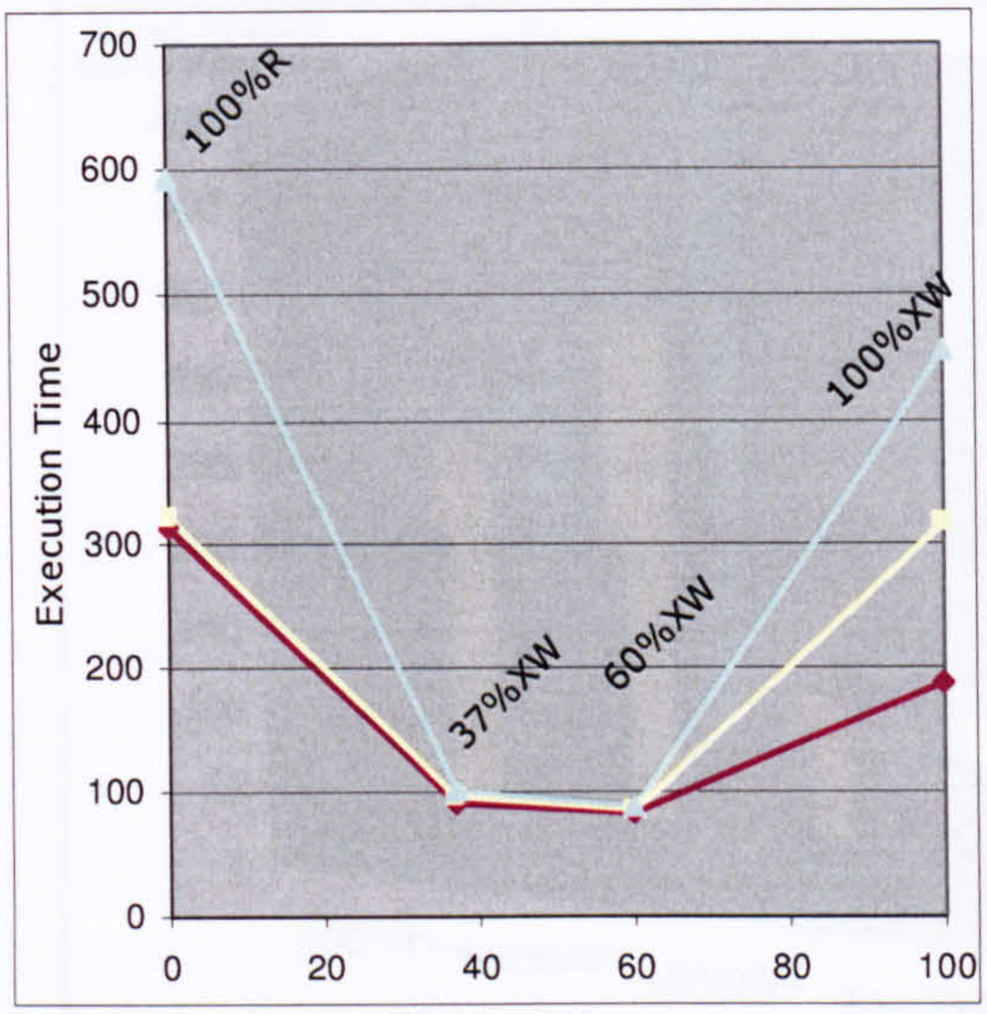
- ***Q5***: Return the first in-proceeding's title in 'C_' tables (or article's title in 'D_' tables) that has the same author X sorted by their absolute order in the original document.

SQL Syntax	`SELECT TOP 1 A_Title.Title as Q05A100R FROM A_Title INNER JOIN A_Doc ON A_Title.DocId = A_Doc.DocId INNER JOIN A_Author ON A_Doc.DocId = A_Author.DocId WHERE (A_Author.Author = @Author) AND (A_Doc.DocTypeId = 1) ORDER BY A_Doc.DocId for XML AUTO;`
XQuery Syntax	`SELECT TOP 1 xmldoc.query('for $x in /dblp/article WHERE $x/author="' + @Author + '" return $x/title') as Q05A100XW FROM B_XMLDocument;`
SQL/XQuery Syntax	`SELECT TOP 1 XMLExtract.query('/inproceedings/title') as Q05A60X FROM C_Doc WHERE DocTypeId = 2 and XMLExtract.exist('/inproceedings[author="' + @Author + '"]') = 1 ORDER BY DocId;`

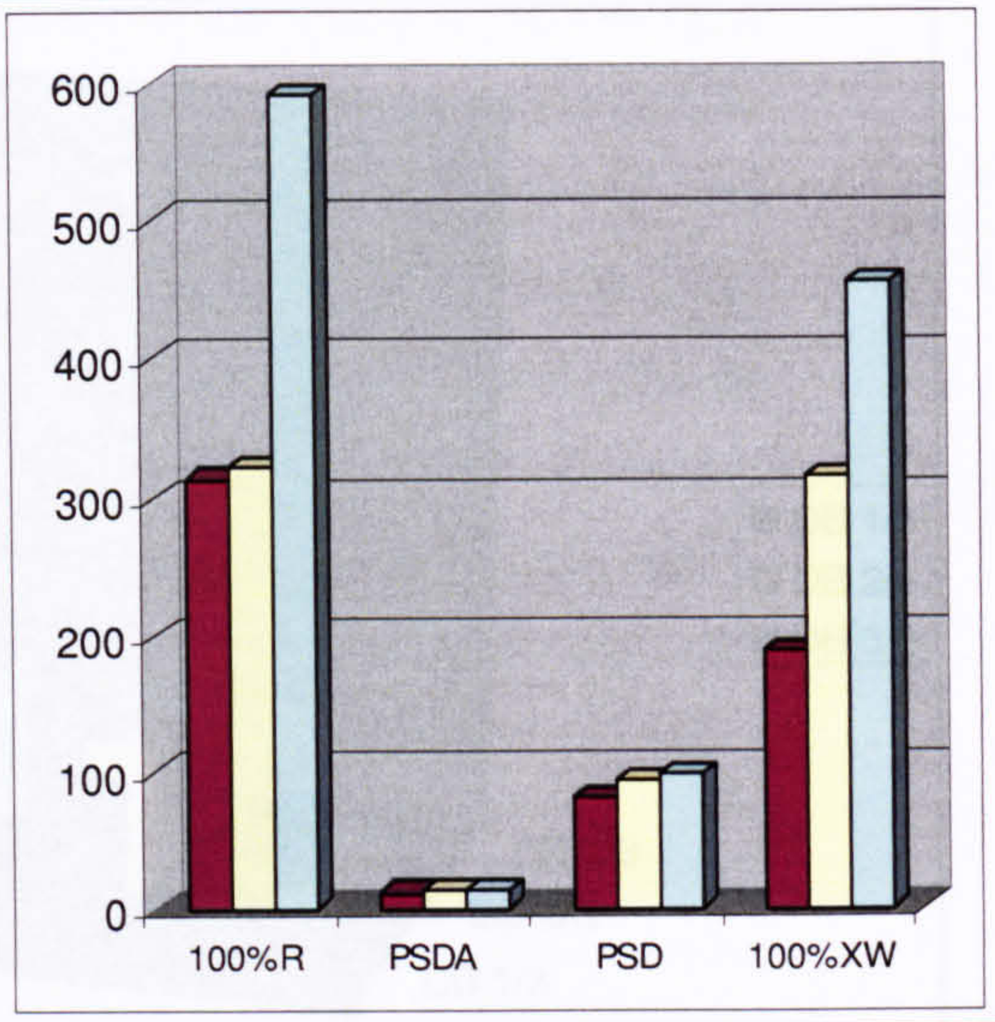
The following graphs show the results for this query



Q4 – Graph 1: Database Size vs. Storage Strategy vs. Performance



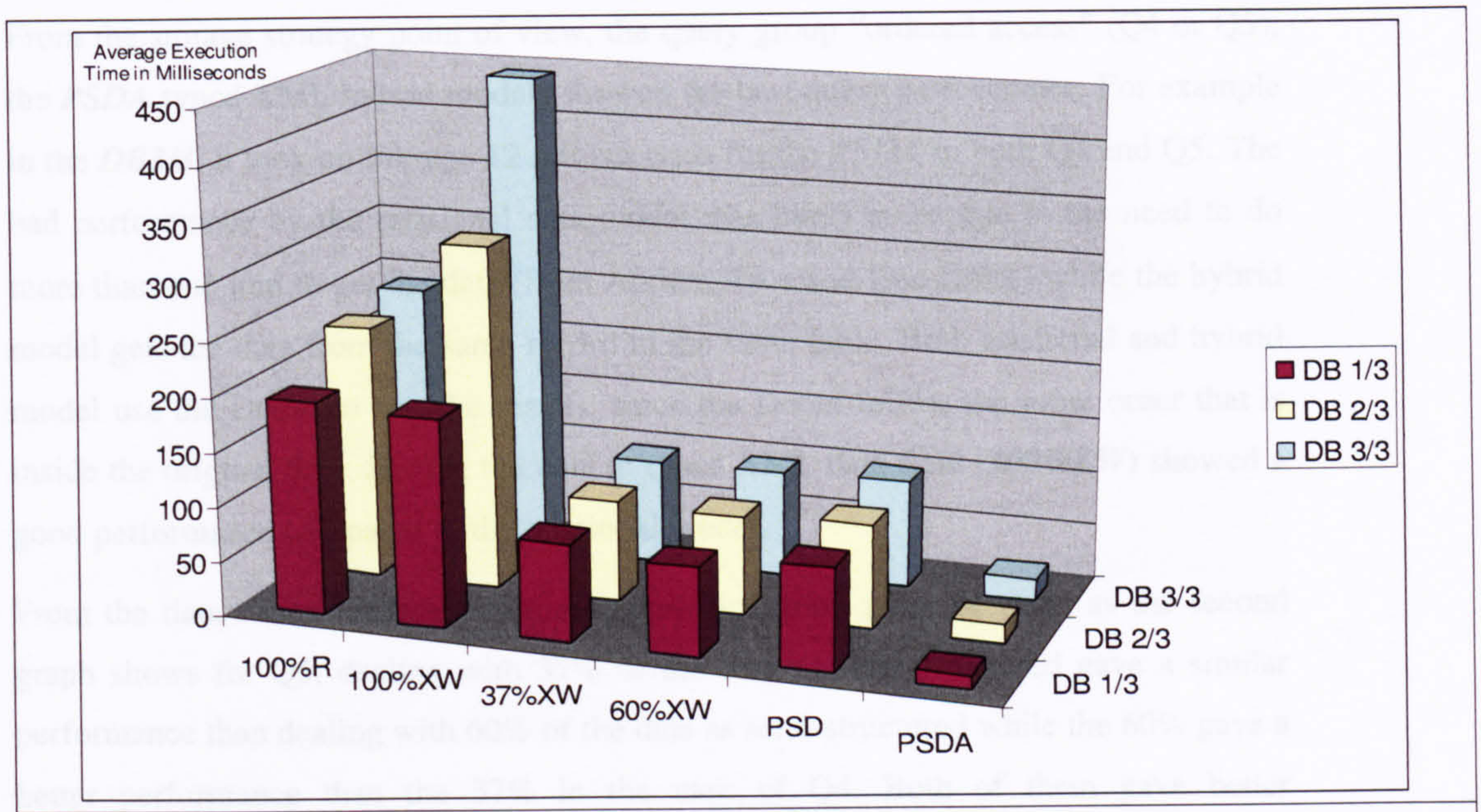
Q4 – Graph 2: Data Instances Dimension



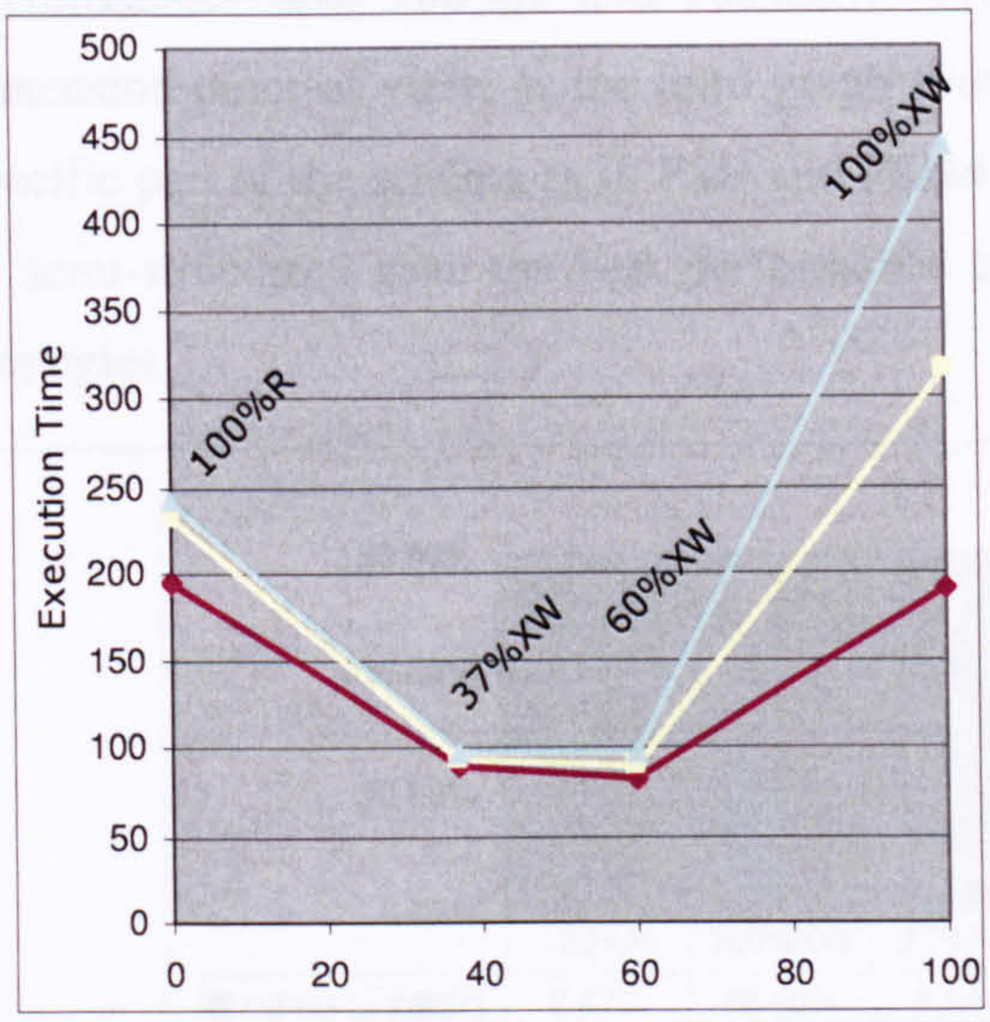
Q4 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	314	141.63	323	102.72	592	124.93
100%XW	188	57.45	316	81.70	454	97.89
37%XW	90	5.65	95	9.81	100	12.35
60%XW	83	7.53	84	7.42	86	10.77
PSD	81	22.73	92	14.20	97	7.23
PSDA	12	0.51	13	1.83	14	1.57

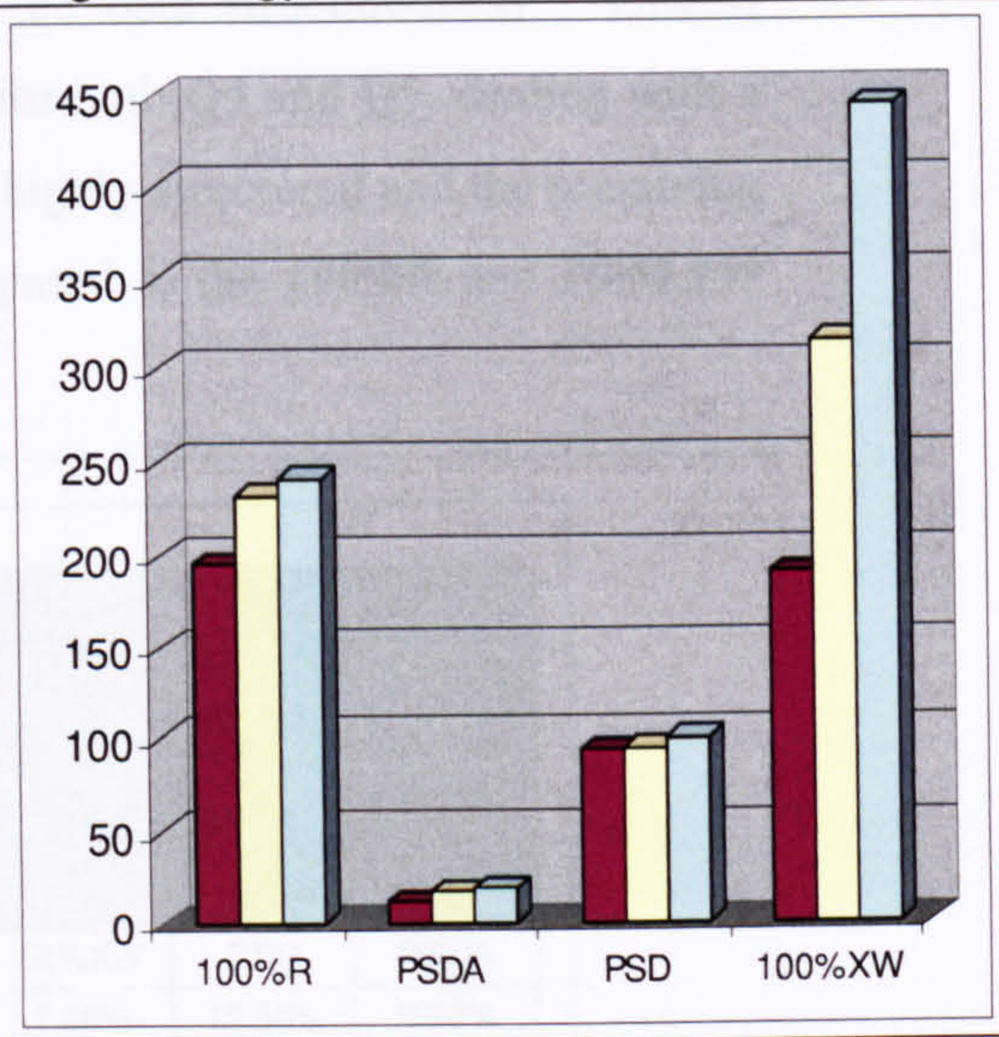
Figure 5.22 Query 4: Relative Ordered Access



Q5 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q5 – Graph 2: Data Instances Dimension



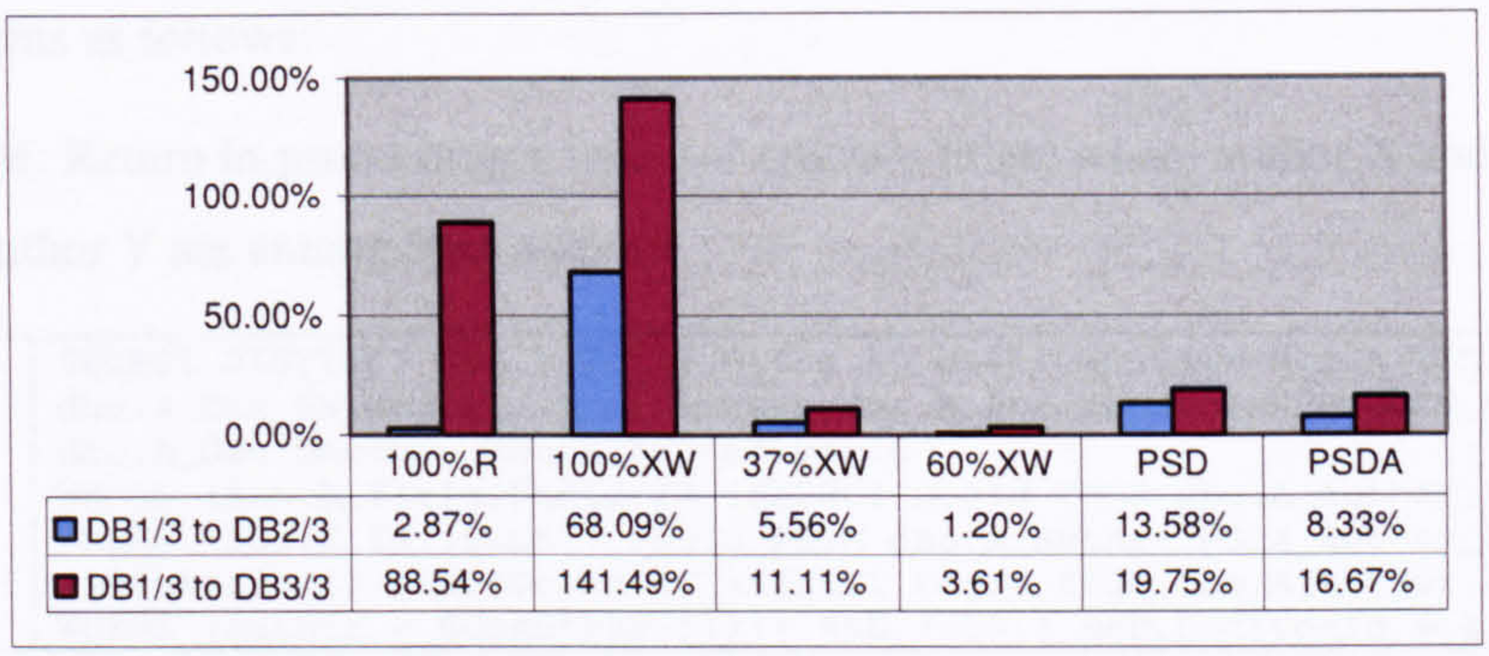
Q5 – Graph 3: Schema Dimension

	DB 1/3		DB 2/3		DB 3/3	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	195	36.85	232	34.19	241	80.36
100%XW	190	66.03	315	84.30	444	99.89
37%XW	89	6.57	92	4.32	96	8.02
60%XW	81	7.08	89	6.56	96	6.44
PSD	93	1.08	94	4.09	99	7.60
PSDA	12	1.07	17	1.04	19	1.57

Figure 5.23 Query 5: Absolute Ordered Access

From the storage strategy point of view, the query group "ordered access" (Q4 or Q5), the ***PSDA*** typed XML hybrid models showed the best query performance. For example in the ***DB3/3***, it took on average 12 milliseconds for the ***PSDA*** in both Q4 and Q5. The bad performance by the relational data model was likely to be due to the need to do more than one join to get the data (from Author, Title and Doc tables) while the hybrid model gets the data from the same record in the same table. Both relational and hybrid model use the DocId to sort the results, since the DocId follow the same order that is inside the original data. Storing the data in typed XML data field (***100%XW***) showed a good performance compared to the relational model.

From the data structuredness - data instances dimension point of view, as the second graph shows for Q5, dealing with 37% of the data as semi-structured gave a similar performance than dealing with 60% of the data as semi-structured while the 60% gave a better performance than the 37% in the case of Q4. Both of them gave better performances than ***100%R*** and ***100%XW***. From the data structuredness - schema dimension point of view, as the third graph shows for both Q4 and Q5, dealing with a specific part of the schema as in ***PSD*** and ***PSDA*** as highly-structured and the remaining as semi-structured gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD	PSDA
DB1/3 to DB2/3	2.87%	68.09%	5.56%	1.20%	13.58%	8.33%
DB1/3 to DB3/3	88.54%	141.49%	11.11%	3.61%	19.75%	16.67%

Figure 5.24 Query 4: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure illustrates the performance deterioration when the database size doubled and tripled for Q4. The worst performance was for the model ***100%XW.***

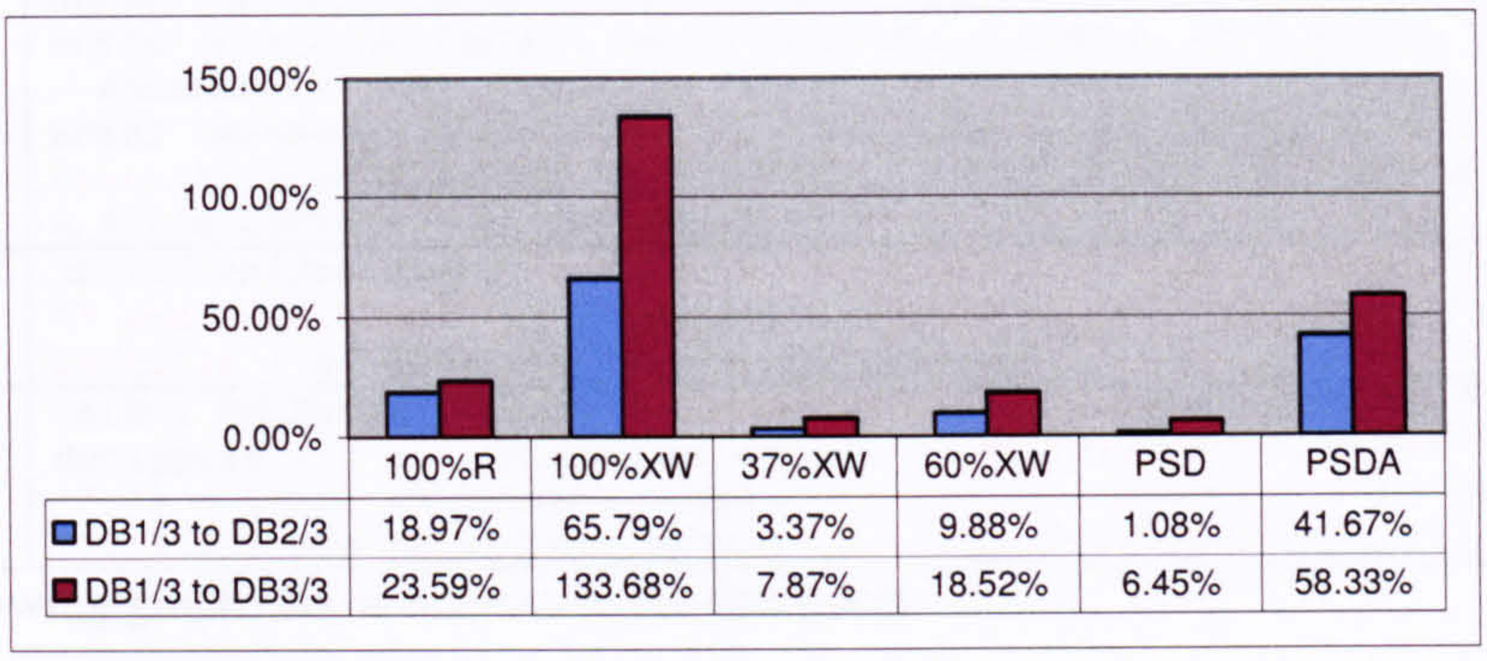


Figure 5.25 Query 5: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q5. The worst performance was for the model ***100%XW.***

5.3.2.4 Quantification

The queries in this group test the existentially (Q6) and universally (Q7) quantified queries. The queries in this group were interpreted to match the data set used in the experiments as follows:

- ***Q6***: Return in-proceeding's titles (or article's titles) where author X and author Y are among their authors

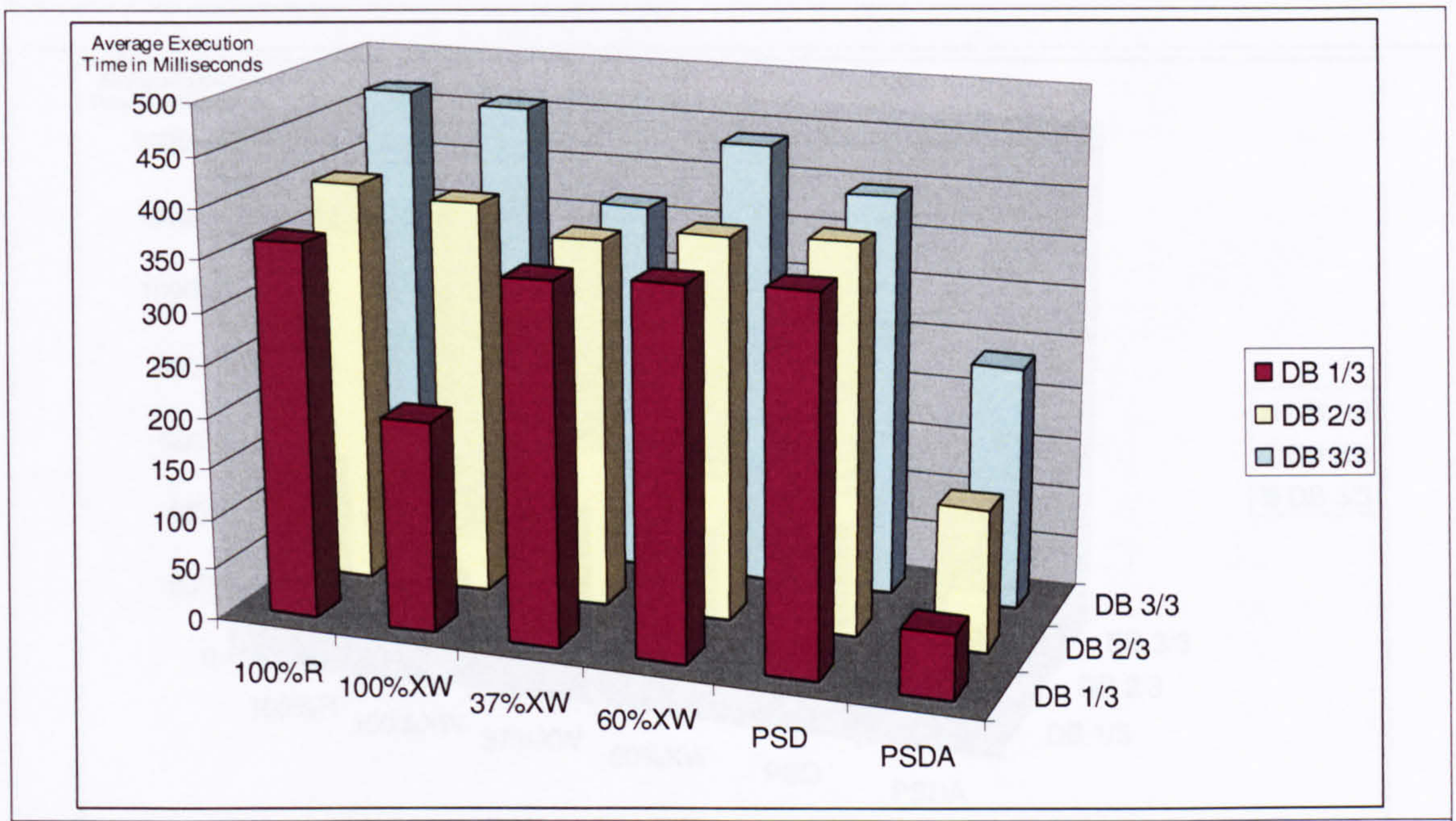
SQL Syntax	SELECT DISTINCT dbo.A_Title.Title AS Q06A100R FROM dbo.A_Title INNER JOIN dbo.A_Doc ON dbo.A_Title.DocId = dbo.A_Doc.DocId INNER JOIN dbo.A_Author ON dbo.A_Doc.DocId = dbo.A_Author.DocId WHERE (dbo.A_Title.DocId IN (SELECT DocId FROM dbo.A_Author AS A_Author_3 WHERE (DocId IN (SELECT DocId FROM dbo.A_Author AS A_Author_2 WHERE (Author = @Author))) AND (DocId IN (SELECT DocId FROM dbo.A_Author AS A_Author_1 WHERE (Author = @CoAuthor))))) AND (dbo.A_Doc.DocTypeId = 1) For XML Auto
XQuery Syntax	SELECT xmldoc.query('for $x in /dblp/article WHERE $x/author="' + @Author + '" and $x/author="' + @CoAuthor + '" return $x/title') AS Q06A100X From B_XMLDocument;
SQL/XQuery Syntax	SELECT XMLExtract.query('/inproceedings/title') AS Q06A60X From C_Doc WHERE XMLExtract.exist('/inproceedings[author="' + @Author + '"]') = 1 and XMLExtract.exist('/inproceedings[author="' + @CoAuthor + '"]') = 1 and doctypeid = 2;

- ***Q7***: Return in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables) that have the two exact authors (author X and author Y)

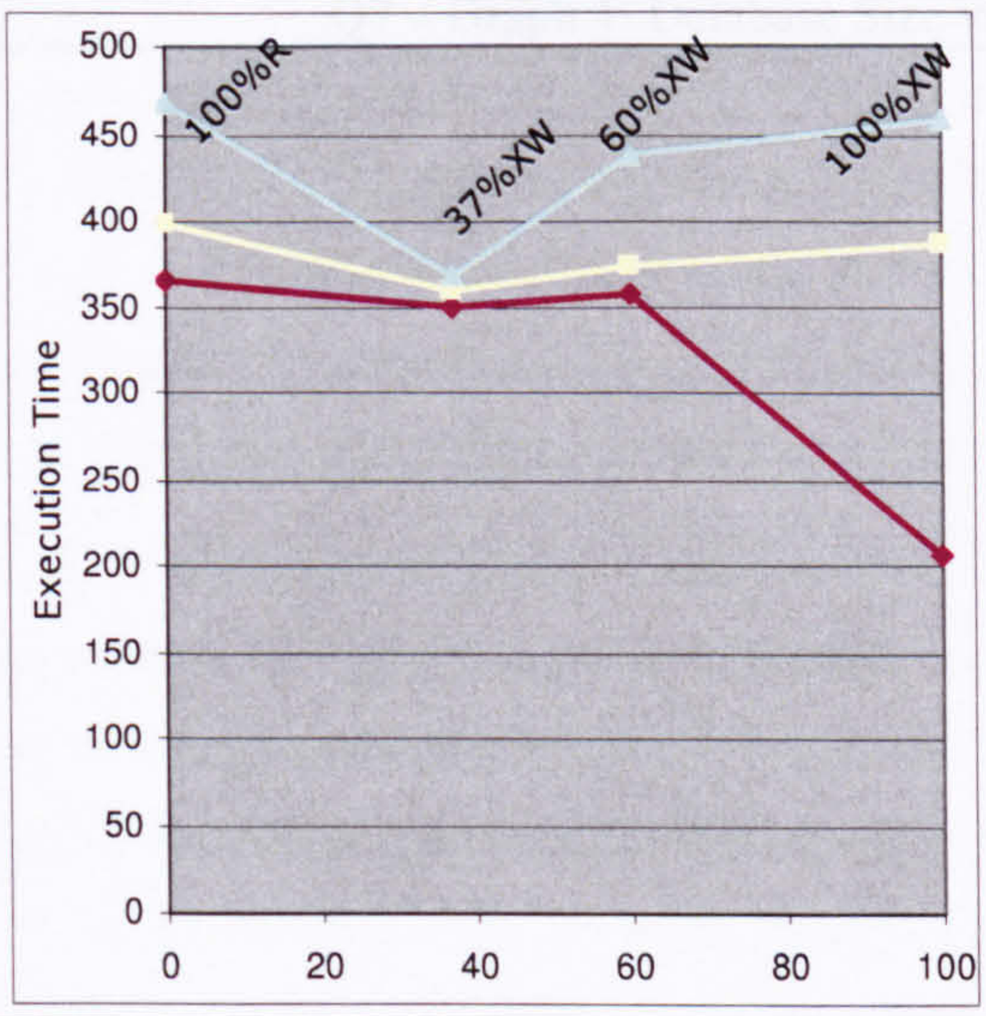
SQL Syntax	SELECT DISTINCT dbo.A_Title.Title AS Q07A100R FROM dbo.A_Title INNER JOIN dbo.A_Doc ON dbo.A_Title.DocId = dbo.A_Doc.DocId INNER JOIN dbo.A_Author ON dbo.A_Doc.DocId = dbo.A_Author.DocId

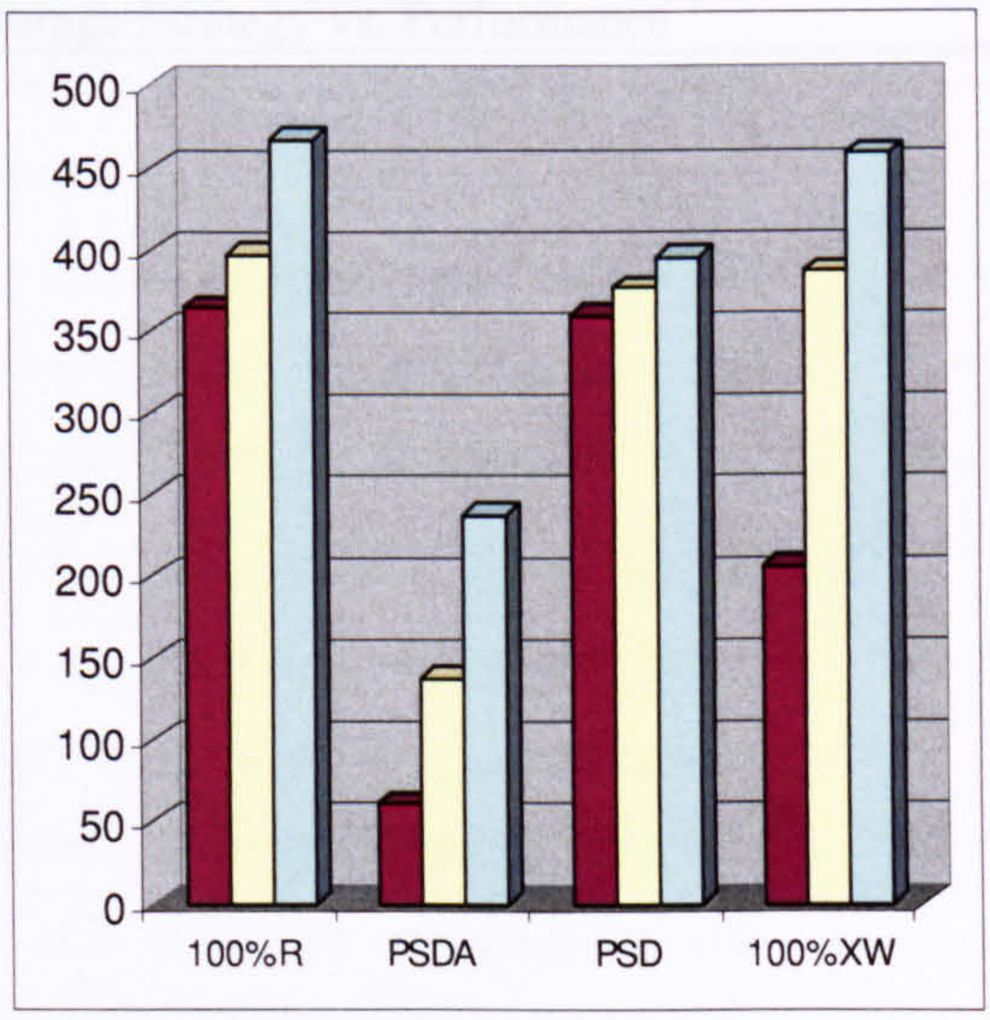
	WHERE (dbo.A_Title.DocId IN (SELECT DocId FROM dbo.A_Author AS A_Author_3 WHERE (DocId IN (SELECT DocId FROM dbo.A_Author AS A_Author_2 WHERE (Author = @Author))) AND (DocId IN (SELECT DocId FROM dbo.A_Author AS A_Author_1 WHERE (Author = @CoAuthor)))))) AND (dbo.A_Doc.DocTypeId = 1) AND ((SELECT Count(*) From A_Author AS A_Author_3 WHERE (A_Author_3.DocId = A_Title.DocId)) = 2) For XML Auto;
XQuery Syntax	SELECT xmldoc.query('for $x in /dblp/article WHERE $x/author="' + @Author + '" and $x/author="' + @CoAuthor + '" and count($x/author) = 2 return $x/title') AS Q07A100X From B_XMLDocument;
SQL/XQuery Syntax	SELECT XMLExtract.query('/inproceedings/title') AS Q07A60X From C_Doc WHERE doctypeid = 2 and XMLExtract.exist ('/inproceedings[author="' + @Author + '"]') = 1 and XMLExtract.exist('/inproceedings[author="' + @CoAuthor + '"]') = 1 and XMLExtract.value('count(/article/author)', 'int') = 2;

The following graphs show the results for this query



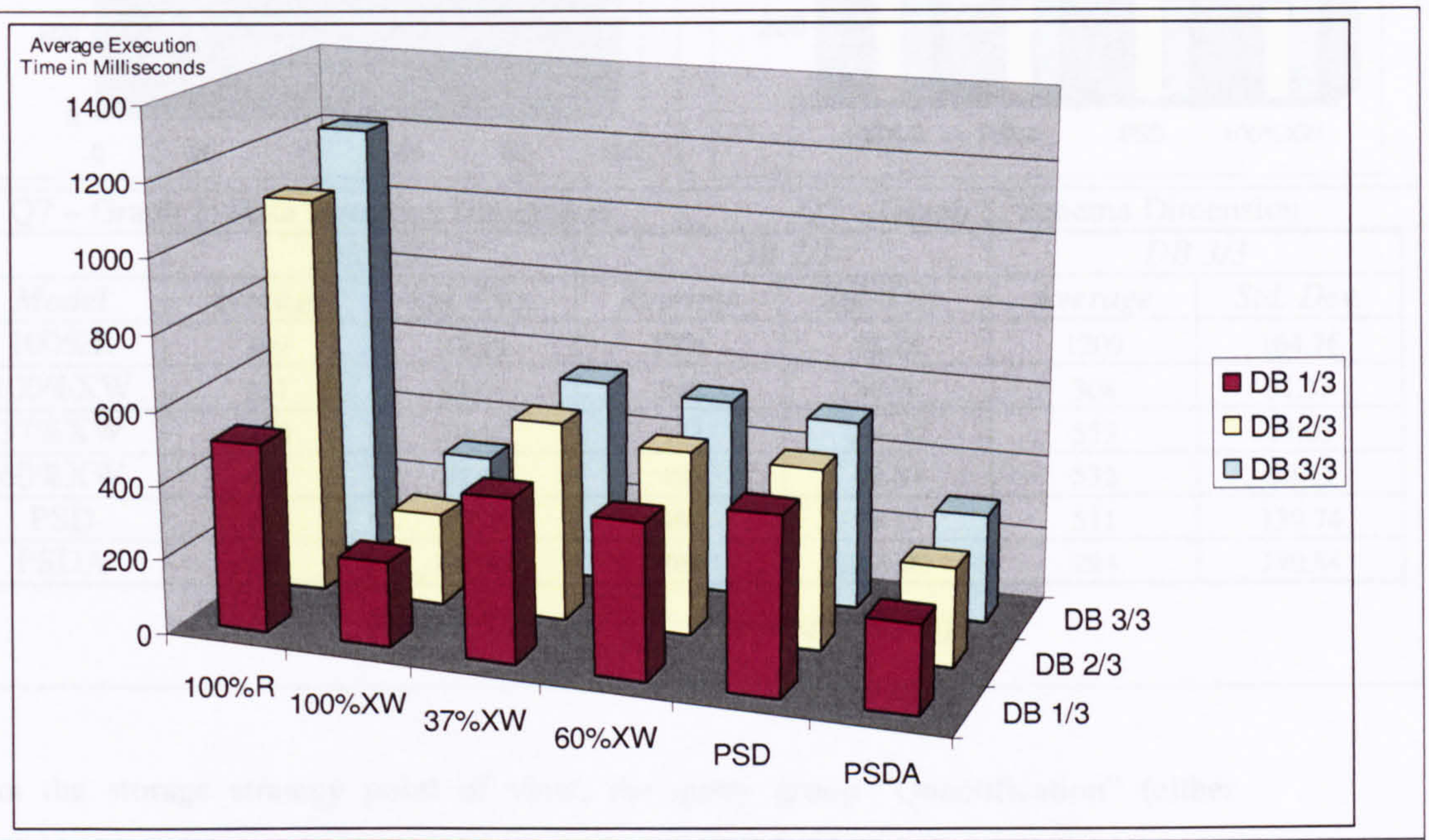
Q6 – Graph 1: Database Size vs. Storage Strategy vs. Performance



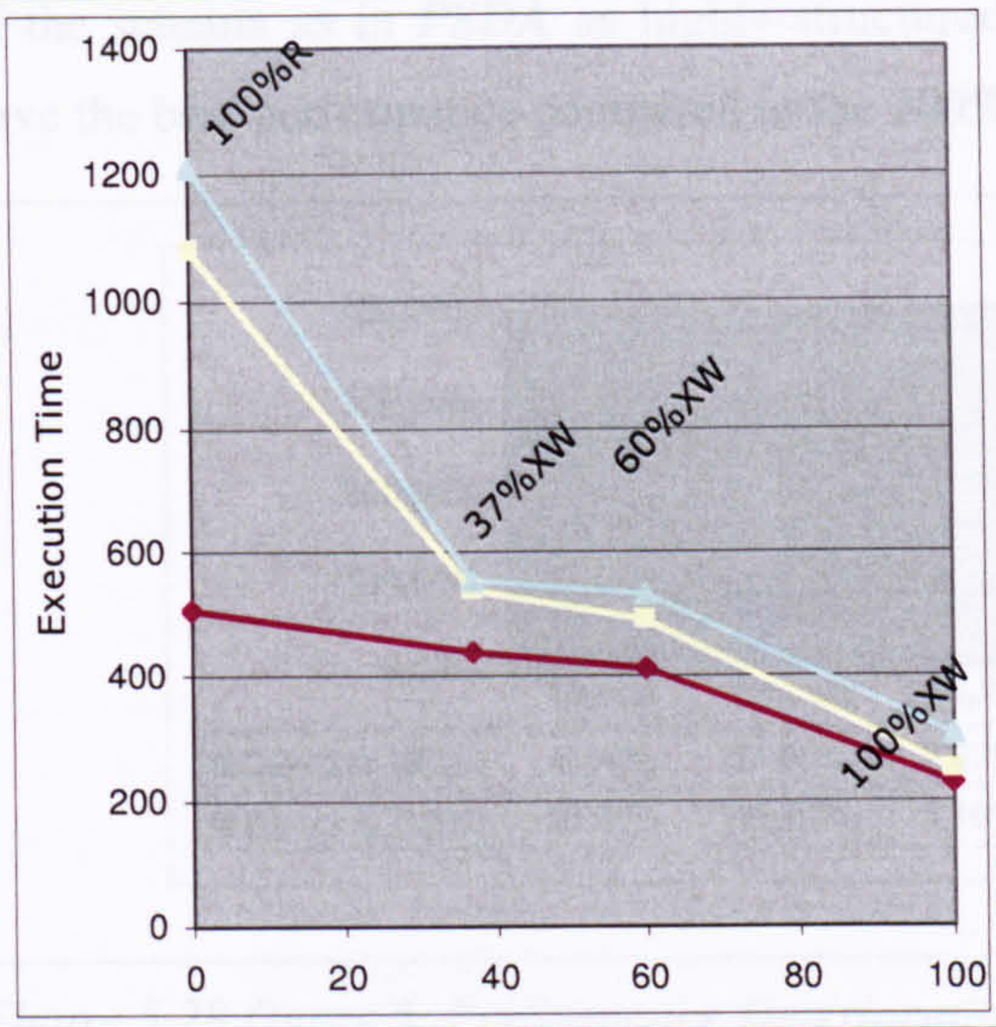


Q6 – Graph 2: Data Instances Dimension			Q6 – Graph 3: Schema Dimension			
	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	365	72.24	398	40.06	468	171.06
100%XW	206	76.64	387	72.81	459	164.03
37%XW	349	29.72	359	32.42	367	59.20
60%XW	356	66.94	373	43.64	438	31.49
PSD	360	121.43	377	68.03	396	44.31
PSDA	61	31.01	138	63.38	238	165.46

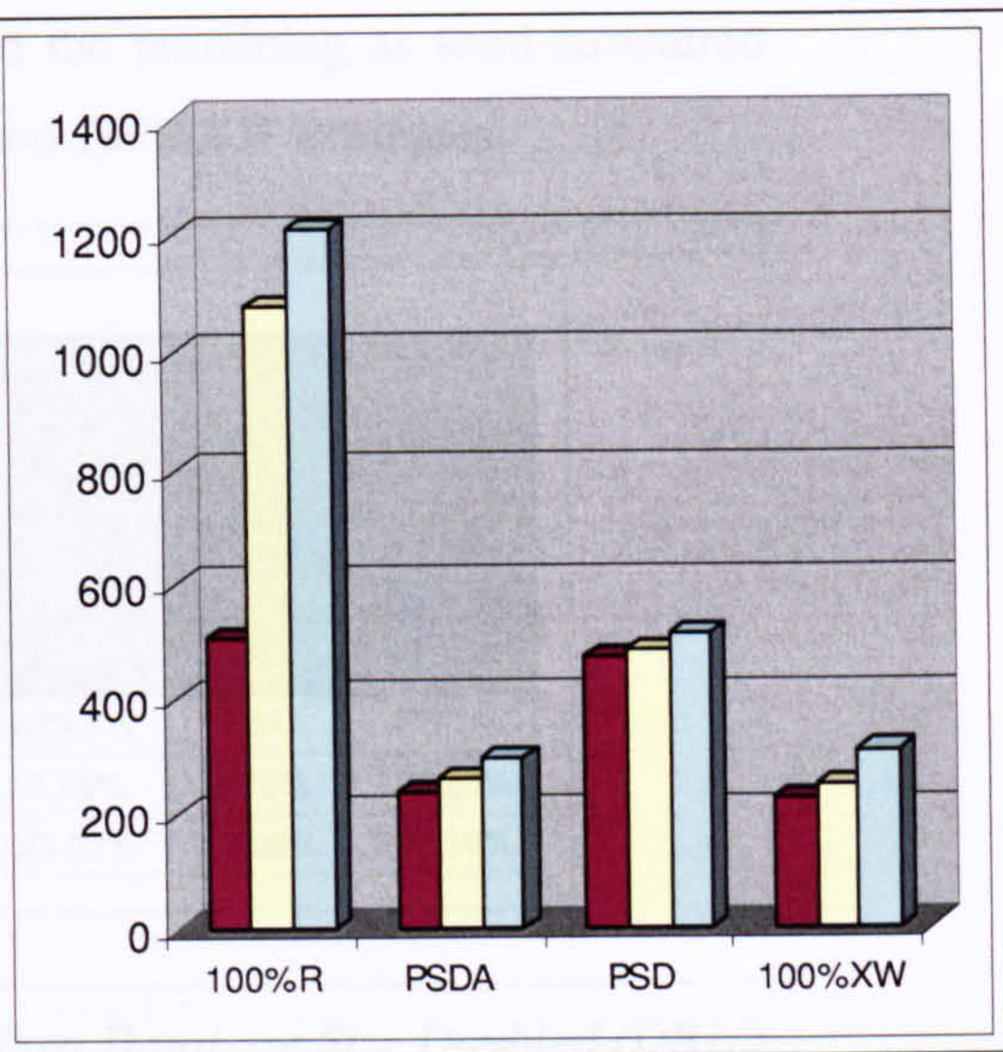
Figure 5.26 Query 6: Existential Quantifier



Q7 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q7 – Graph 2: Data Instances Dimension



Q7 – Graph 3: Schema Dimension

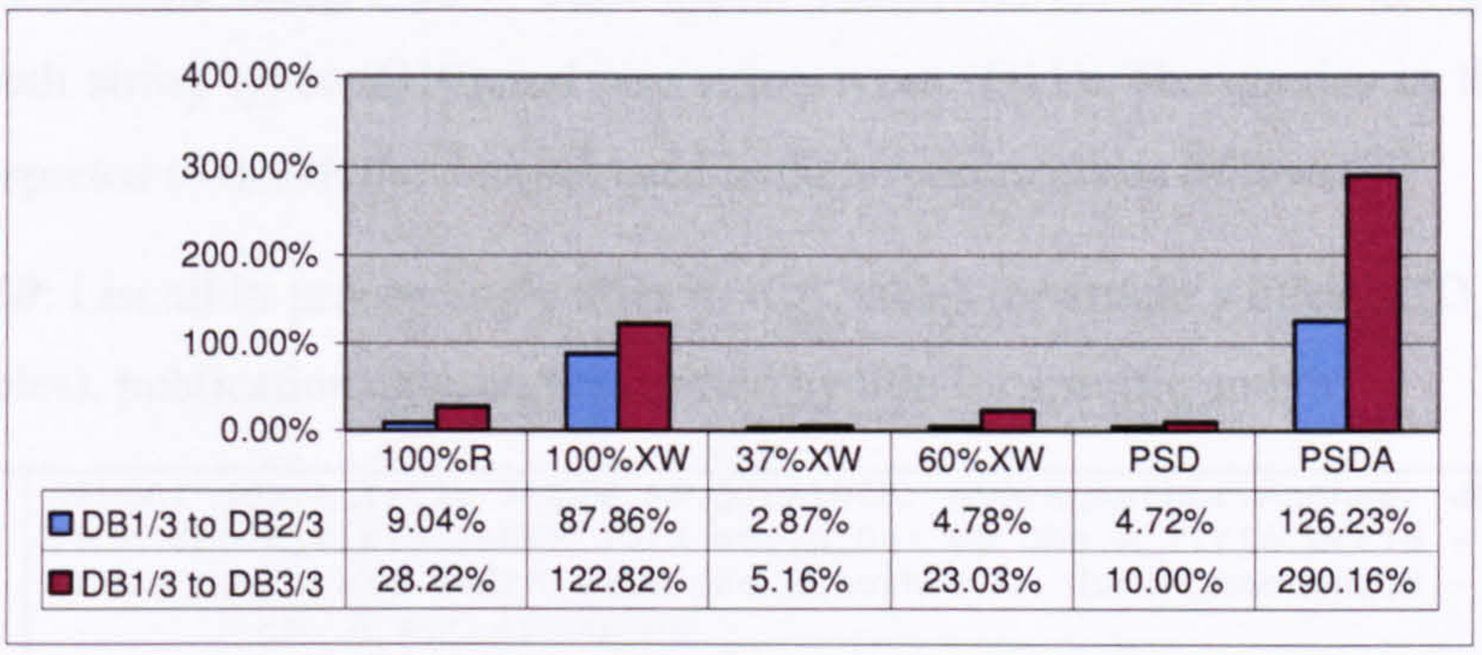
	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	509	32.59	1076	28.28	1209	164.76
100%XW	227	87.64	249	86.76	308	51.33
37%XW	440	79.62	537	211.12	553	77.03
60%XW	413	52.97	493	99.64	532	60.43
PSD	472	107.79	484	79.72	511	139.74
PSDA	236	147.69	261	129.82	294	190.84

Figure 5.27 Query 7: Universal Quantifier

From the storage strategy point of view, the query group "Quantification" (either existentially (Q6) or universally (Q7)), the typed ***PSDA*** XML hybrid models showed the best query performance. For example in the ***DB3/3***, it took on average 294 for Q6 and 238 for Q7. The good performance of the ***100%XW*** (especially in Q6) was likely to be due to the 'where' statement inside the XQuery which checked both Author and CoAuthor in the same condition inside the XQuery while in the proposed system it had to be checked in two different conditions as the SQL and XQuery syntax showed.

From the data structuredness - data instances dimension point of view, as the second graph shows for Q6, dealing with 37% of the data as semi-structured gave slightly better performance than dealing with 60% of the data as semi-structured. For Q7, dealing with 60% of the data as semi-structured gave a slightly better performance than dealing with 37% of the data as semi-structured. From the data structuredness - schema dimension point of view, as the third graph shows for both Q6 and Q7, dealing with a specific part

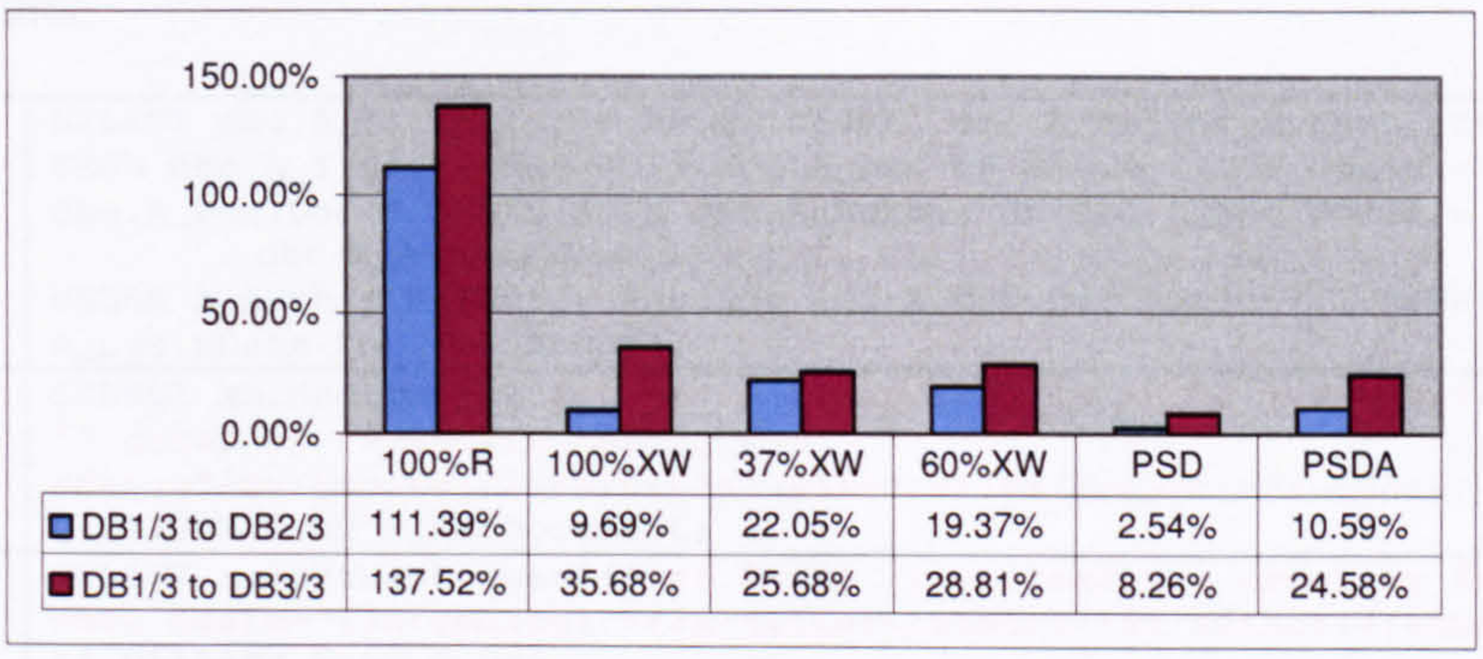
of the schema as in ***PSDA*** as highly-structured and the remaining as semi-structured gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD	PSDA
DB1/3 to DB2/3	9.04%	87.86%	2.87%	4.78%	4.72%	126.23%
DB1/3 to DB3/3	28.22%	122.82%	5.16%	23.03%	10.00%	290.16%

Figure 5.28 Query 6: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure represents the performance deterioration when the database size doubled and tripled for Q6. The worst performance was for the hybrid model ***PSDA.***



	100%R	100%XW	37%XW	60%XW	PSD	PSDA
DB1/3 to DB2/3	111.39%	9.69%	22.05%	19.37%	2.54%	10.59%
DB1/3 to DB3/3	137.52%	35.68%	25.68%	28.81%	8.26%	24.58%

Figure 5.29 Query 7: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q7. The worst performance was for the model ***100%R.***

5.3.2.5 Sorting

Even though the generic data type of element content in XML documents was string, users may cast the string type to other types. Therefore, the queries in this group test sorting both string types (Q10) and non-string types (Q11). The queries in this group were interpreted to match the data set used in the experiments as follows:

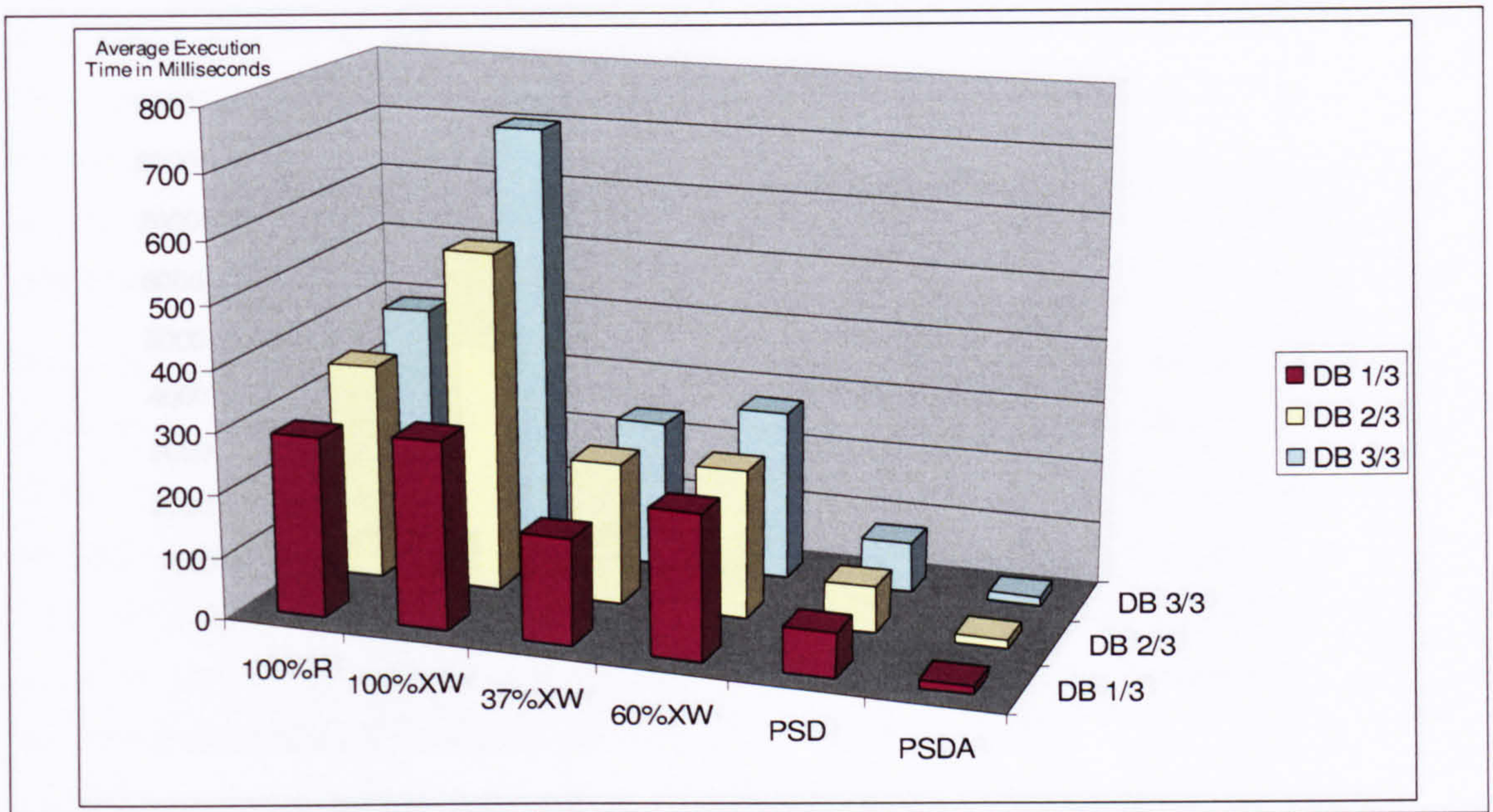
- ***Q10***: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors sorted by title for specific author.

SQL Syntax	SELECT dbo.A_Title.Title AS Q10A100R, dbo.A_Author.Author, dbo.A_Doc.MDate FROM dbo.A_Title INNER JOIN dbo.A_Doc ON dbo.A_Title.DocId = dbo.A_Doc.DocId INNER JOIN dbo.A_Author ON dbo.A_Doc.DocId = dbo.A_Author.DocId WHERE A_Author.Author = @Author and A_Doc.DocTypeId = 1 ORDER BY A_Title.Title For XML Auto;
XQuery Syntax	SELECT xmldoc.query('for $x in /dblp/article WHERE $x/author="' + @Author + '" ORDER BY ($x/title/text())[1] return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q06A100X From B_XMLDocument;
SQL/XQuery Syntax	SELECT xmlextract.query('for $x in /inproceedings ORDER BY ($x/title/text())[1] return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q10A60X From C_Doc WHERE DocTypeId = 2 and XMLExtract.exist('/inproceedings[author="' + @Author + '"]') = 1';

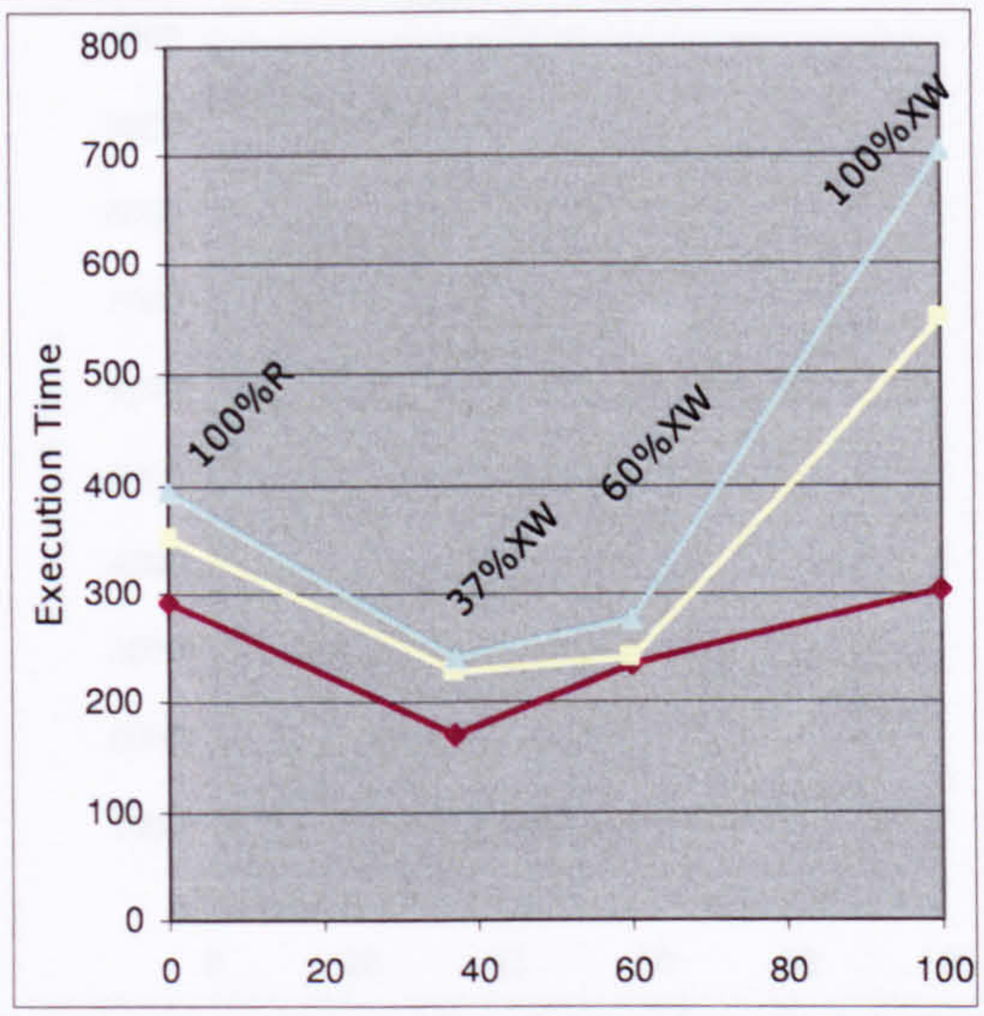
- ***Q11***: List all in-proceeding's titles in 'C_' tables (or article's titles in 'D_' tables), publication date, authors sorted by publication date for specific author.

SQL Syntax	SELECT dbo.A_Title.Title AS Q11A100R, dbo.A_Author.Author, dbo.A_Doc.MDate FROM dbo.A_Title INNER JOIN dbo.A_Doc ON dbo.A_Title.DocId = dbo.A_Doc.DocId INNER JOIN dbo.A_Author ON dbo.A_Doc.DocId = dbo.A_Author.DocId WHERE A_Author.Author = @Author and A_Doc.DocTypeId = 1 Order By A_Doc.MDate For XML Auto;
XQuery Syntax	SELECT xmldoc.query('for $x in /dblp/article WHERE $x/author="' + @Author + '" order by $x/@mdate return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q11A100X From B_XMLDocument;
SQL/XQuery Syntax	SELECT xmlextract.query('for $x in /inproceedings order by $x/@mdate return <doc mdate="{$x/@mdate}">{$x/title}<authors>{$x/author}</authors></doc>') AS Q11A60X From C_Doc WHERE DocTypeId = 2 and XMLExtract.exist('/inproceedings[author="' + @Author + '"]') = 1;

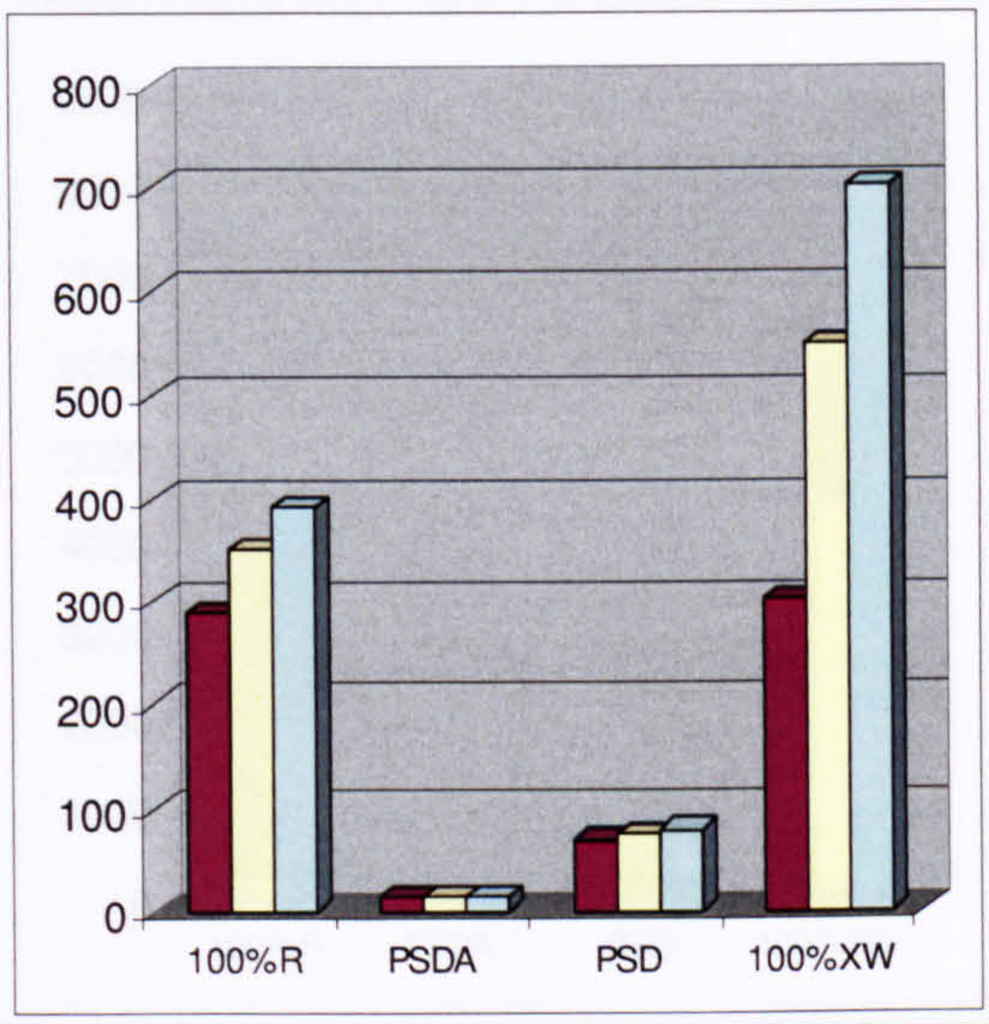
The following graphs show the results for this query



Q10 – Graph 1: Database Size vs. Storage Strategy vs. Performance



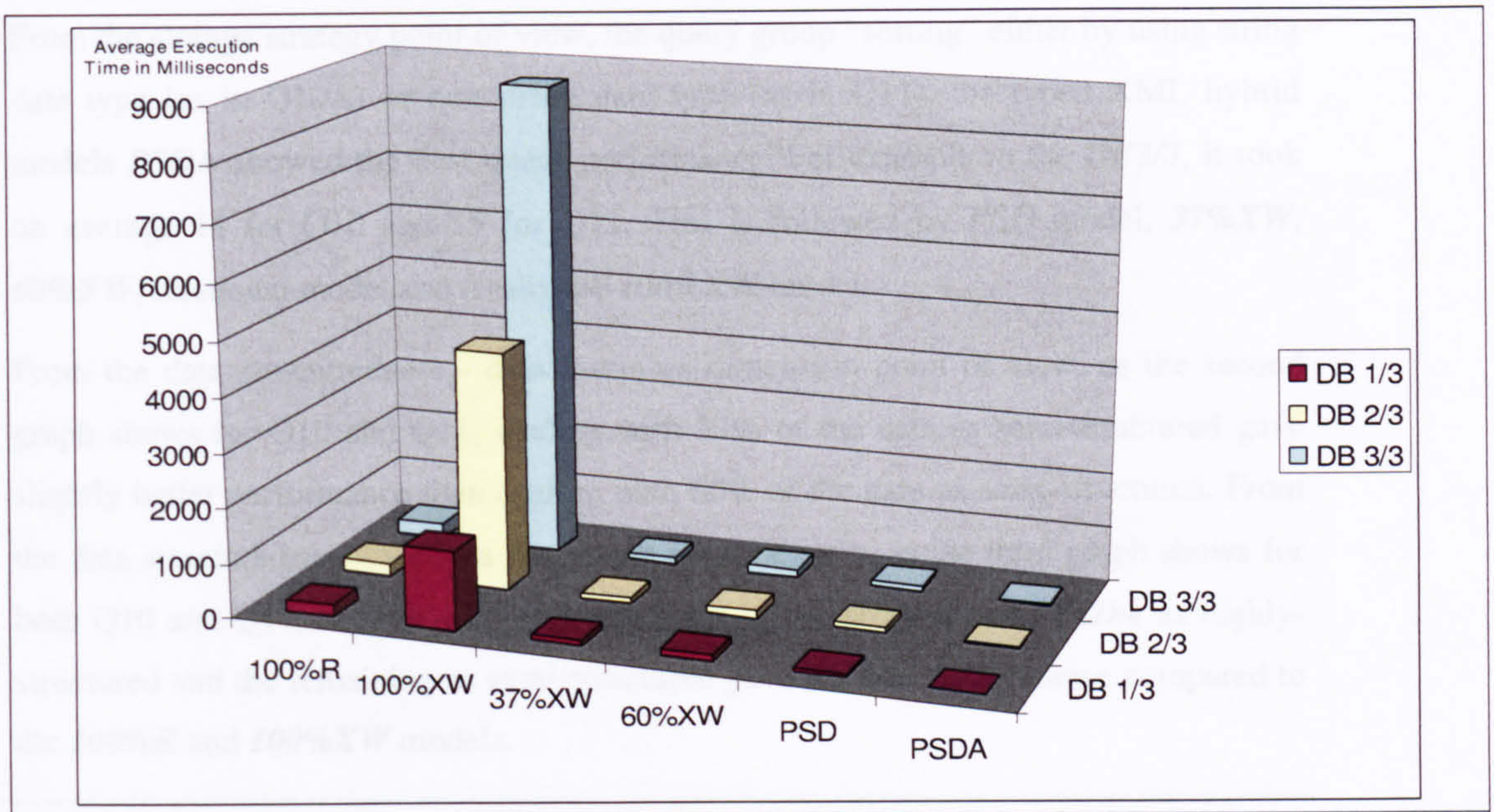
Q10 – Graph 2: Data Instances Dimension



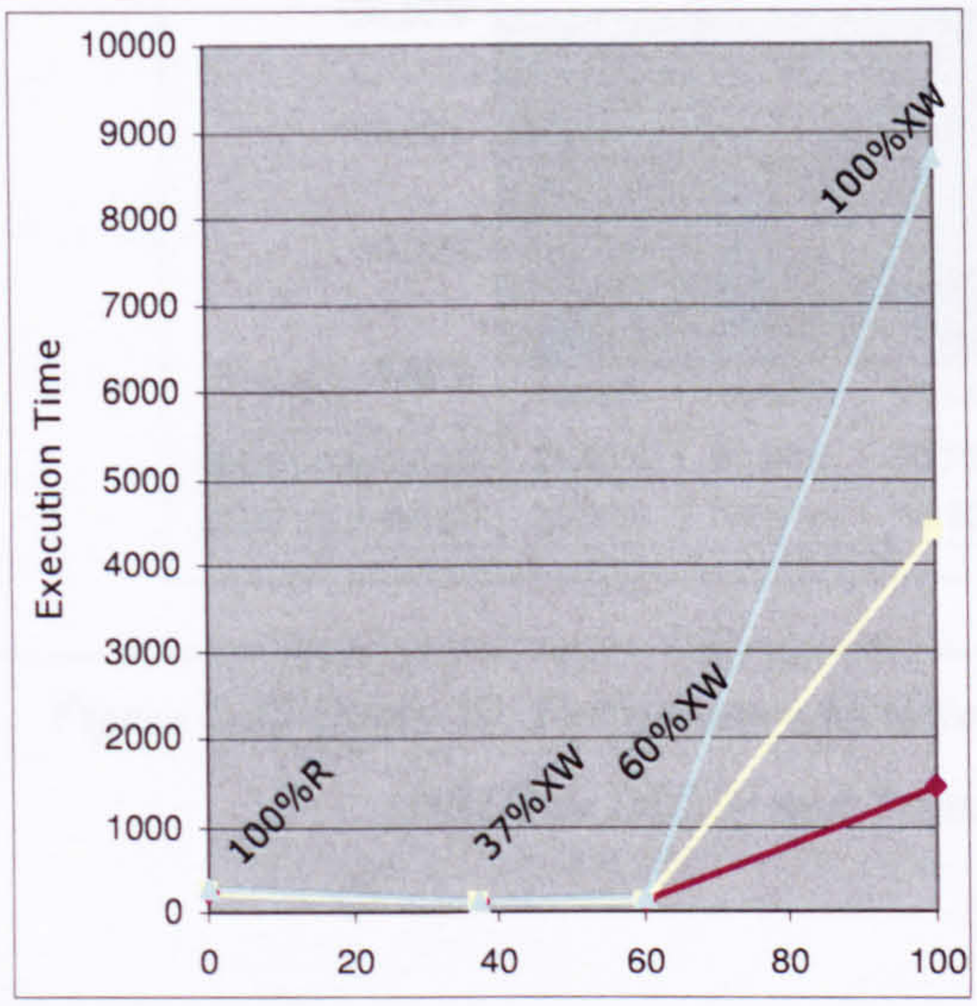
Q10 – Graph 3: Schema Dimension

	DB 1/3		DB 2/3		DB 3/3	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	290	134.70	351	12.02	393	21.66
100%XW	303	16.69	548	55.64	703	66.32
37%XW	170	18.44	227	49.83	238	76.46
60%XW	231	85.10	238	31.56	276	78.46
PSD	71	4.47	75	5.57	80	11.73
PSDA	14	1.35	14	1.51	14	1.58

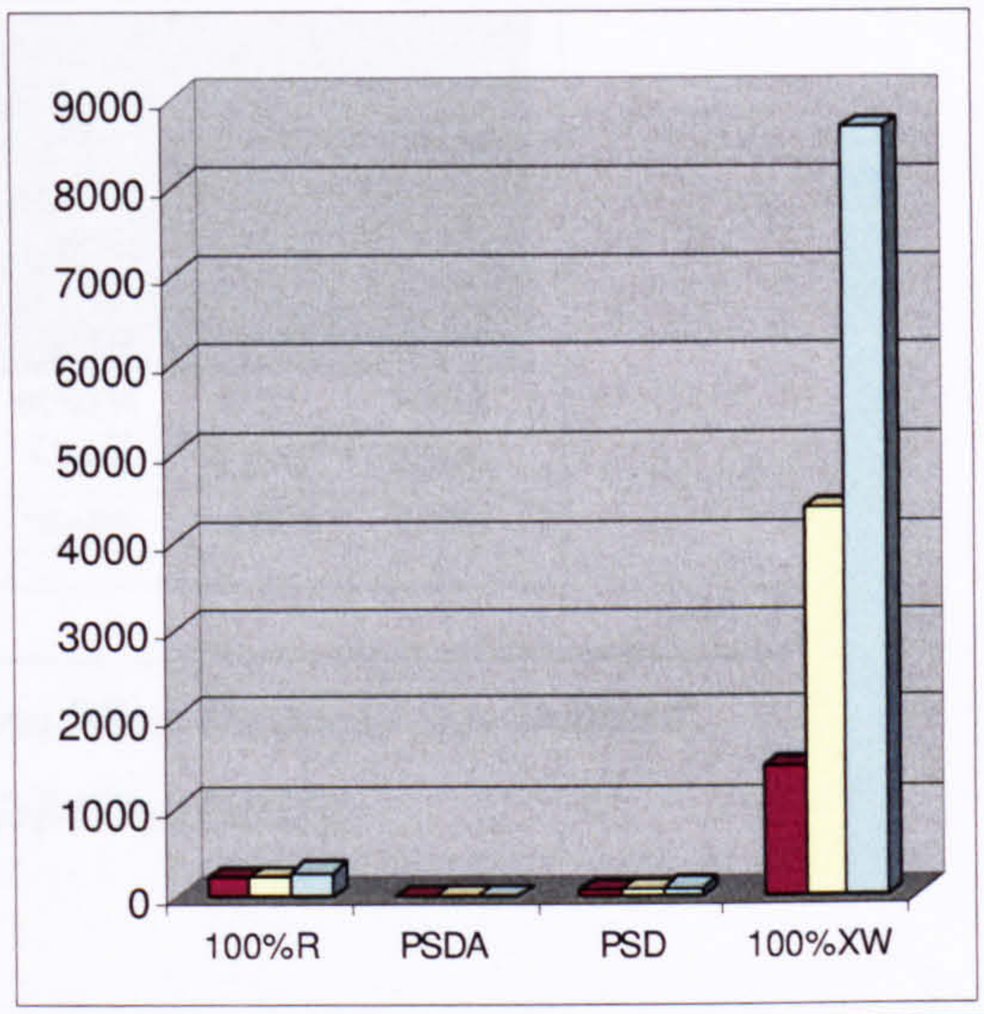
Figure 5.30 Query 10: String Sorting



Q11 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q11 – Graph 2: Data Instances Dimension



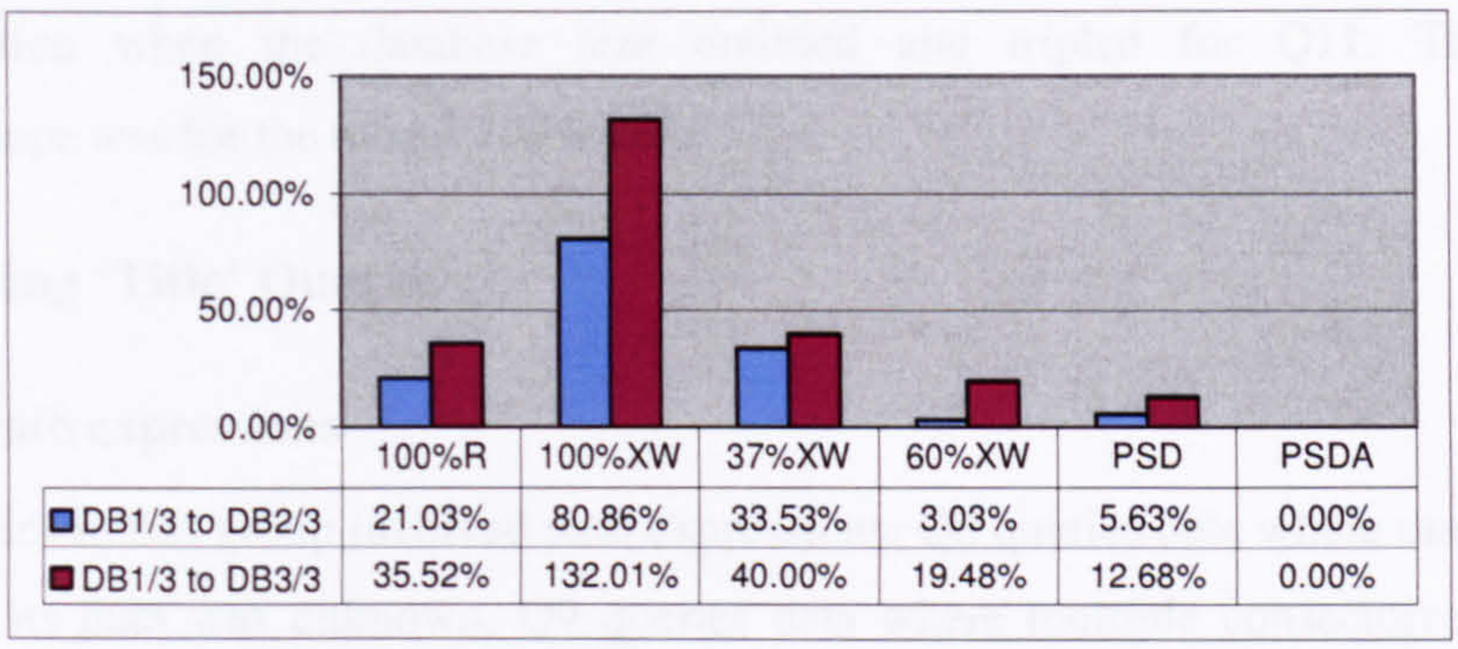
Q11 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	213	53.84	234	12.02	280	34.41
100%XW	1438	675.21	4384	1535.07	8672	1976.47
37%XW	96	20.16	102	25.59	111	6.46
60%XW	121	7.63	125	23.16	139	13.28
PSD	75	6.08	86	8.04	90	7.81
PSDA	12	1.63	13	1.17	15	1.06

Figure 5.31 Query 11: Non-string Sorting

From the storage strategy point of view, the query group "sorting" either by using string data type (as in Q10A) or non-string data type (as in Q11), the typed XML hybrid models ***PSDA*** showed the best query performance. For example in the ***DB3/3***, it took on average 14 for Q10 and 15 for Q11. This is followed by ***PSD*** model, ***37%XW***, ***60%XW***, relational model and finally the ***100%XW*** model.

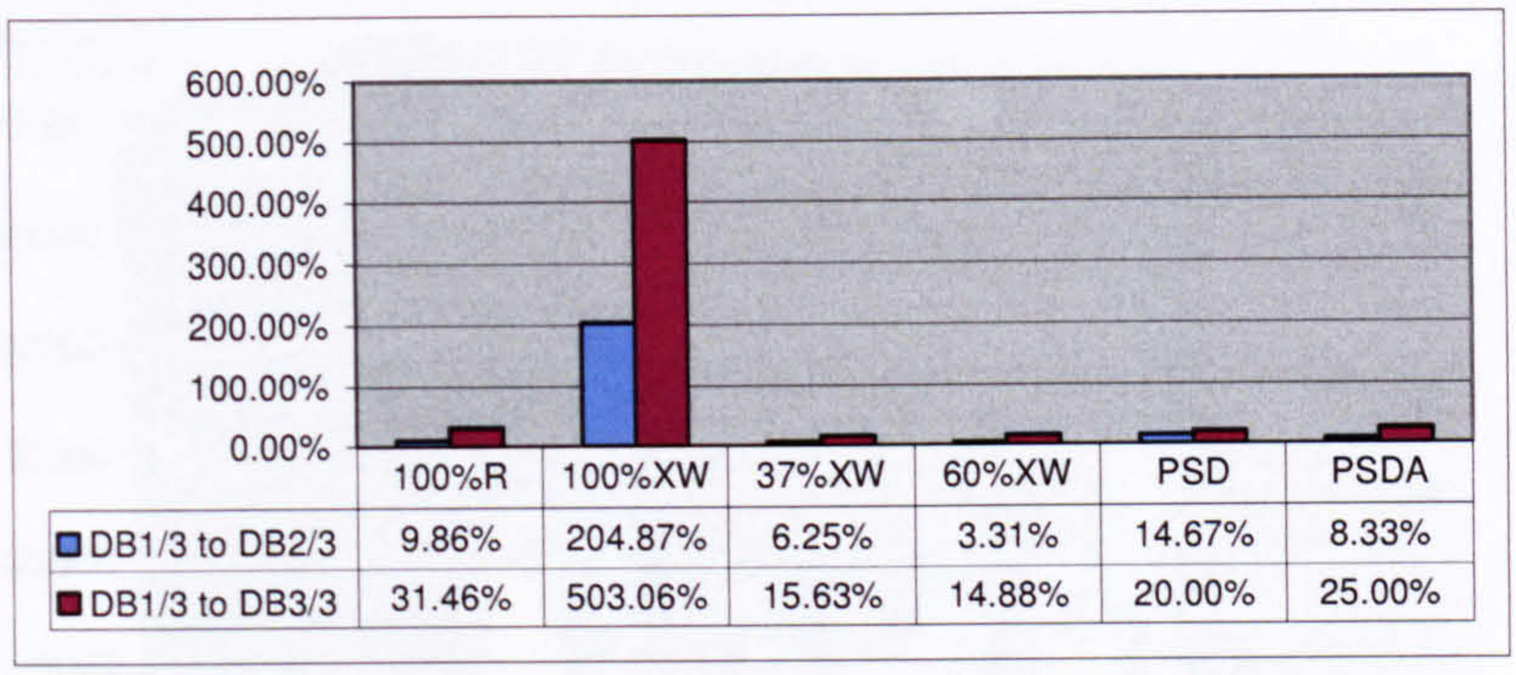
From the data structuredness - data instances dimension point of view, as the second graph shows for Q10 and Q11, dealing with 37% of the data as semi-structured gave slightly better performance than dealing with 60% of the data as semi-structured. From the data structuredness - schema dimension point of view, as the third graph shows for both Q10 and Q11, dealing with a specific part of the schema as in ***PSDA*** as highly-structured and the remaining as semi-structured gave the best performance compared to the ***100%R*** and ***100%XW*** models.



	100%R	100%XW	37%XW	60%XW	PSD	PSDA
DB1/3 to DB2/3	21.03%	80.86%	33.53%	3.03%	5.63%	0.00%
DB1/3 to DB3/3	35.52%	132.01%	40.00%	19.48%	12.68%	0.00%

Figure 5.32 Query 10: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q10. The worst performance was for the model ***100%XW.***



	100%R	100%XW	37%XW	60%XW	PSD	PSDA
DB1/3 to DB2/3	9.86%	204.87%	6.25%	3.31%	14.67%	8.33%
DB1/3 to DB3/3	31.46%	503.06%	15.63%	14.88%	20.00%	25.00%

Figure 5.33 Query 11: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q11. The worst performance was for the model ***100%XW***.

5.3.3 Using 'Title' Queries

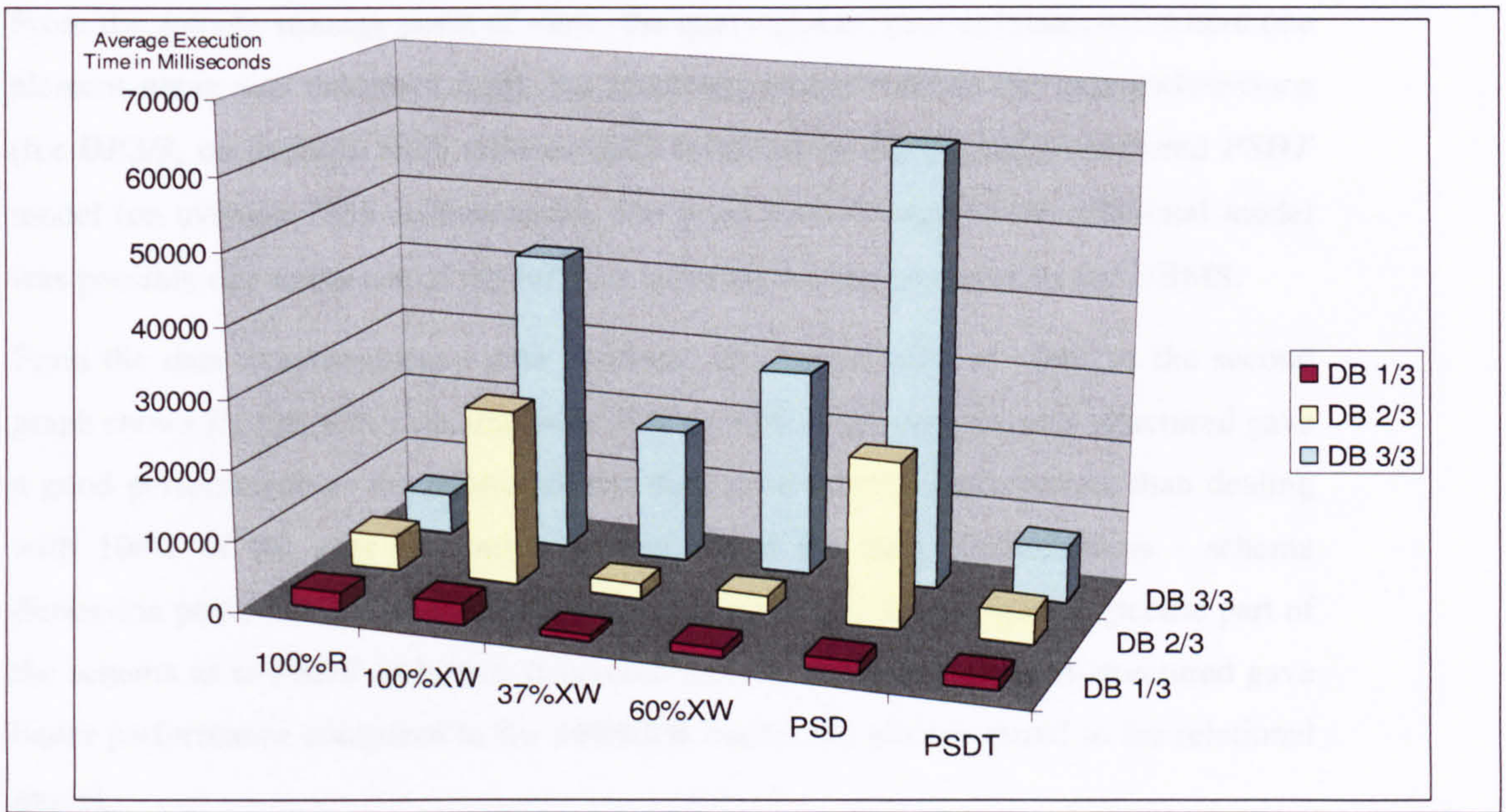
5.3.3.1 Path expressions

The queries in this group involved path expressions: Q8 queries data where one element name in its path was unknown, Q9 queries data where multiple consecutive element names in its path were unknown. The queries in this group were interpreted to match the data set used in the experiments as follows:

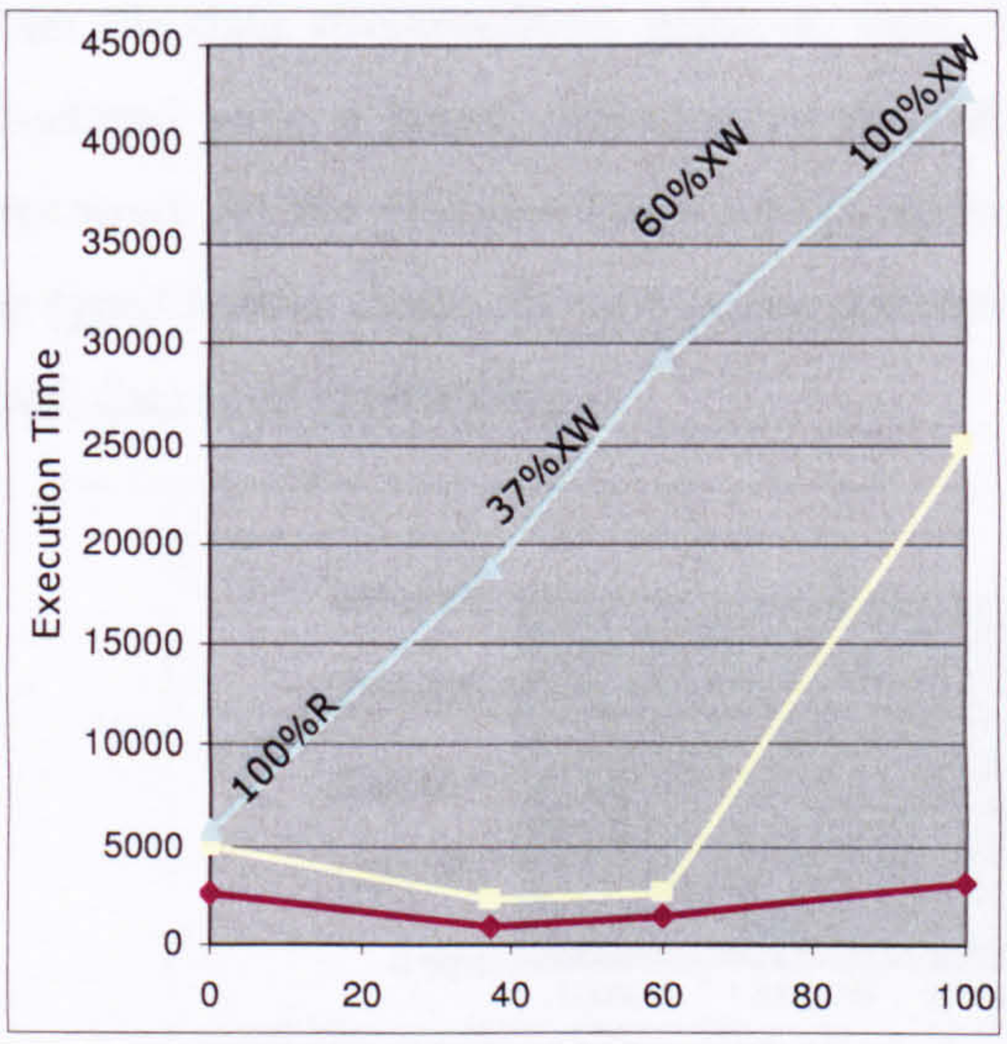
- *Q8*: Return in-proceeding's titles (or article's titles) that contain word XYZ within their title.

SQL Syntax	`SELECT Title AS Q08A100R from A_Title WHERE Title like '%' + @SearchWord + '%' And DocId in (SELECT DocId from A_Doc WHERE DocTypeID = 1) For XML Auto`
XQuery Syntax	`SELECT XMLDoc.query ('(dblp/article/title/text())[contains(.,"' + @SearchWord + '")]') AS Q08A100X from B_XMLDocument`
SQL/XQuery Syntax	`SELECT XMLExtract.query('/inproceedings/title') AS Q08A60XW From C_Doc WHERE DocTypeId = 2 And (XMLExtract.exist('(/inproceedings/title/text())[contains(.,"' + @SearchWord + '")]') = 1 );`

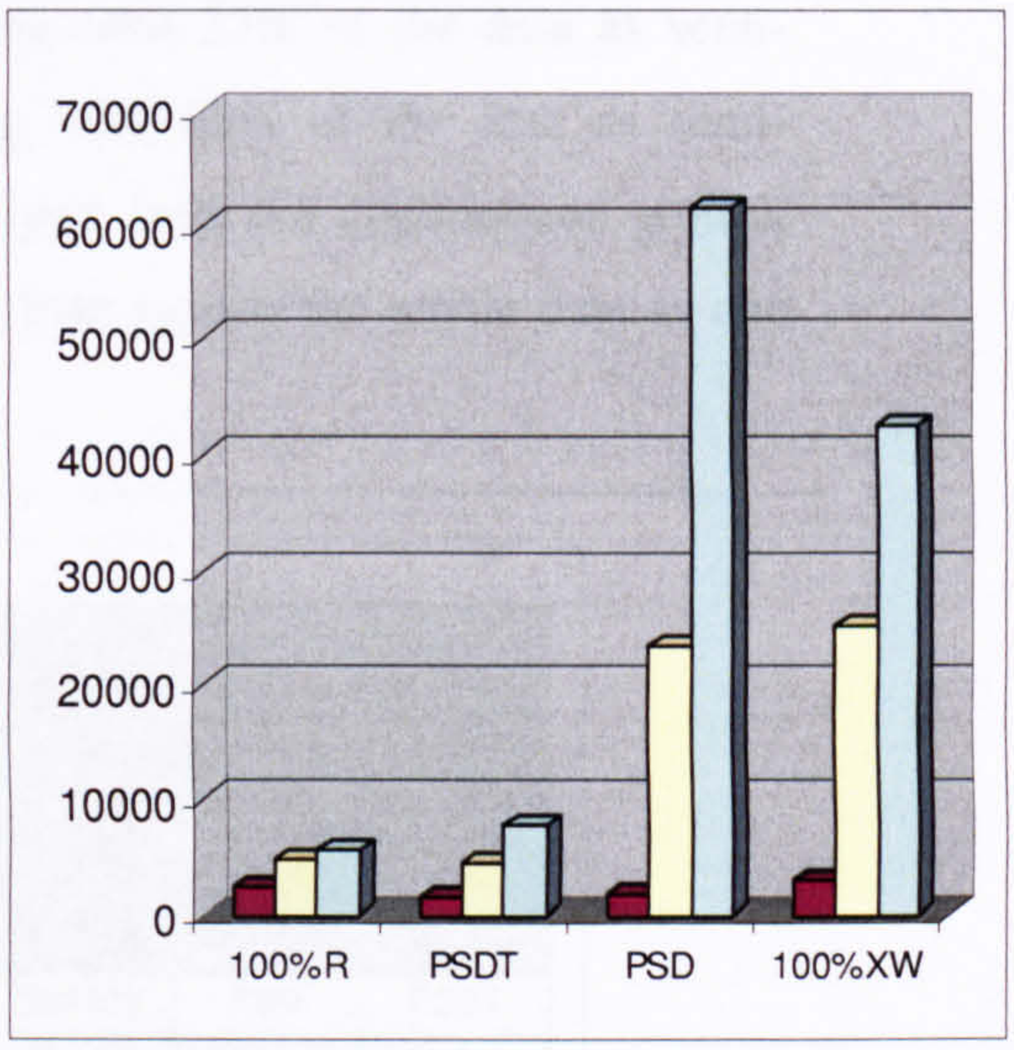
The following graphs show the results for this query



Q8 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q8 – Graph 2: Data Instances Dimension



Q8 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	2557	347.21	4866	146.37	5855	1341.39
100%XW	3121	959.70	24962	1803.61	42579	1185.69
37%XW	816	83.08	2174	1075.81	18810	5005.49
60%XW	1316	285.40	2595	642.92	29199	7401.66
PSD	1822	634.72	23193	3497.09	61481	9052.65
PSDT	1734	837.26	4509	1354.14	7855	3082.74

Figure 5.34 Query 8: Regular Path Expressions - Unknown Element

From the storage strategy point of view, the query group "path expressions" where one element name was unknown (Q8), the relational model showed the best performance (for ***DB3/3***, on average 5855 milliseconds) followed by the partially-structured ***PSDT*** model (on average 7855 milliseconds). The good performance by the relational model was possibly due to the use of the full text indexing feature provided by the DBMS.

From the data structuredness - data instances dimension point of view, as the second graph shows for Q8, either dealing with 37% or 60% of the data as semi-structured gave a good performance as the relational, but they gave a better performance than dealing with 100% of the data as semi-structured. From the data structuredness - schema dimension point of view, as the third graph shows for Q8, dealing with a specific part of the schema as in ***PSDT*** as highly-structured and the remaining as semi-structured gave better performance compared to the ***100%XW*** model but not compared to the relational model.

From the data structuredness point of view, dealing with 37% of the data as semi-structured gave a better performance than dealing with 60% of the data as semi-structured. As the previous query groups showed (apart from the qualification group), the typed hybrid model showed better performance than storing the whole data as one XML data field (***100%XW***).

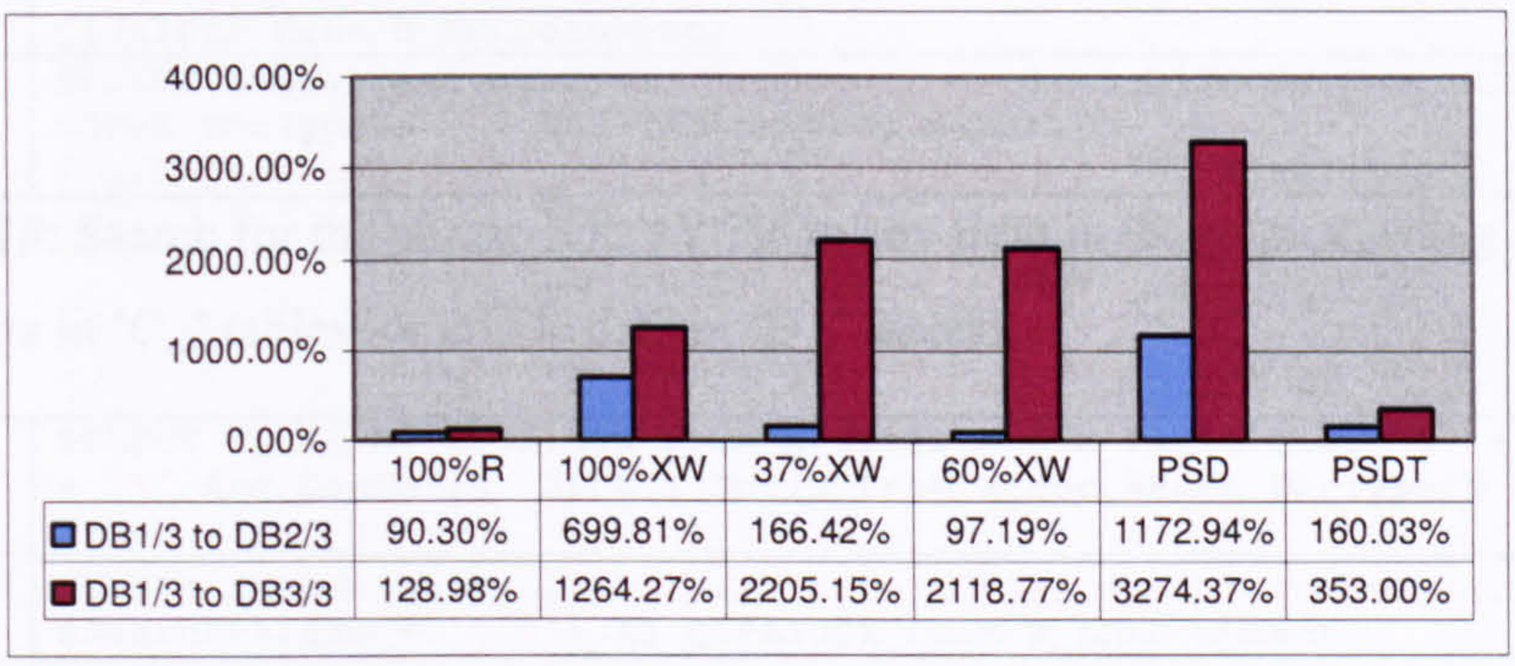


Figure 5.35 Query 8: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure represents the performance deterioration when the database size doubled and tripled for Q8. The worst performance was for the model ***PSD*** followed by ***37%XW*** and ***60%XW.***

5.3.3.2 Text search

These queries tested the information retrieval capabilities of the systems. Two cases were tested: uni-gram search (Q17) where the query contained one particular word and bi-gram and n-gram search (Q18) where multiple words were involved. The queries in this group were interpreted to match the data set used in the experiments as follows:

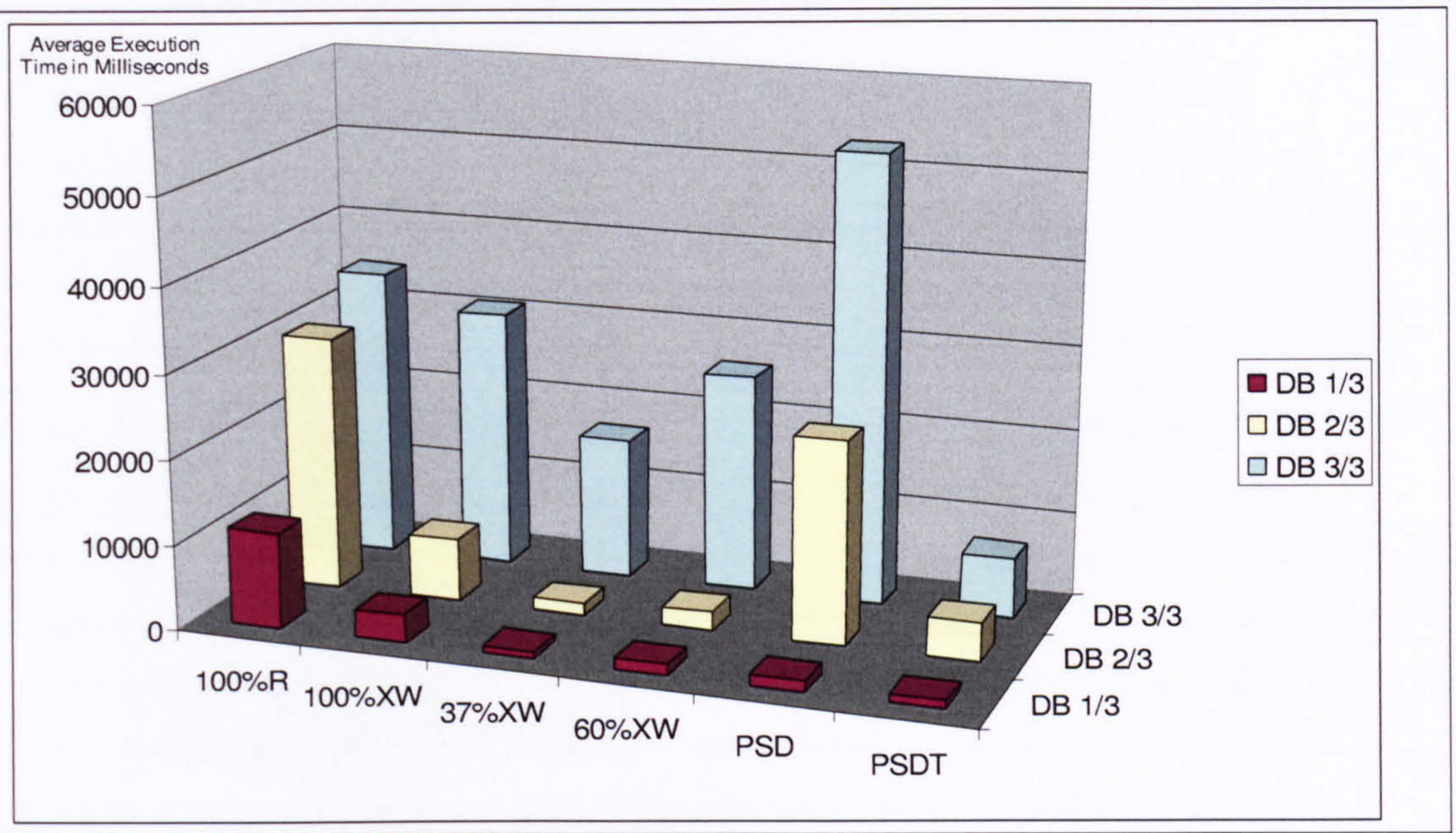
- ***Q17***: Search for the word XYZ in any field in the in-proceeding data in 'C_' tables (or article data in 'D_' tables).

SQL Syntax	SELECT cast ('<![CDATA[' + dbo.A_author.author + ']]>' AS XML) AS Q17A100R FROM dbo.A_author INNER JOIN dbo.A_Doc ON dbo.A_author.DocId = dbo.A_Doc.DocId WHERE Author like '%' + @SearchWord + '%' And A_Doc.DocTypeID = 1 UNION ALL SELECT cast ('<![CDATA[' + dbo.A_Editor.Editor + ']]>' AS XML) AS Q17A100R FROM dbo.A_Editor INNER JOIN dbo.A_Doc ON dbo.A_Editor.DocId = dbo.A_Doc.DocId WHERE Editor like '%' + @SearchWord + '%' And A_Doc.DocTypeID = 1 ... And so on for the remaining 20 tables (Address, Title, Booktitle, Pages, Year, Journal, volume, month, URL, EE, CDRom, Cite, Publisher, CrossRef, ISBN, Series, School, Chapter, Number, Note)
XQuery Syntax	SELECT xmldoc.query('for $x in /dblp/article WHERE ($x)/title/text()[contains(.,"' + @SearchWord + '")] return $x') AS Q17A100X From B_XMLDocument;
SQL/XQuery Syntax	SELECT XMLExtract.query('/inproceedings') AS Q17A60X From C_Doc WHERE DocTypeId = 2 And (XMLExtract.exist ('(/inproceedings/title/text())[contains(.,"' + @SearchWord + '")]') = 1);

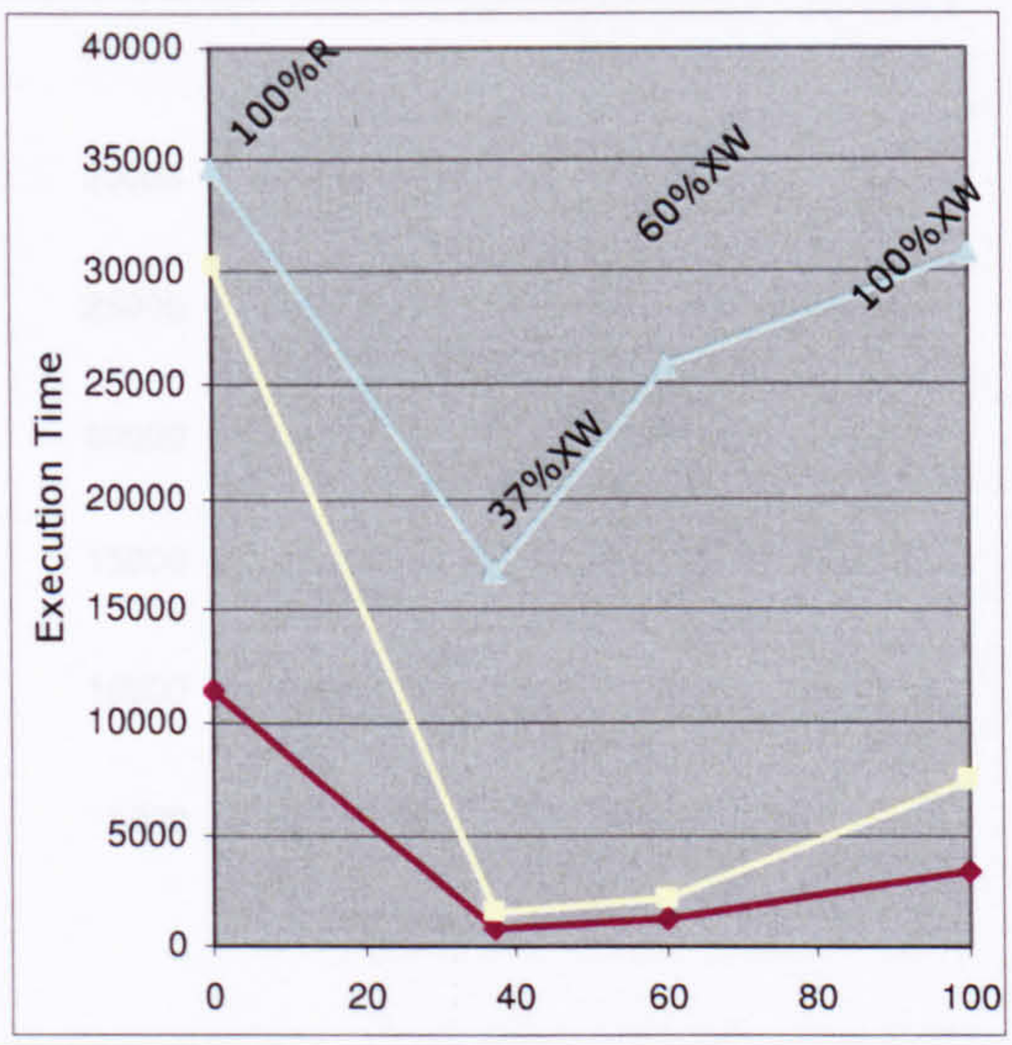
- ***Q18***: Search for the phrase XX YY ZZ in any field in the in-proceeding data in 'C_' tables (or article data in 'D_' tables).

SQL Syntax	SELECT Title AS Q18A100R from A_Title WHERE Title like '%' + @SearchPhrase + '%' And DocId in (SELECT DocId from A_Doc WHERE DocTypeID = 1) For XML Auto
XQuery Syntax	SELECT XMLDoc.query ('(dblp/article/title/text())[contains(.,"' + @SearchPhrase + '")]') AS Q18A100X from B_XMLDocument
SQL/XQuery Syntax	SELECT XMLExtract.query('/inproceedings/title') AS Q18A60X From C_Doc WHERE DocTypeId = 2 And (XMLExtract.exist('(/inproceedings/title/text())[contains(.,"' + @SearchPhrase + '")]') = 1);

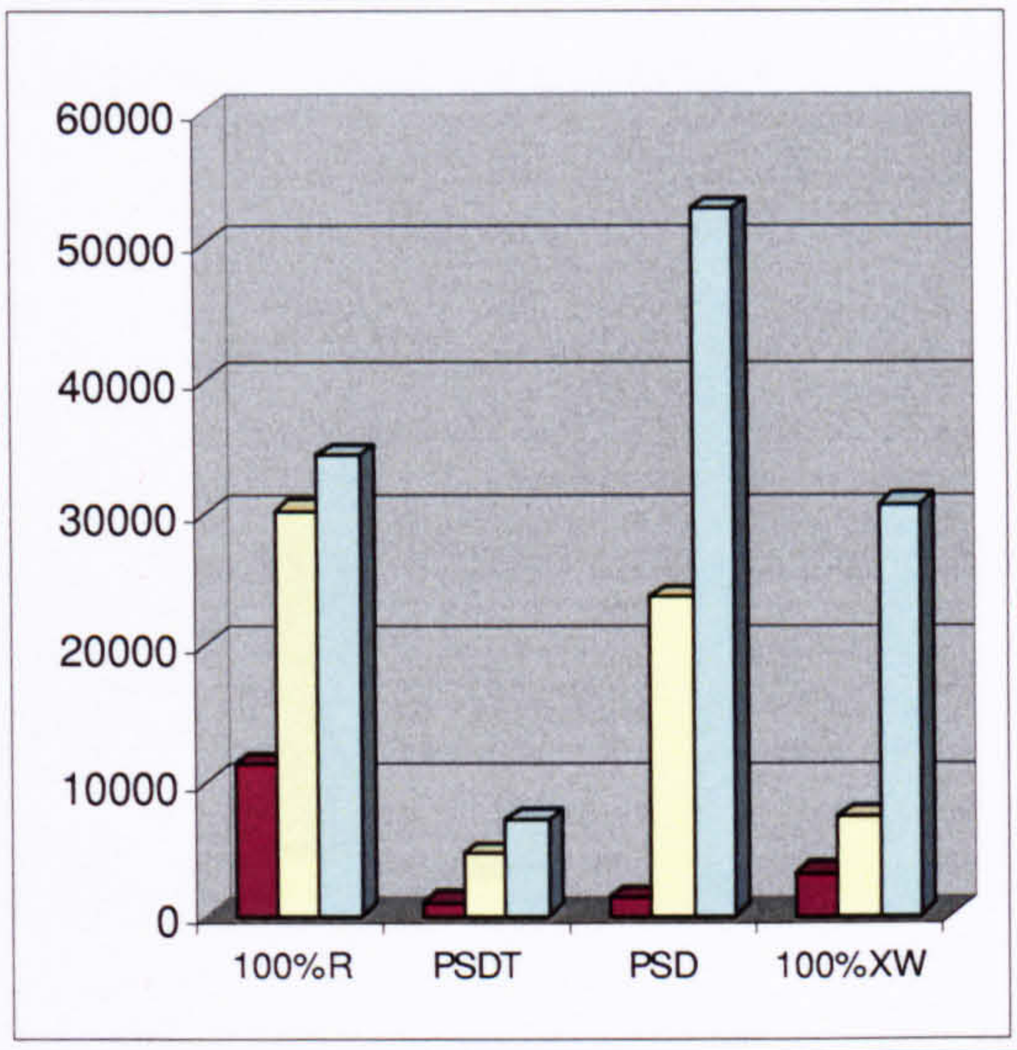
The following graphs show the results for this query



Q17 – Graph 1: Database Size vs. Storage Strategy vs. Performance



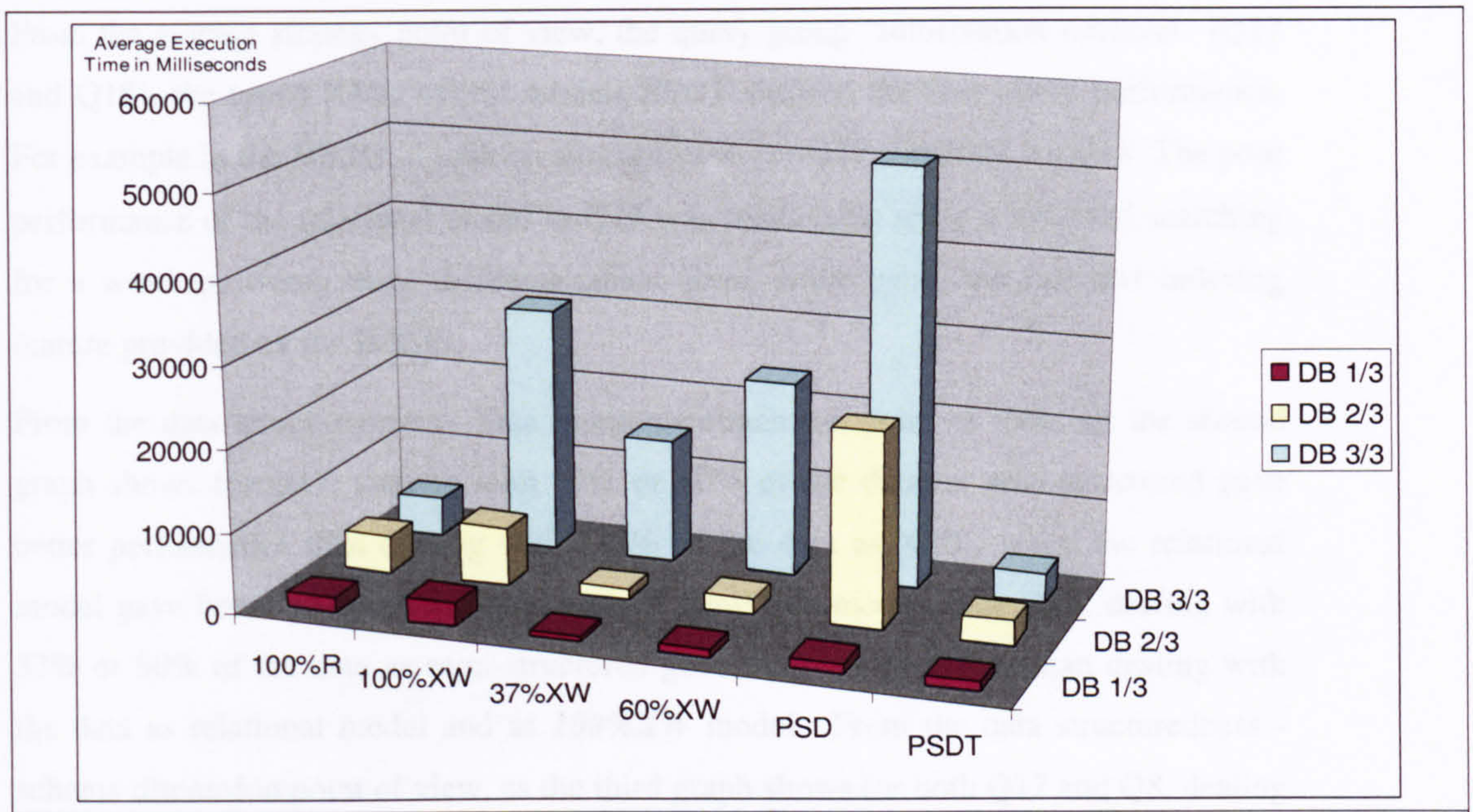
Q17 – Graph 2: Data Instances Dimension



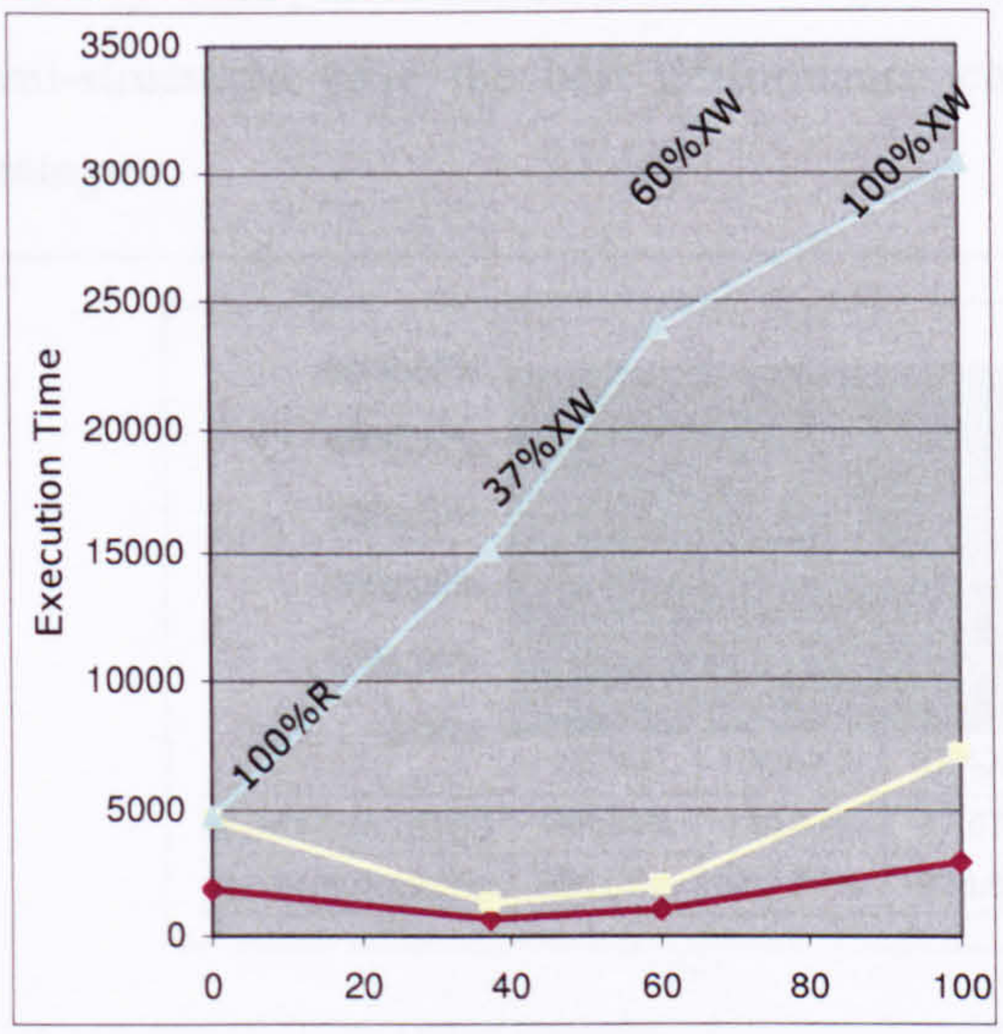
Q17 – Graph 3: Schema Dimension

	DB 1/3		DB 2/3		DB 3/3	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	11288	2523.41	30252	2647.40	34549	3981.95
100%XW	3315	1286.83	7408	1485.64	30749	2392.47
37%XW	820	132.63	1452	286.89	16635	2283.69
60%XW	1223	255.90	2159	288.54	25848	2136.88
PSD	1298	92.07	23844	1392.61	52799	2350.26
PSDT	999	231.92	4645	1729.06	7140	2529.06

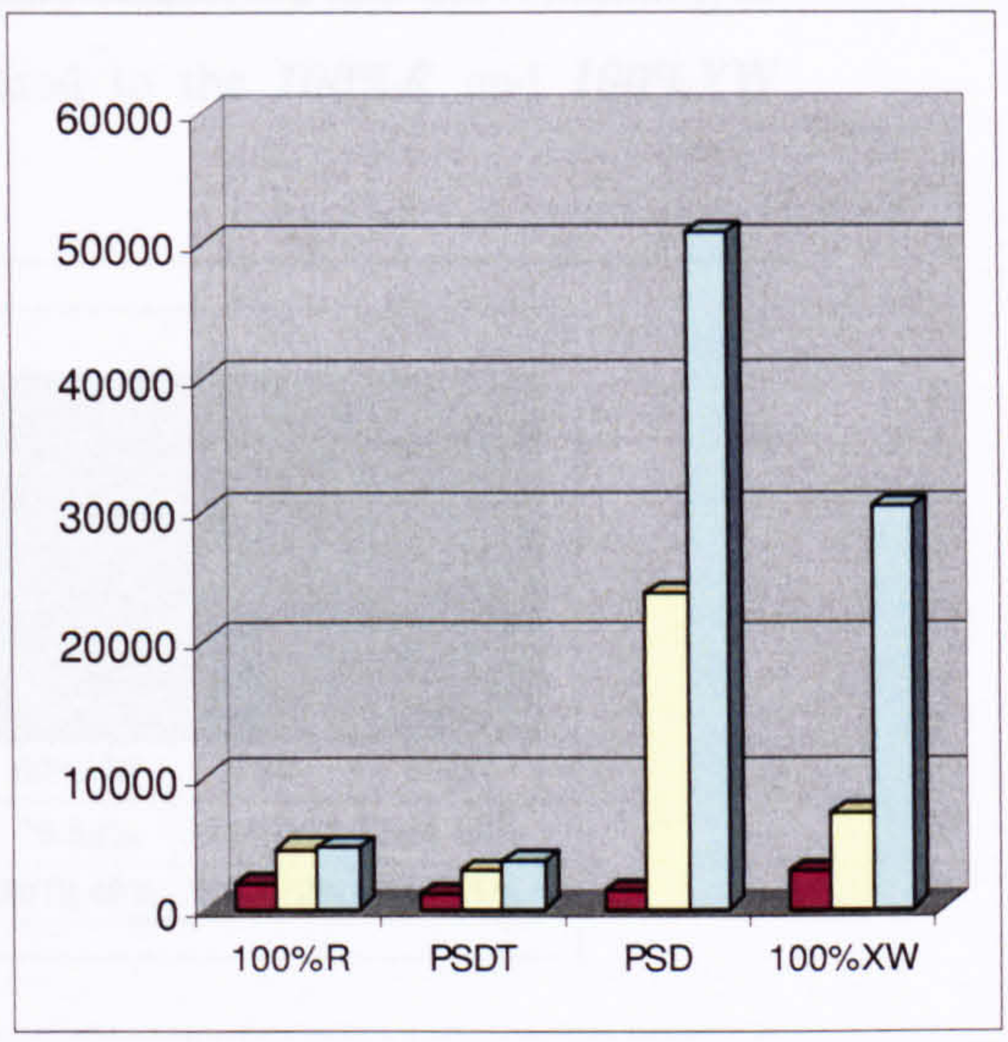
Figure 5.36 Query 17: Text Search - Uni-gram Search



Q18 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q18 – Graph 2: Data Instances Dimension



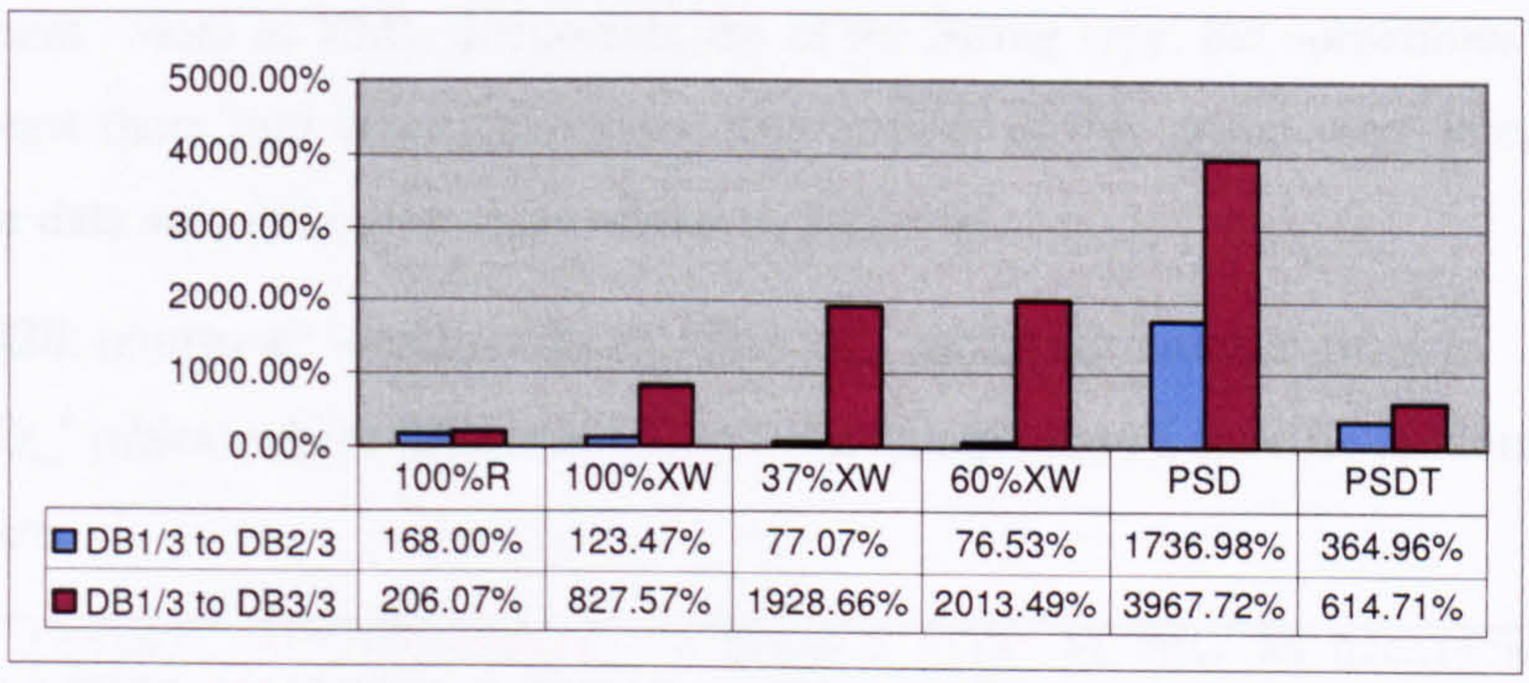
Q18 – Graph 3: Schema Dimension

	DB 1/3		DB 2/3		DB 3/3	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	1832	76.54	4573	226.27	4791	913.81
100%XW	2913	1318.59	7148	1525.97	30355	3798.85
37%XW	695	7.99	1269	45.44	15187	915.10
60%XW	1054	199.85	1926	35.79	23960	1284.36
PSD	1290	97.35	23687	1659.29	50928	3895.13
PSDT	1098	39.61	2983	462.78	3602	243.65

Figure 5.37 Query 18: Text Search - N-gram Search

From the storage strategy point of view, the query group "information retrieval" (Q17 and Q18), the typed XML hybrid models ***PSDT*** showed the best query performance. For example in the ***DB3/3***, it took on average 7140 for Q17 and 3602 for Q18. The poor performance of the relational model in Q17 was predictable since it involved searching for a word in twenty three different tables. Even while using the full text indexing feature provided by the DBMS.

From the data structuredness - data instances dimension point of view, as the second graph shows for Q17, dealing with 37% or 60% of the data as semi-structured gave better performance than dealing with 100% of the data as XML, while the relational model gave better performance than both of these two models. For Q18, dealing with 37% or 60% of the data as semi-structured gave better performance than dealing with the data as relational model and as ***100%XW*** model. From the data structuredness - schema dimension point of view, as the third graph shows for both Q17 and Q8, dealing with a specific part of the schema as in ***PSDT*** as highly-structured and the remaining as semi-structured gave the best performance compared to the ***100%R*** and ***100%XW*** strategies.



	100%R	100%XW	37%XW	60%XW	PSD	PSDT
DB1/3 to DB2/3	168.00%	123.47%	77.07%	76.53%	1736.98%	364.96%
DB1/3 to DB3/3	206.07%	827.57%	1928.66%	2013.49%	3967.72%	614.71%

Figure 5.38 Query 17: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure illustrates the performance deterioration when the database size doubled and tripled for Q17. The worst performance was for the model ***PSD.***

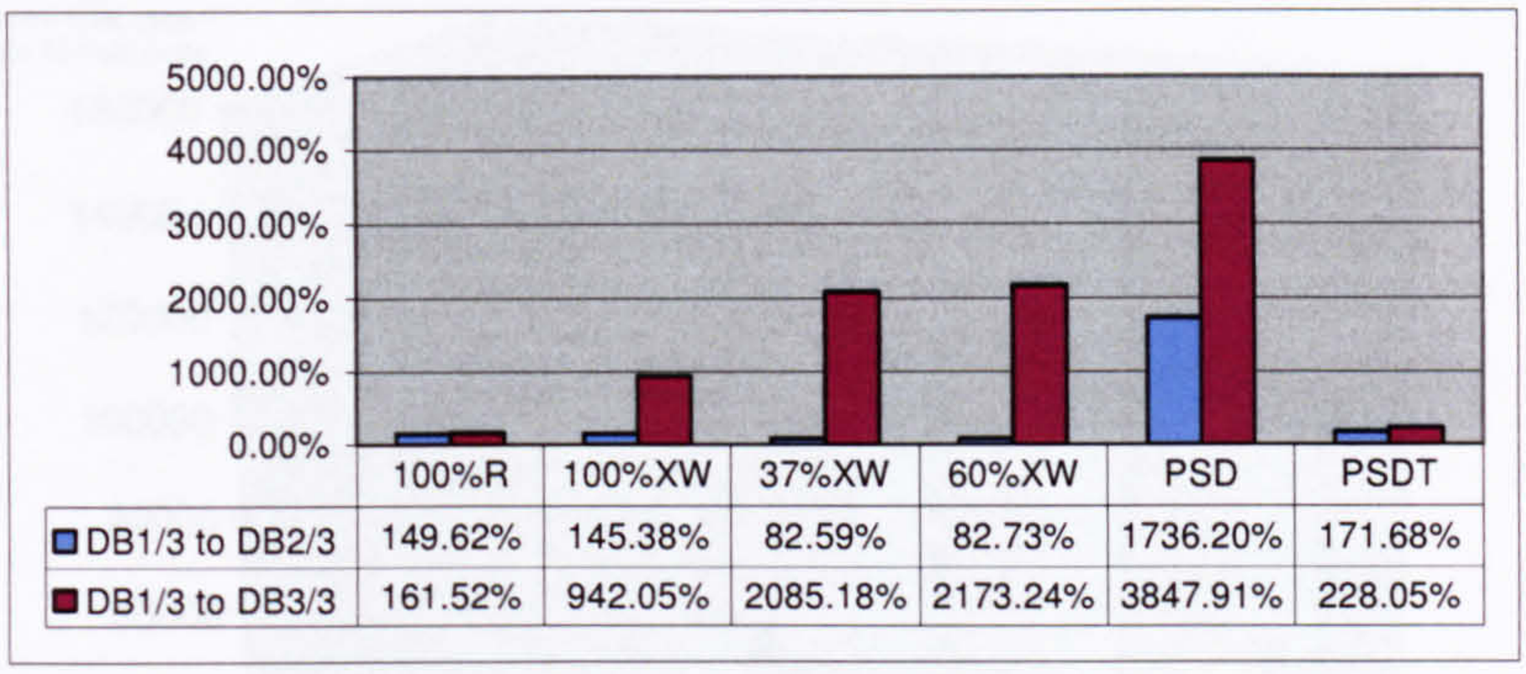


Figure 5.39 Query 18: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q18. The worst performance was for the model ***PSD*** followed by ***37%XW*** and ***60%XW.***

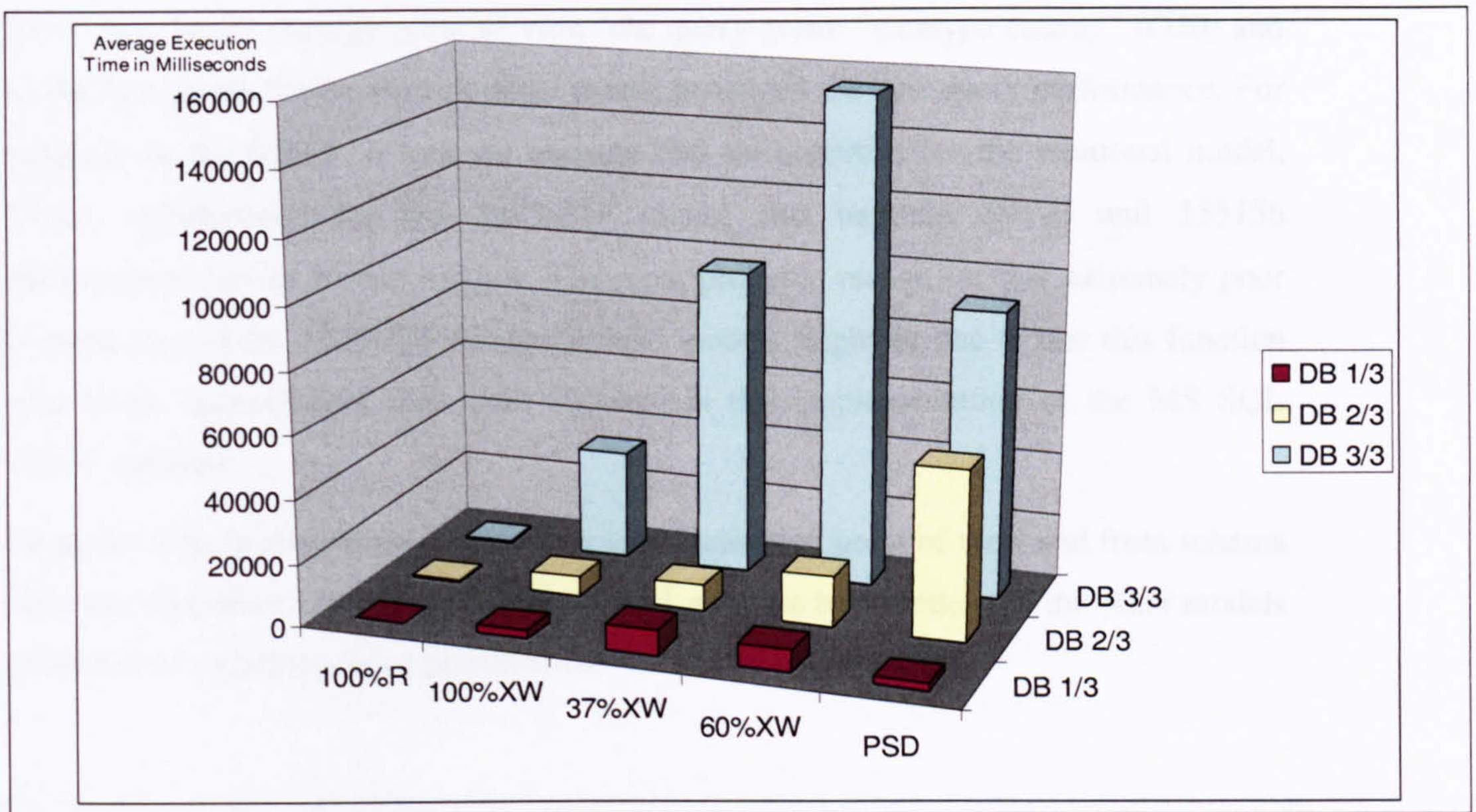
5.3.3.3 Datatype Casting

The element values in XML documents are of the String type, but sometimes there is a need to cast them into other data types. The queries in this group were interpreted to match the data set used in the experiments as follows:

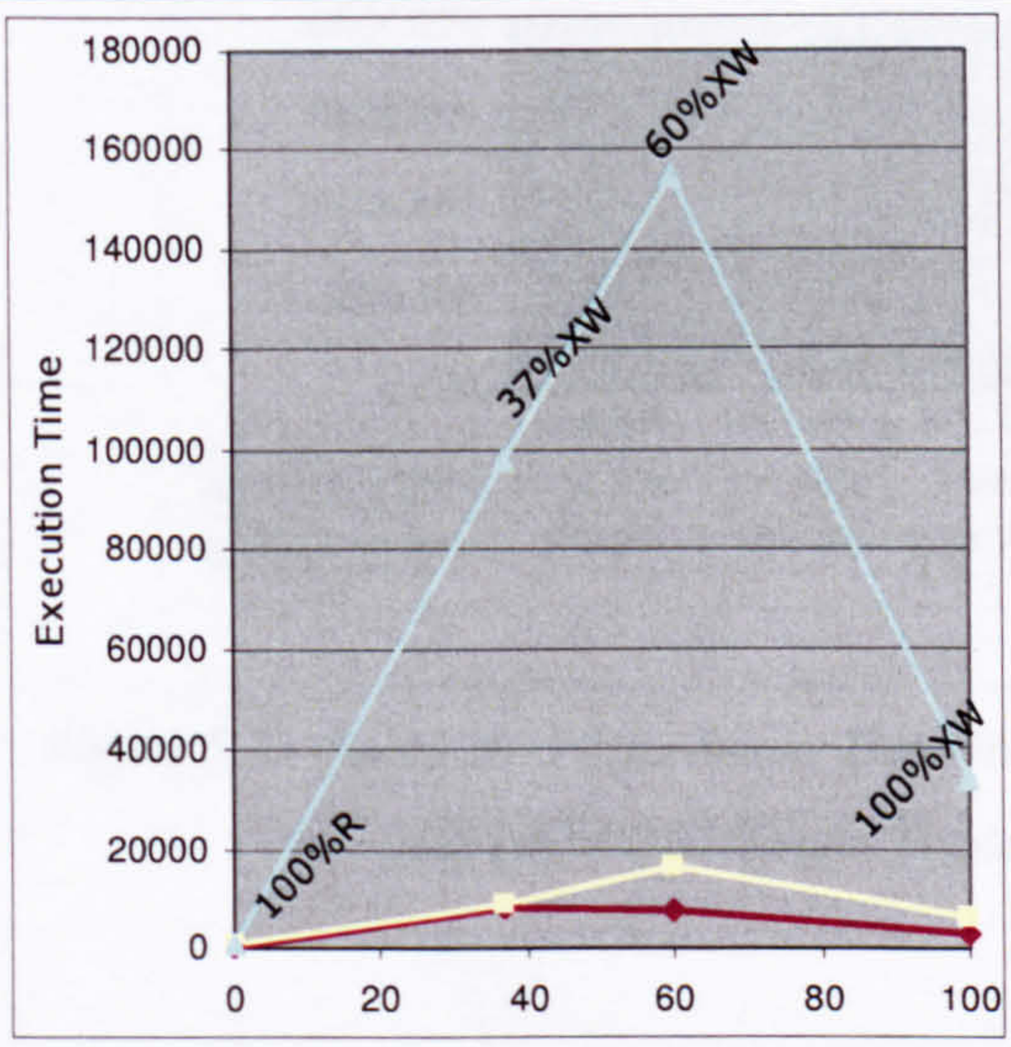
- ***Q20***: returns all in-proceedings' titles 'C_' tables (or articles' titles in 'D_' tables) where their title's length was longer than a specific random size.

SQL Syntax	`SELECT CAST ('<![CDATA[' + Title + ']]>' AS XML) AS Q20A100R from A_Title WHERE Len(Title) > @TitleLengthBiggerThan and docid in (SELECT docid from A_Doc WHERE doctypeid = 1)`
XQuery Syntax	`SELECT xmldoc.query('for $x in /dblp/article/title WHERE ($x)/text()[string-length()>' + CAST( @TitleLengthBiggerThan AS varchar(5)) + '] return $x') AS Q20A100X From B_XMLDocument;`
SQL/XQuery Syntax	`SELECT XMLExtract.query('/inproceedings/title') AS Q20A60X From C_Doc WHERE DocTypeId = 2 And XMLExtract.exist('(/inproceedings/title/text())[string-length() > ' + CAST( @TitleLengthBiggerThan AS varchar(5)) + ']') = 1;`

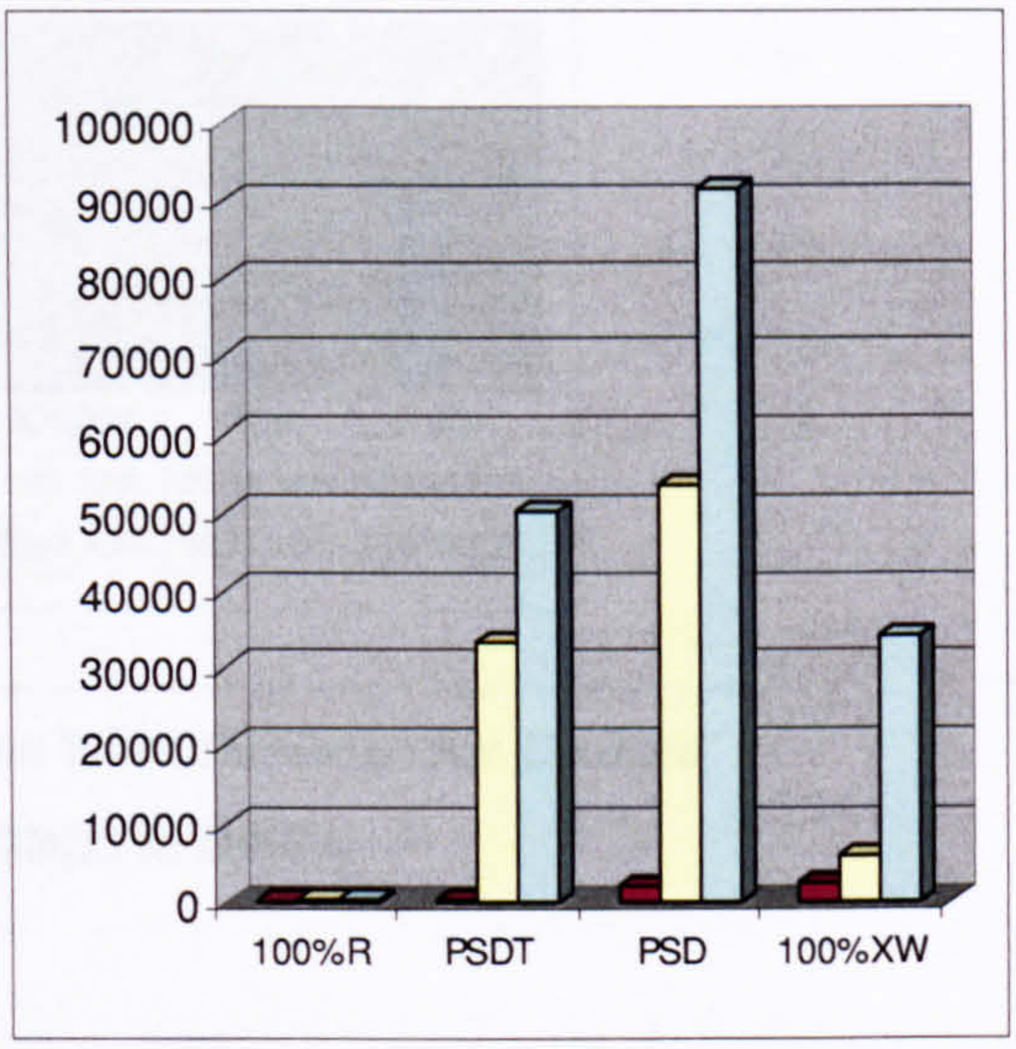
The following graphs show the results for this query



Q20 – Graph 1: Database Size vs. Storage Strategy vs. Performance



Q20 – Graph 2: Data Instances Dimension



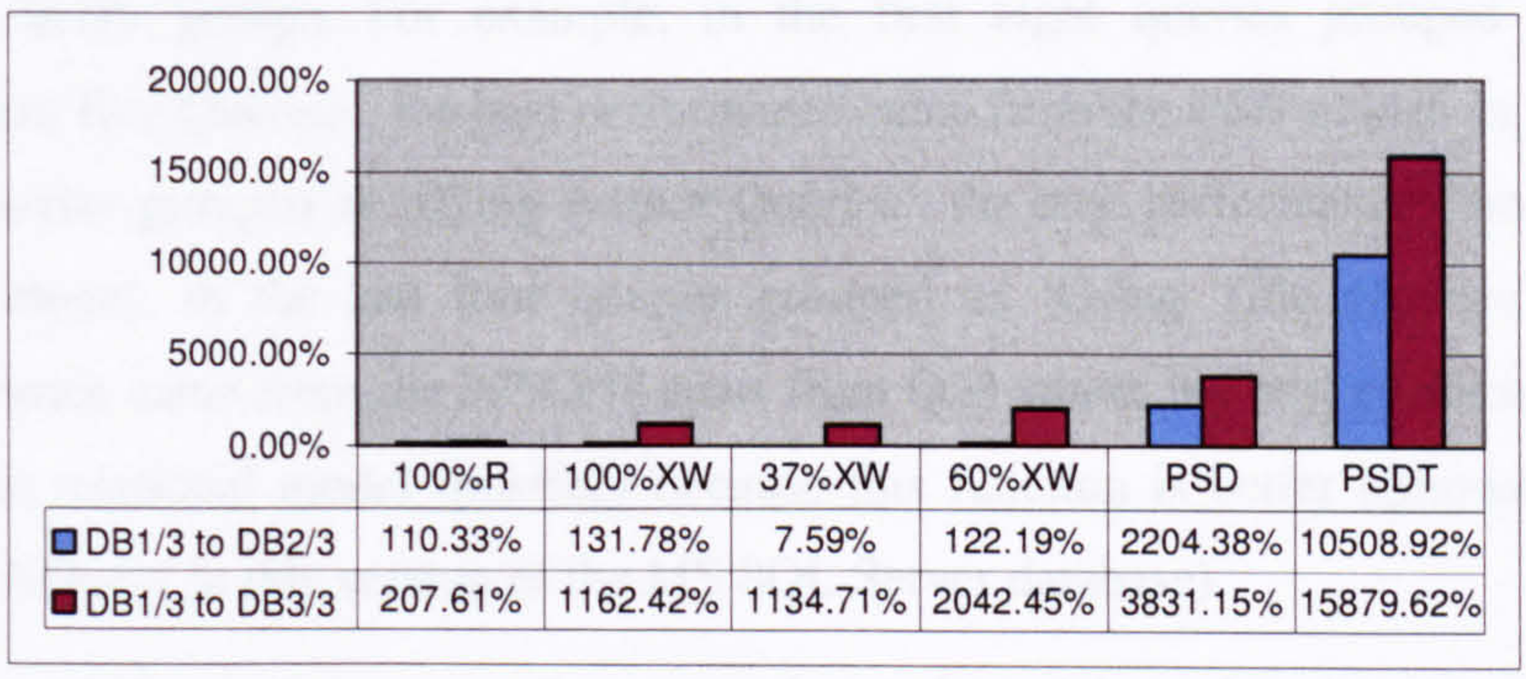
Q20 – Graph 3: Schema Dimension

	DB 1/3		*DB 2/3*		*DB 3/3*	
Model	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*	*Average*	*Std. Dev.*
100%R	184	22.26	387	72.31	566	104.07
100%XW	2706	306.89	6272	1409.53	34161	7450.52
37%XW	7906	408.88	8506	737.72	97616	5672.99
60%XW	7242	777.86	16091	718.23	155156	7462.53
PSD	2331	497.50	53715	5678.67	91635	18660.45
PSDT	314	40.62	33312	2807.87	50176	1796.30

Figure 5.40 Query 20: Data Type Cast

From the storage strategy point of view, the query group "datatype casting" (Q20) and as the first graph shows, the relational model produced the best query performance. For example in the ***DB3/3***, it took on average 566 milliseconds for the relational model, 34161 milliseconds for the ***100%XW*** model and between 50176 and 155156 milliseconds for the hybrid models. The most probable reason for this extremely poor performance of the ***100%XW*** and the hybrid models might be due to that this function was better optimized in SQL than XQuery in this implementation of the MS SQL Server database.

From the data structuredness - data instances dimension point of view and from schema dimension point of view, the relational model gave the best model and the other models produced an extremely poor performance.



	100%R	100%XW	37%XW	60%XW	PSD	PSDT
DB1/3 to DB2/3	110.33%	131.78%	7.59%	122.19%	2204.38%	10508.92%
DB1/3 to DB3/3	207.61%	1162.42%	1134.71%	2042.45%	3831.15%	15879.62%

Figure 5.41 Query 20: Performance Deterioration When Database Size Doubled (DB1/3 to DB2/3) and Tripled (DB1/3 to DB3/3)

From the database size point of view, the above figure shows the performance deterioration when the database size doubled and tripled for Q20. The worst performance was for the model ***PSDT.***

5.4 Results Analysis

The previous section presented the experiments' results and explained them based on their individual query functionality (from Q1 to Q20). This section is dedicated to an analysis of the overall queries performance with respect to the different variants these

experiments were designed to measure. These variants are: storage strategy, query type, data structuredness, scalability and database storage size.

The interdependencies of the first three variants are presented in the following subsection 5.4.1 "experiments' overall analysis", while the following two subsections are dedicated to scalability and database storage size respectively. This is followed by a discussion about the experiments' limitations and general findings from these results in sections 5.5 and 5.6 respectively.

5.4.1 Experiments' Overall Analysis

The main aim of the experiments, as explained in the previous chapter, was to compare the relative performance of the different storage models when dealing with partially-structured data by using different queries functionality. Reflecting on the results produced by the experiments, it can be seen that none of the systems performed well in all the query groups. For example, in the first eight queries grouped as 'Using Document Key Queries', the best performance came from the ***PSD*** model. In the second eight queries grouped as 'Using Author Queries', the best performance came from the ***PSDA*** model. In the last four queries grouped as 'Using Title Queries', the best performance came from the ***37%XW*** apart from Q20 where the best performance came from the relational model (possibly because this function is better optimised in SQL than in XQuery in this version of the MS SQL Server database).

In some of the query groups, the pure relational model produced poor performance compared to the proposed hybrid model (such as irregular data and retrieval of individual documents). This is likely to be due to the fact that the relational model shreds the data and it took more time to reassemble it using the join operation. Storing all XML data into one field always produced a poorer performance than dividing the data into smaller XML elements. In a number of queries, the ***100%XW*** model showed a good performance, but the other hybrid models also showed a good performance (such as quantification group Q6 and Q7).

The main hypothesis of the thesis from section 3.2.2 was:

> *For the class of XML documents which contains both a prescribed highly-structured part and a semi-structured part, performance enhancement may be achieved over existing query processing techniques for semi-structured documents by using relational database query processing and optimisation*

technology to exploit pre-knowledge of the prescribed highly-structured part of the data.

The question that this hypothesis poses is: can dealing with parts of the XML document as structured and other parts as semi-structured produce better query performance than storing the whole of the data as semi-structured or all of it as highly-structured? To answer this question, the following table summarises the results for ***DB3/3*** (the biggest database in size) for the different queries functionality and gives them a relative score according to their performance. Each column represents a storage strategy and each row represents a query type. In each cell, the number indicates the score of the relative performance of the specific storage strategy for specific query type against the best performance in this query type. One represents the best performance and other numbers represent the relative performance of this model in comparison to the best performance. For example, In Q1, the best performance was achieved by ***37%XW*** with 63 milliseconds, so its relative performance is one. The ***PSD*** average performance was 65 milliseconds. So its relative performance is 65/63 = 1.03, and is considered as 1. The ***60%XW*** average performance was 74 milliseconds. So its relative performance is 74/63 = 1.17, and is considered as 1. The 100%R average performance was 156 milliseconds. So its relative performance is 156/63 = 2.47, and is considered as 2. Finally, The 100%XW average performance was 522 milliseconds. So its relative performance is 522/63 = 8.28, and is considered as 8.

	100%R	100%XW	37%XW	60%XW	PSD	PSDA	PSDT
Q1 Exact Match (Shallow)	2	8	1	1	1	-	-
Q9 Path expressions (Multiple unknown elements)	8	13	1	1	1	-	-
Q12 Document construction (preserving the structure)	6	16	2	3	1	-	-
Q13 Document construction (transforming the structure)	49	876	5	7	1	-	-
Q14 Irregular data (missing elements)	16	32	1	1	1	-	-
Q15 Irregular data (empty values)	185	18	1	1	1	-	-
Q16 Retrieval of individual documents	1621	16	2	3	1	-	-
Q19 References and joins	17	18	6	7	1	-	-
Q2 Exact Match (Deep)	2	5	2	3	3	1	-
Q3 Function application (Count)	17	32	7	6	8	1	-
Q4 Ordered access (current matching position)	42	32	7	6	7	1	-
Q5 Ordered access (absolute order)	13	23	5	5	5	1	-
Q6 Quantification (existentially)	2	2	2	2	2	1	-
Q7 Quantification (universally)	4	1	2	2	2	1	-
Q10 Sorting (string type)	28	50	17	20	6	1	-
Q11 Sorting (non-string type)	19	578	7	9	6	1	-
Q8 Path expressions (one unknown element)	1	7	3	5	11	-	1
Q17 Text Search (uni-gram)	5	4	2	4	7	-	1
Q18 Text Search (n-gram)	1	8	4	7	14	-	1
Q20 Datatype Casting	1	60	172	274	162	-	89

Table 5.1 Summary of all the relative performance results

The relative performance in Table 5.1 suggests some interesting relative characteristics of the storage strategies. None performed well for all query groups. For example, though PSD performed well for accessing via the document key (Qs 1, 9, 12-16 and 19), it was relatively poor where author names are queried (Qs 3-5, 10 and 11). In the latter group, the PSDA model had the best performance, which possibly demonstrates the

relative efficiency of querying conventional atomic attributes via indices, in contrast to using typed XML instances. However, the poor performance of 100%R compared to the hybrid approach, illustrates the advantage of storing semi-structured data into an XML data field rather than shredding it into a relational structure. Examples of the superiority of the hybrid model over pure structure mapping include irregular data (Q14, 15), document construction (Q12 and 13) and retrieval of individual documents (Q16). This is likely to be due to the fact that the relational model shreds the data, thus incurring the overhead of multiple join operations to re-assemble a document. This was very clear in Q16. This shredding is reduced by the hybrid approach, thus reducing the number of joins and disk access. However, as a counter example, though three of the queries in which article titles are queried (Qs 8, 17 and 18) performed well with ***PSDT***, this was not the case for the fourth (Q20), in which the length of the title was tested, where best performance was achieved using 100%R. Other examples include queries (Q1, Q2, Q6, Q8, Q18 and Q20) where the ***100%R*** model produced a good performance. Storing the whole XML data as a typed XML instance (***100%XW***) always produced a poorer performance than strategies which divided the data into smaller XML elements. In a small number of queries, the ***100%XW*** model showed good performance, but in these cases so did the other hybrid models (e.g., quantification (Q6-7)).

Thus, the messages are mixed. However, if we consider the performance of the specific strategies the results give a clearer view.

Pure structure mapping (***100%R***): In most instances this approach produced poor performance, relative to the hybrid approaches. The exceptions were Q1, Q2, Q6, Q8, Q18 and Q20. This was unexpected, since the DBLP data set is relatively well structured and is therefore well suited to a structure mapping approach. Experiments with different data set sizes demonstrated a near linear deterioration in query time as size increased, with an average deterioration of 58.52% to 124.87%, respectively, as the size doubled and tripled. However, this representation seems particularly ill-suited for irregular data (Q14), with a possibly exponential deterioration from 300% to 1052.63%, as the size doubled and tripled. Thus, data set size and application characteristics are important factors when considering this storage strategy.

Pure use of typed XML data types (***100%XW***): Using only typed XML instances produced poor performance in 90% of the cases. The exceptions are Q6 and Q7 (quantification). However, in general, this approach was poor for all conventional

relational-style retrievals, involving selection, projection and join. Also, the deterioration in query performance was more extreme than for ***100%R***. On average deterioration was, respectively, 120.68% and 344.53% as the data set size doubled and tripled. The worst case was for the path expression query, Q8, where the deterioration was from 699.81% to 1264.27%. Thus, average performance was poor and the deterioration associated with the increase in the data set size is worse than linear, suggesting this approach has viability only for small data sets.

Vertical hybrid approaches (***PSD***, ***PSDA*** and ***PDST***): In this approach part of the XML schema common to all repeating instances is structure mapped, and the rest are represented as typed XML instances. As expected, this approach outperformed the two pure base approaches in cases where the query keywords were within the structure mapped part. However, this superiority was surprising for the text searches (Q17 and 18) where all fields had to be accessed. Also, our anticipation was that performance would continue to improve as the ratio of structured to semi-structured data increased, since we believed conventional relation querying was likely to outperform the added XML facilities. But, given the fact that the DBLP data set is mainly well structured, if this were true, the best performances would be for ***100%R***, which was not the case, since there seems to be a threshold beyond which performance then starts to deteriorate. In most cases, the query performance deterioration with increase in data set size was near linear. For ***PSD***, ***PSDA*** and ***PSDT*** this deterioration was respectively from 355.07% to 768.81%, 25.29% to 62.85%, and 2801.40% to 4268.85% as size doubled and tripled. ***PSDA***'s worst deterioration was when processing quantification (Q6), where the increase in query time was from 126.23% to 290.16%, as the size doubled and tripled. PSDT deteriorated most for datatype casting (Q20), from 10508.92% to 15879.62% as size doubled and tripled. PSD has two step changes with respect to query performance. These were, also for datatype casting (Q20), with a deterioration of 2204.38% when the size doubled, and for text searching (Q17), where there was 3968.72% deterioration when size tripled. Thus, though our hypothesis is largely born out by these results, there are other factors which any decision model must take into account, including the overheads incurred by data shredding, and the impact of data set size.

Horizontal hybrid approaches (***37%XW***, ***60%XW***): In this approach some types of repeating instances are structure mapped and the others are stored as typed XML

instances. This approach mainly produced a middle ranking performance, with similar performances from ***37%XW*** and ***60%XW***, and neither consistently outperforming the other. However, in the majority of cases, both out performed ***100%R*** and ***100%XW***. Specifically, in 85% and 80% of cases this hybrid models respectively gave better or similar performance than ***100%R*** and ***100%XW***. Thus, there seems to be an advantage in horizontally partitioning data into structured and semi-structured representation, as well as the more obvious benefits of the vertical approach. More worryingly, the approach also demonstrated a possibly exponential deterioration, as data set size increased. Both 37%XW and 60%XW exhibited a similar average deterioration, from 27.49% to 394.70% and from 28.82% and 444.63%, as the size doubled and tripled. However, for the path expression query, Q8, the deterioration in performance was from 166.24% to 2205.15%. Thus, although in most cases this approach improves on pure structure mapping, data set size is a critical consideration, as the approach does not appear to scale to very large data sets.

5.4.2 Scalability

In this section, the results are analysed from the scalability point of view. Scalability can be seen in three different ways; database size, database complexity or the query or application complexity. The later is dealt with in the experiment by using different query sets, some of which were more complex than the others (for example, shallow exact match and deep exact match).

The experiments were designed to test the impact of database size growth for the different query performances in that there were three database sizes. These three sizes were ***DB3/3*** which represents the original XML document, ***DB2/3*** which represents approximately two thirds of the original XML document and finally ***DB1/3*** which represents approximately one third of the original XML document size. They had exactly the same structure and varied in database size only.

When the results were presented in section 5.3 for each query group, the effect of the change of the database size was presented for each query group; when the database sized doubled from ***DB3/3*** to ***DB2/3*** and when the database size tripled from ***DB1/3*** to ***DB3/3***. To summarize these results:

- In the relational model ***100%R***, the maximum increase in performance when the database size doubled was 300.00% and when the database size tripled was

1052.63% in Q14. The average increase in all queries was 58.52% and 124.87% when the database size doubled and tripled respectively.

- In the ***100%XW***, the maximum increase in performance when the database size doubled was 699.81% and when the database size tripled was 1264.27% in Q8. The average increase in all queries was 120.68% and 344.53% when the database size doubled and tripled respectively.

- In the ***37%XW***, the maximum increase in performance when the database size doubled was 166.24% and when the database size tripled was 2205.15% in Q8. The average increase in all queries was 27.49% and 394.70% when the database size doubled and tripled respectively.

- In the ***60%XW***, the maximum increase in performance when the database size doubled was 122.19% in Q20 and when the database size tripled was 2118.77% in Q8. The average increase in all queries was 28.82% and 444.63% when the database size doubled and tripled respectively.

- In the ***PSD***, the maximum increase in performance when the database size doubled was 2204.38% in Q20 and when the database size tripled was 3968.72% in Q17. The average increase in all queries was 355.07% and 768.81% when the database size doubled and tripled respectively.

- In the ***PSDA***, the maximum increase in performance when the database size doubled was 126.23% and when the database size tripled was 290.16% in Q6. The average increase in all queries was 25.29% and 62.85% when the database size doubled and tripled respectively. These results were based on eight queries only as shown in the group 'using author queries'

- In the ***PSDT***, the maximum increase in performance when the database size doubled was 10508.92% and when the database size tripled was 15879.62% in Q20. The average increase in all queries was 2801.40% and 4268.85% when the database size doubled and tripled respectively. These results were based on four queries only as in the group 'using title queries'

The above data can be summarised in the following table

System	Database Size Doubled	Database Size Tripled
100%R	58.52%	124.87%
100%XW	120.68%	344.53%
37%XW	27.49%	394.70%
60%XW	28.82%	444.63%
PSD	355.07%	768.81%
PSDA	25.29%	62.85%
PSDT	2801.40%	4268.85%

Table 5.2 Average Deterioration In Query Performance

The above table shows that when the database doubled, the 37%XW, 60%XW and PSDA gave – on average of the twenty queries - the lowest deterioration in query performance. However, when the database tripled, the hybrid model PSDA gave the lowest deterioration in query performance. This shows that from the database size scalability point of view, hybrid models can produce good overall performance. The other hybrid models (PSD and PSDT) gave a worse performance than the relational model.

The above conclusion is indicative and can not be interpreted as a general rule. This is due to the fact that this is the average deterioration and in a specific case one model could be better or worse than the other model. The above results should be interpreted only in the scope of the hardware and the software used in the experiments. Any change of the processing power or the available memory (therefore the available caching) to the database management system or the software used in running the experiment could produce different results.

5.4.3 Database Storage Size

The impact of the storage size (the amount of storage space needed to store the data in the storage media, i.e. the computer disk) for the different storage models is one of the factors affecting the choice of a storage model. However, it is not an important factor, in view of the sharp decrease in cost of the storage media in the recent years. This section

presents the different storage models used in the experiments and discusses their storage size.

As all the models used MS SQL Server as the database management system, the relative storage size for each of the storage strategies is calculated in relation to the original XML document size. This gives a relative indication of how each storage model was used in terms of storage space. This comparison is based on the biggest database (***DB3/3***), where the size of the initial XML document was 341,503 kilobytes (KB). The following table summarises this comparison. The 'data' column shows how many KB was used to store the data, the 'indices' column shows how many KB was used to store the different indices and finally the 'unused' column shows how many KB was reserved by the database management system but not actually used for either data or indices storage. The 'total' column is the summation of the previous three columns ('data', 'indices' and 'unused'). Finally, the last column shows the ratio between the total size and the original XML document size.

Storage Model	Data (KB)	Indices (KB)	Unused (KB)	Total (KB)	Ratio between total size and the original XML document size
100%R	355,824	231,728	3,120	590,672	1.73
37%XW	520,400	1,365,480	6,992	1,892,872	5.54
60%XW	661,480	2,065,568	19,272	2,746,320	8.04
100%XW	563,448	3,455,800	288	4,019,536	11.77

Table 5.3 Database Storage Sizes

As the above comparison shows, the best model in terms of the storage size is the relational model with about 1.73 times of the original document size. The worse storage size comes as a result of storing the entire document in one XML data field with about 11.77 times of the original document size. The hybrid models are in the middle between these two extremes as they are a hybrid between relational and XML data fields. This is probably due to the fact that storing data into an XML data field as a new technology is not optimised in terms of storage size and there maybe in the future more optimised ways to store XML data using less storage size.

5.5 Experiments Limitations

The first limitation of the experiment is that it depends on one data set (that is to say one XML document) to represent its verdict. The choice of this data set was discussed and justified in section 4.4.1. The design of the experiments took this limitation into consideration by varying the different ratios of semi-structured data to highly-structured data (for example by using two different ways to slice the data (horizontal and vertical) and by using different percentages of the data as semi- and highly-structured as explained in details in section 5.2.1). However, the consequence of using one XML document was that a variant within the experiment had to be the extent to which the structured data was interpreted as partially structured data, since it was not possible to vary the inherent structuredness of the data.

Another limitation regarding the use of existing data is that some of the query groups were not be tested properly due to the nature of the data set as follows:

- Irregular data – empty elements: there were no empty data in the data set
- References and joins: the data set consists of a single XML document

Finally, using only one XML document with the same structure allows only testing the scalability from the database size point of view and not from the database complexity point of view. This is because the experiments used three different database sizes (***DB1/3***, ***DB2/3*** and ***DB3/3***) for the same data set, which means the same structure complexity.

The second limitation was using one database technology in all the experiments (that is to say using MS SQL Server as the test bed for all the different storage models). This is a general problem to any researcher in this field. As storing XML in relational database is still emerging and in its early days, new versions would be expected with more storage and query optimisations and there is a higher probability of variations in performance between different database management systems. To overcome this point, the analysis of the results was represented in a relative manner whenever possible, to give an indication of how the performance between the different storage models could be compared.

5.6 Findings and Conclusions

The above results showed that there was no single storage model that outperforms all other models in all casses. The choice of the right storage strategy must depend on the query work load, the type of the data used and the degree of structuredness of the data. Also, although the absolute timing are specific to the experiment set up, they are indicative of relative performance, given the database and application characteristics. Therefore a potential application of these results could be an evidence-based method to inform the design of relationally-represented XML databases. To conclude this chapter, I elaborate on this possibility by proposing a flow chart to give the database designer a logical path to follow in order to build his/her database storage structure or to enhance an existing database storage structure based on the experiments' results. Each question or decision in this flow chart is based on the previous experiments and their results.

The main flow chart is divided into several smaller flow charts (numbered I, II, III...). Each of these flow charts are described first then presented and linked to another flow chart as appropriate.

In the flow chart I, the basic question is how the database designer can describe the structuredness characteristics of the data. This is based on the analysis of the degree of the structuredness as in section 3.3. The possible responses are either totally structured, totally unstructured or somewhere in between. In the first case, the best storage scenario is to use the relational model only. In case that the data is totally un-structured there are two ways to go, if there is an XML Schema or DTD that can describe the structure of this data, the base scenario is to store the data into one XML data field (as in the 100%XW scenario) and if not, then the best scenario is to use the Information Retrieval techniques to store the data. The final possible response is that the data is described as a hybrid between highly-structured and semi-structured (or un-structured). This is the area specific to this research and this branch is continued in flow chart II.

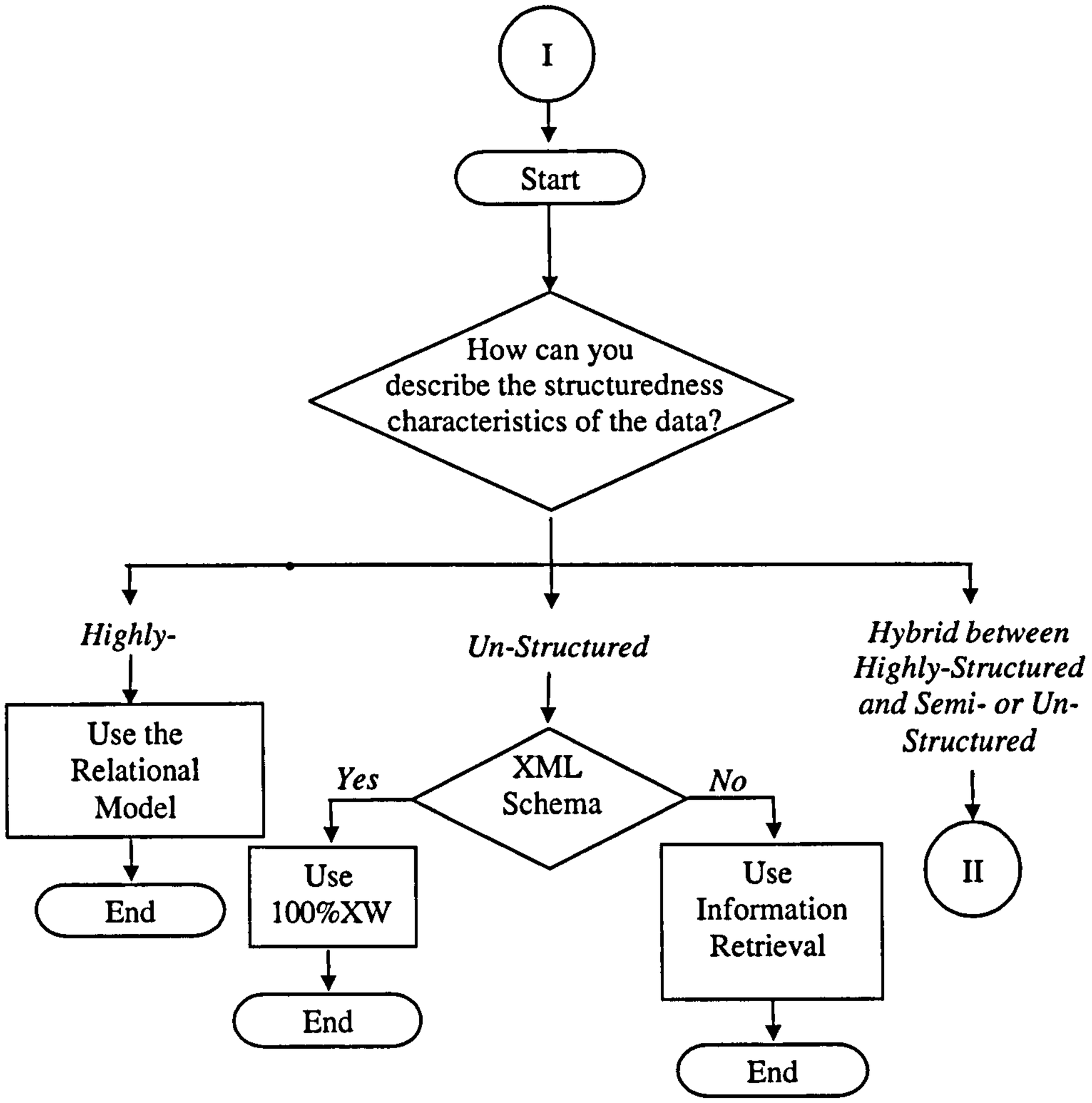


Figure 5.42 Flow Chart I

In the second flow chart (II), the basic question is how the main query work load can be described. There are three different scenarios; the first scenario involves querying the data using the key value, this is similar to the queries in section 5.3.1 'Using Document Key'. The result of this path is described in flow chart III. The second scenario follows querying the data using a string value, this is similar to the queries in section 5.3.2 'Using Author Queries'. The result of this path is described in flow chart IV. The last scenario involves querying the data using a part of a string value, this is similar to the queries in section 5.3.3 'Using Title Queries'. The result of this path is followed in flow chart V.

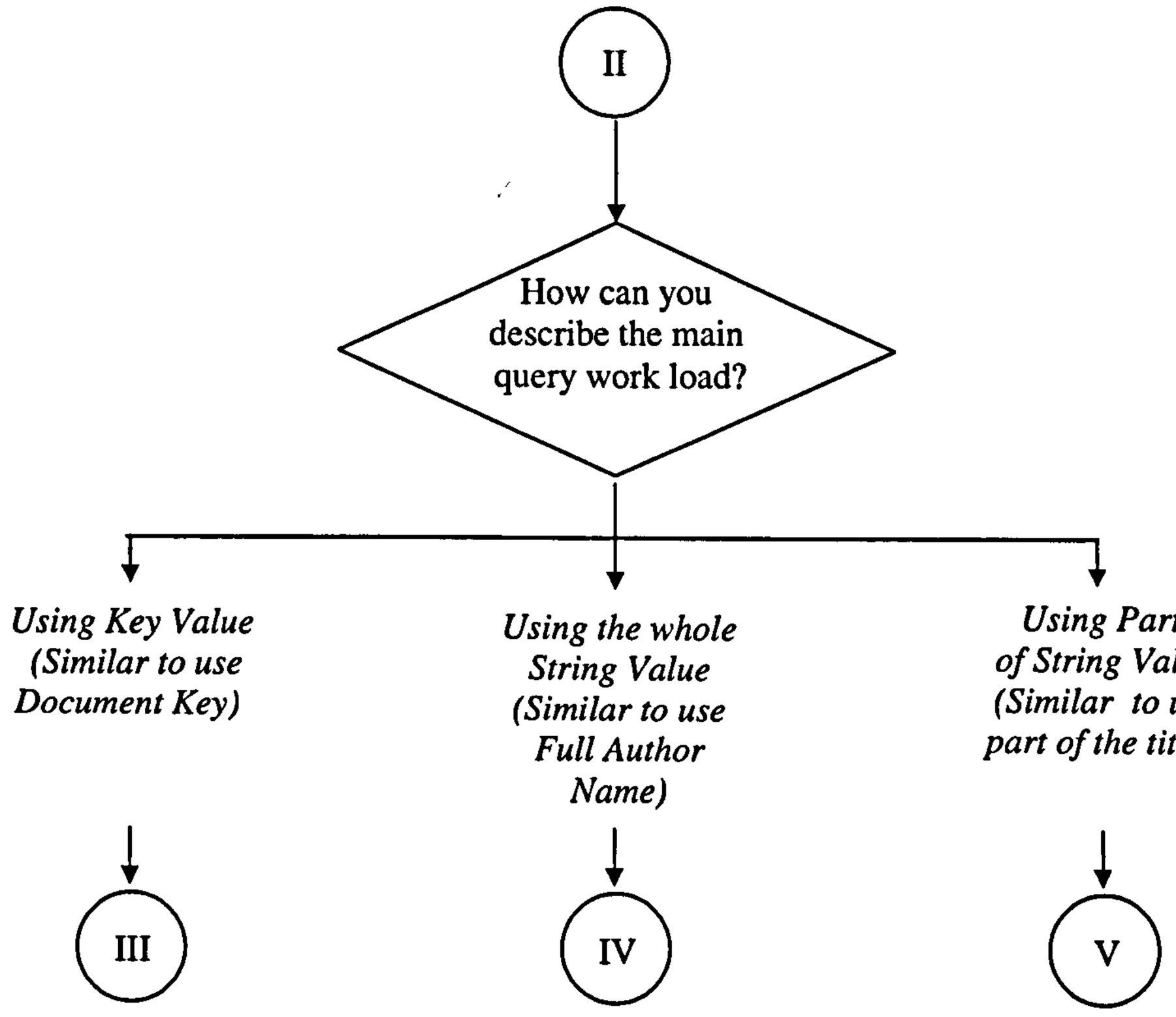


Figure 5.43 Flow Chart II

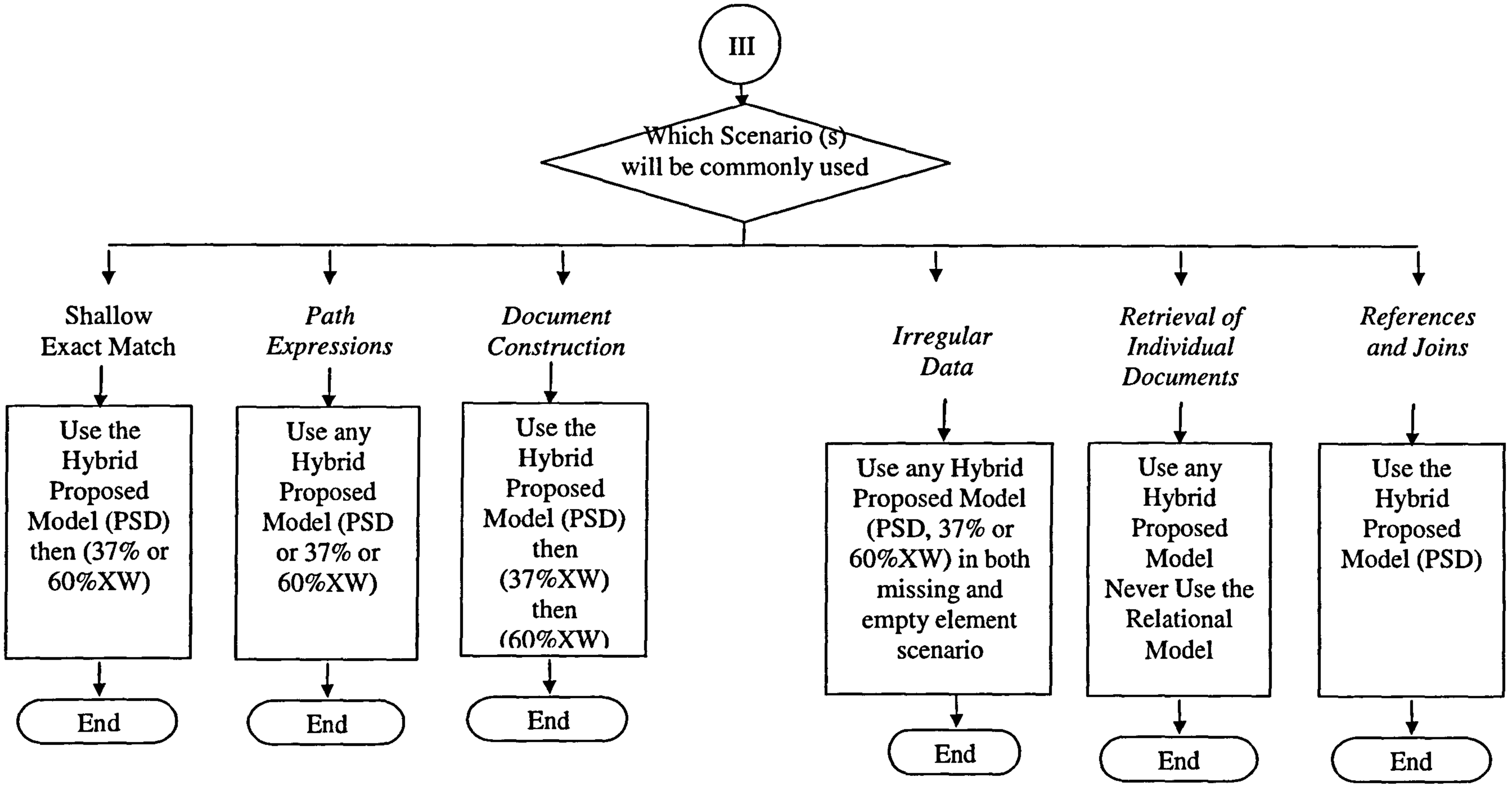


Figure 5.44 Flow Chart III

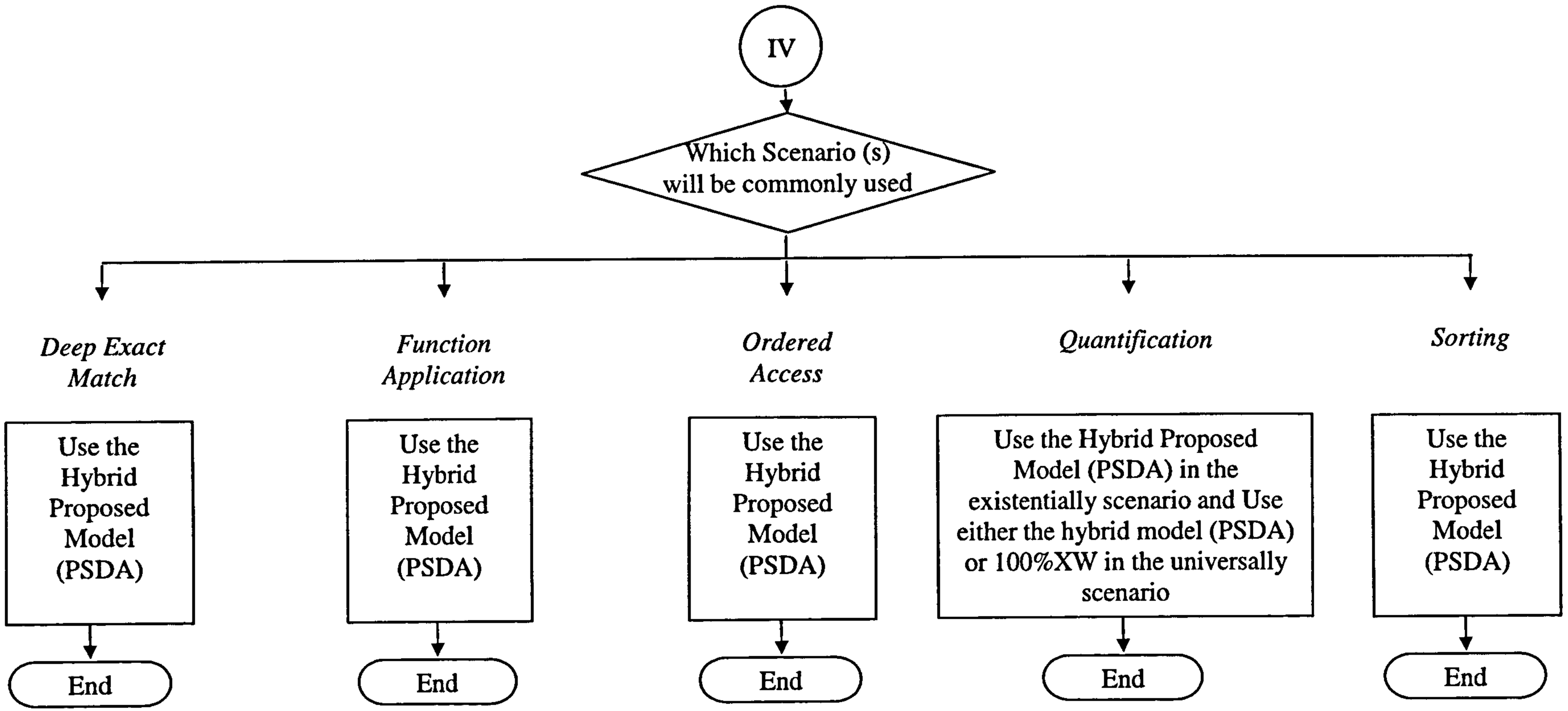


Figure 5.45 Flow Chart IV

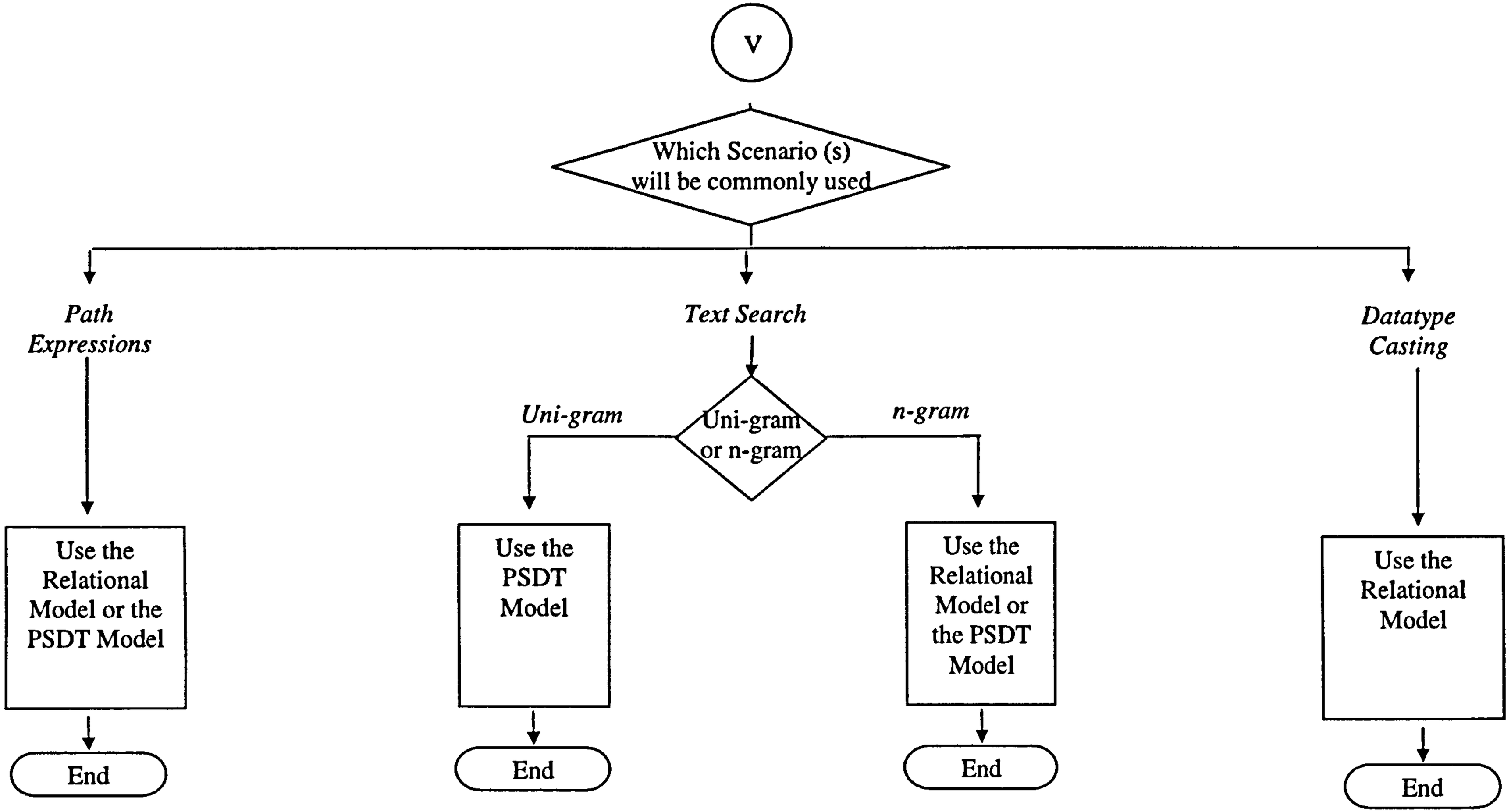


Figure 5.46 Flow Chart V

In the last three flow charts (III, IV and V), the basic question is to identify the most commonly used scenario (or scenarios) used to describe in detail the query work load. Flow chart III is based on section 5.3.1 where the most common scenario to query the data was using the key value. There were six different sub-scenarios as follows: 5.3.1.1 shallow exact match, 5.3.1.2 path expressions, 5.3.1.3 document construction, 5.3.1.4 irregular data, 5.3.1.5 retrieval of individual documents and 5.3.1.6 References and joins. Each scenario was described in more detail in its relevant section. The outcome will define the best possible storage scenario (or scenarios) that can be used in this case. Flow chart IV is based on section 5.3.2, where it is expected that the most common scenario to query the data will use a string value. There were five different sub-scenarios as follows: 5.3.2.1 deep exact match, 5.3.2.2 function application, 5.3.2.3 ordered access, 5.3.2.4 quantification and 5.3.2.5 sorting. Finally, flow chart V is based on section 5.3.3 as the most common scenario to query the data using a part of a string value, there were three different sub-scenarios as follows: 5.3.3.1 path expressions, 5.3.3.2 text search and 5.3.3.3 datatype casting.

In this chapter, the results of the experiments were discussed and analysed. The experiments' results were grouped by the different query functionalities. This was followed by an overall analysis of the different storage strategies with respect to the different variants the experiments were designed to measure. The experiments' limitations and general findings were finally presented. The next chapter concludes this thesis by discussing the main findings and contributions of this research in general and presents the future research work envisaged to further develop this research.

Chapter 6 Conclusion and Future Work

6.1 Introduction

The research presented in this thesis is concerned with seeking better ways to store and query XML data using relational database technology. Specifically, the research considers a class of XML data which is partly structured and partly semi-structured, referred to as partially structured data. The aim is to establish for this class of XML data, whether pre-knowledge of the structured component can be exploited when storing the data within a relational database, such that query processing efficiency can be improved by exploiting relational query processing and optimisation technology, while at the same time, providing a flexible way to store the semi-structured part. To that end, the research proposed a hybrid XML–Relational model to store and query partially-structured XML documents.

This chapter concludes the thesis by discussing two final and important points. Section 6.2 discusses the main findings and contributions of this research and section 6.3 proposes future research work to further develop this research.

6.2 Main Findings and Contributions

This research contributes to the ever evolving field of XML database management. In the existing literature, there have been many examples for different storage models to manage XML-encoded data. However, the main research streams dealt with highly-structured XML data or semi-structured XML data. This research has contributed in regard to this body of literature by addressing XML documents which combine both highly-structured and semi-structured data, defined as partially-structured XML data. In this section, the main findings and contributions are discussed in detail.

The research has contributed by proposing a hybrid relational-XML storage model to store partially-structured XML encoded data, in which a combination of structure mapping and XML types are used within a relational database, so as to exploit pre-knowledge of the structured part in query processing. In this hybrid approach, an information-preserving mapping to a relational schema is defined for the highly-structured parts of a document, such that conventional relational optimisation techniques can be applied when querying this data. The semi-structured parts are stored

as instances of the XML data type, so as to provide a flexible way to manage and manipulate this part of the document using regular-expression based querying (see section 3.6).

The main contribution of this research is the performance analysis of the above proposed hybrid relational-XML model for storing and querying partially structured data, based on a standard benchmark set of queries (XBench, Yao et al. 2002, 2003, 2004), which establishes the impact on query performance as the structuredness, database size and the different query characteristics. The results of the experiments showed that there was no single storage model that outperforms all other models' performance in all different scenarios. In most of the cases, the hybrid models performed better than the relational and XML data type models (see section 5.4).

Another contribution of this research is to explore two different dimensions to vary the ratio of semi-structured to highly-structured parts inside a partially-structured XML document. The first dimension, which is called the vertical dimension because of the conventional tabular representation of data in which schema elements and their instances are denoted as columns, concerns the ratio of semi-structured to structured components of the schema. This dimension can be seen as a schema dimension. The second, which can be called the horizontal dimension, is the ratio of semi-structured to structured data instances. This dimension can be seen as the data instances dimension (see section 4.4.1). Varying these two dimensions can lead to a better storage model to store and query partially-structured data.

As result of the analysis described above, a heuristic method has been developed, by which the results of the performance analysis can be utilised by the database designer to seek optimal relational storage models for XML-encoded partially-structured data. This heuristic method has been translated into a flow chart which showed the potential application of the experiments' results. It can be used as an evidence-based method to guide the designer of the relationally represented XML databases (see section 5.6).

6.3 Future Research Work

In terms of future research work, there are certain aspects on which further research can build on. These can be defined in two categorizes: to address general limitations of the research and to address the limitations of the experiment design. Both are discussed below.

6.3.1 Future Work Related to the Research Limitation

The future work building on the research presented in this thesis can be summarised as follows:

- Investigate the other storing systems (such as the native XML database management system, Vectorizing approach (Buneman et al. 2005) ...etc.) and incorporate their features into the current proposed model. As these storing systems are evolving, they could lead to a good performance to the general problem of manipulating XML data.

- Investigate the possibility of automating the adjustment of the different ratio of semi-structured and highly-structured based on the query work load and the system performance. This could lead to an intelligent system that could adapt the ratio between semi-structured and highly-structured data inside the database to achieve better performance. This could be achieved by varying both the vertical and horizontal dimensions based on the query work load, the characteristics of the data and the data distribution.

- Applying multi-thread experiment testing in a network environment compared to a single thread testing in a stand alone machine. This could affect the way the data is modelled and stored. For example, dedicating one thread to deal with highly-structured data while the other to semi-structured data. Using the same model in network environment could lead to different results based on different query work load.

- Building an expert system that utilizes the results and scaling the different output and producing a compromise or a combined storage model to produce the best storage model based on the data and query work load information. This could lead to a further developed heuristic model towards helping the data base designer to design his storage model. This system could weight the designer input, for example in 80% of the cases the data will be searched by using the document key, while in the remaining 20% of the cases the data will be searched by using the author name.

6.3.2 Future Work Related to the Design of the Experiments

The limitations of the experiments were discussed in more detail in section 5.5. To address these limitations, the future work suggested in this direction can be summarised as follows:

- Use more than one data set to test the proposed system instead of using one data set to deliver the verdict. This will give wider and more diverse results that could lead to different findings based on different scenarios. Especially, it would be useful to explore more partially-structured data sets where the boundary between the highly-structured and the semi-structured is much more defined.
- Calibrating the proposed model by using different database management systems. In such case, the results from one DBMS can be compared to the other DBMS. This will generalize the findings. At present, they are restricted to a specific DBMS (MS SQL Server).
- Use other different intermediate percentages in partitioning the data in the horizontal dimension (as 60% and 37% are specific to the DBLP data set). This can be done when running the experiment on different data set or by engineering the percentage for the same data set (for example by deleting potion of the data to create a different data instance percentages).
- Re-design the query set to test the highly-structured part of the data and/or a combination of highly-structured and semi-structured part of the data instead of testing only the semi-structured part of the data.
- Use a data set where the two un-tested groups in the XML benchmark (irregular data and references and join) can be tested. This limitation was specific for the DBLP data set used in the experiments. This can lead to using multiple document scenarios specifically to test the reference and join query group.
- Test the scalability from the complexity point of view. This adds a new dimension to the analysis when testing the complexity of different data sets for the same query group.

6.4 Final Remarks

This thesis has presented a performance analysis of storage strategies for representing partially-structured data within a relational database. The hybrid approach, in which storage mapping is combined with the use of XML data type instances, was shown to have query performance advantages over pure structure mapping and sole use of XML data types. However, the results are inconclusive, since they identify a number of anomalies, problems of scaling these approaches to large data sets, and the existence of thresholds where the cost of data shredding appears to outweigh the advantages of utilizing relational query processing. Each of these is a motivation for further research. Also, the experiments described are confined to a certain type of data, but nonetheless they provided valuable insights into the relative performance of different storage models as has been discussed in the body of this thesis.

The main contributions of this research were in the analysis of relative performances within a specific configuration, rather than across systems, as in other performance studies. The research also contributes by focusing on partially-structured data and the impact of the horizontal and vertical dimensions of data structuredness, which is under researched in the field of XML database management. Finally, a heuristics-based model was devised to inform XML/relational design.

References

Abiteboul, S. (1997) Querying Semi Structured Data. *In Proceedings of International Conference on Database Theory ICDT*, p. 1 -- 18.

Abiteboul, S. (2001) Semistructured Data: from Practice to Theory. *In Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science LICS*, IEEE Computer Society, Washington, DC, United States of America, p. 379.

Abiteboul, S. and Vianu, V. (1997) Queries and Computation on the Web. *International Conference on Database Theory ICDT*, p. 262 -- 275.

Abiteboul, S., Buneman, P. and Suciu, D. (1999) *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann, ISBN:1-55860-622-X.

Abiteboul, S., Cluet, S. and Milo, T. (1993) Querying and updating the file. *In Proceedings of 19th International Conference on Very Large Databases*, Dublin, Ireland, p. 73 -- 84.

Abiteboul, S., Quass, D., McHugh, J., Widom, J. and Wiener, J. (1997). The Lorel Query Language for Semistrucutred data. *International Journal on Digital Libraries*, Volume 1, Issue 1, p. 68 -- 88.

Al-Wasil, F., Fiddian, N. J. and Gray, W. (2006) Query Translation for Distributed Heterogeneous Structured and Semi-structured Databases. *British National Conference on Databases BNCOD*, Belfast, p. 73 -- 85.

Al-Wasil, F., Gray, W. A., and Fiddian, N. J. (2006a). Establishing an XML metadata knowledge base to assist integration of structured and semi-structured databases. In Proceedings of the 17th Australasian Database Conference. Volume 49. Hobart, Australia, p. 69 -- 78.

Amer-Yahia, S. and Srivastava, D. (2002) A Mapping Schema and Interface for XML Stores. In *Proceedings of the 4th international Workshop on Web information and Data Management WIDM*, Mclean, Virginia, United State of America.

Amer-Yahia, S., Du, F., and Freire, J. (2004). A Comprehensive Solution to the XML-to-Relational Mapping Problem. *In Proceedings of the 6th Annual ACM*

international Workshop on Web information and Data Management, WIDM. Washington DC, United State of America, ACM Press, New York, NY, p. 31--38.

Astoria - Astoria Software [online]. Available form:

http://www.astoriasoftware.com/ [Accessed 10.06.2006]

Atay, M., Chebotko, A., Liu, D., Lu, S. and Fotouhi, F. (2007). Efficient schema-based XML-to-Relational data mapping. Information Systems, Volume 32 (3), p. 458 - 476.

Atay, M., Liu, D., Sun, Y., Lu, S. and Fotouhi, F. (2004). Mapping XML data to relational data: A DOM-based approach. *In 8th IASTED International Conference on Internet and Multimedia Systems and Applications*, Kauai, Hawaii, p. 59 -- 64.

Balmin, A and Papakonstantinou, Y (2005) Storing and Querying XML data using denormalized relational databases *The VLDB Journal,* Volume 14, p. 30 -- 49.

Barbosa, D., Barta, A., Mendelzon, A., Mihaila, G., Rizzolo, F. and Rodriguez-Gianolli, P. (2001). ToX: The Toronto XML Engine. *In Proceedings of the Workshop on Information Integration on the Web*, Rio de Janeiro, Argentina.

Barbosa, D., Mendelzon, A., Keenleyside, J. and Lyons, K. (2002). ToXgene: A Template-Based Data Generator for XML. *In Proceedings of the 5th International Workshop on the World Wide Web and Databases (WebDB)*, p. 49--54.

Barbosa, D., Mignet, L., and Veltri, P. (2006). Studying the XML Web: Gathering Statistics from an XML Sample. *World Wide Web*, Volume 9, Issue 2, p. 187 -- 212.

Batory, D. (1979). On searching transposed files. *ACM Transactions on Database Systems TODS*, Volume 4, Issue 4, p. 531 -- 544.

Beyer, K., Cochrane, R. J., Josifovski, V., Kleewein, J., Lapis, G., Lohman, G., Lyle, B., Özcan, F., Pirahesh, H., Seemann, N., Truong, T., Van der Linden, B., Vickery, B., and Zhang, C. (2005). System RX: one part relational, one part XML. In *Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data,* Baltimore, Maryland, United State of America.

BizTalk web site [online]. Available form:

http://www.microsoft.com/biztalk/default.mspx [Accessed 30.05.2006]

Bohannon, P., Freire, J. Haritsa, J., Ramanath, M., Roy, P. and Simon, J. (2002) LegoDB: Customizing Relational Storage for XML Documents. *Proceedings of the 28th VLDB Conference*, Hong Kong, China.

Böhme, T. and Rahm, E. (2001). XMach-1: A benchmark for XML data management. *In Proceedings of Data Bank Systeme*, Technik und Wissenschaft (BTW). Oldenburg, Germany, p. 264 -- 273.

Böhme, T. and Rahm, E. (2002) Multi-user Evaluation of XML Data Management Systems with XMach-1. *Efficiency and Effectiveness of XML Tools, and Techniques EEXTT*, p. 148 -- 158.

Bourret, R. [online] (2005) XML and Databases. Available from: http://www.rpbourret.com/xml/XMLAndDatabases.htm

Brassan, S., Lee, M., Li, Y., Lacroix, Z. and Nambiar, U. (2002) The XOO7 benchmark. *In Proceedings of Very Large Data Bases 2002 Workshop Efficiency and Effectiveness of XML Tools, and Techniques EEXTT*, Lecture Notes in Computer Science Volume 2590, p. 146 -- 147.

Buneman, P. (1997) Semistructured Data. *Symposium On Principles Of Database Systems PODS*, p. 117 – 121

Buneman, P. et al (2005) Vectorizing and Querying Large XML Repositories. *International Conference on Data Engineering, ICDE*, Tokyo, Japan.

Buneman, P., Davidson, S., Fernandez, M. and Sucui, D. (1997) Adding Structure to unstructured data. *In proceeding of the international conference on Database Theory*, Deplhi, Springer Verlag, p. 336 -- 350.

Buneman, P., Davidson, S., Hillebrand, G. and Suciu, D. (1996) A query language and optimization techniques for unstructured data. *In Proceedings of ACM-SIGMOD International Conference on Management of Data*, Montreal, Canada, p. 505 -- 516.

Buneman, P., Fernandez, M. and Suciu, D. (2000) UnQL: A query language and algebra for semistructured data based on structural recursion. *VLDB Journal.*

Carey, M., DeWitt, D. and Naughton, J. (1993) The OO7 benchmark. *In ACM*

SIGMOD Conference. p. 12 -- 21.

Cattell, R. G. G., Barry, D. K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T. and Velez, F. (2000) *The Object Data Standard ODMG 3.0*. Morgan Kaufmann, ISBN: 1558606475.

Chamberlin, D., Robie, J. and Florescu, D. (2000) Quilt: An XML Query Language for Heterogeneous Data Sources. *International Workshop on the Web and Databases (WebDB'2000)*, Dallas, United State of America.

Chaudhuri, S, Chen, Z, Shim, K and Wu, Y (2005) Storing XML (with XSD) in SQL Databases: Interplay of Logical and Physical Designs. *IEEE transactions on knowledge and data engineering*, volume 17 (12).

Chawathe, S., Garcia-Molina, G., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. and Widom, J. (1994). The TSIMMIS Project: Integration of Heterogeneous Information Sources. *In Proceedings of 10th Meeting of the Information Processing Society Conference*, Tokyo, Japan, p. 7 -- 18.

Christophides, V., Abiteboul, S., Cluet, S., and Scholl, M. (1994) From structured documents to novel query facilities. *SIGMOD Record*, Volume 23 (2), p. 313 -- 324.

Chung, T et al (2001) Extracting Object-Oriented Database Schemas from XML DTDs Using Inheritance, *Proceedings 2nd International Conference EC-Web.*

Chung, T. and Kim, H. (2003). Techniques for the evaluation of XML queries: a survey. *Data Knowledge Engineering*, Volume 46 (2), p. 225 -- 246.

Clark, J. (1997) Comparison of SGML and XML World Wide Web Consortium Note 15-December-1997 [online]. Available from: http://www.w3.org/TR/NOTE-sgml-xml-971215 [Accessed 01.05.2006]

Cluet, S., Delobel, C., Siméon, J. and Smaga, K. (1998) Your mediators need data conversion! *In Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, Seattle, Washington, United States of America, p. 177 -- 188.

Cooper, B., Sample, N., Franklin, M., Hjaltason, G.R. and Shadmon, N. (2001) A fast index for semistructured data. *Proceedings of the 27th International Conference on Very Large Data Bases*. Rome, Italy, Morgan Kaufmann, Los Altos, CA, p. 341 --

350.

Deutsch, A., Fernandez, M. and Suciu, D. (1999). Storing semistructured data with STORED. *In Proceedings of the 25th ACM SIGMOD International Conference on Management of Data.*

Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D. (1998) XML-QL: A Query Language for XML. World Wide Web Consortium Note NOTE-xml-ql-19980819. [online]. Available from: http://www.w3.org/TR/NOTE-xml-ql/ [Accessed 1.10.2004].

DeWitt, D. (1993) The Wisconsin Benchmark: Past, Present, and Future. *The Benchmark Handbook for Database and Transaction Systems (2nd Edition).* Morgan Kaufmann, ISBN 1-55860-292-5.

Document Object Model (DOM) Level 3 Core Specification Version 1.0 W3C Recommendation 07 April 2004 Available form: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/ [Accessed 10.06.2006]

Eisenberg, A. and Melton, J. (2004) Advancements in SQL/XML. *SIGMOD Record 33*. Volume (3), p. 79--86.

Elmacioglu, E. and Lee, D. (2005). On six degrees of separation in DBLP-DB and more. *SIGMOD Record*. Volume 34 (2), p. 33 -- 40.

Elmasri, R. and Navathe, S. (2006) *Fundamentals of Database Systems*, 5nd Edition, Pearson/Addison-Wesley, ISBN 032141506X.

Fernandez M., Florescu D., Levy A., Suciu D. (1997) A Query Language for a Web-Site Management System. *SIGMOD Record*, Volume 26 (3), p. 4 -- 11.

Fernandez, M., Florescu, D., Levy, A. and Suciu, D. (1998) Web-Site Management: The Strudel Approach. *IEEE Data Engineering*. Bull. Volume 21 (2), p. 14 -- 20.

Fernandez, M., Florescu, D., Levy, A., Suciu, D. (2000) Declarative Specification of Web Sites with Strudel. *VLDB Journal*, Volume 9 (1), p. 38--55

Fernandez, M., Simeon, J., Wadler, P. (eds.), Cluet, S., Deutsch, A., Levy, A., Maier, D., McHugh, J., Robie, J., Suciu, D. and Widom, J. (1999) XML Query Languages Experiences and Exemplars. *Draft manuscript, communication to the XML*

Query W3C Working Group http://www.w3.org/1999/ 09/ql/docs/xquery.html.

Fiebig, T. , Helmer, S., Kanne, C., Mildenberger, J., Moerkotte, G., Schiele, R. and Westmann, T. (2002) Anatomy of a native xml base management system. *Technical Report 01,* University of Mannheim, Germany.

Florescu, D. (2005) Managing semi-structured data. *Queue 3, 8.* www.acmqueue.com. p. 18 -- 24.

Florescu, D. and Kossmann, D. (1999) Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin* Volume 22 (3), p. 27 -- 34.

Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. and Widom, J. (1995). Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. *In Proceedings of the AAAI Symposium on Information Gathering*, Stanford, California, United State of America, p. 61 -- 64.

Garcia-Molina, H., Quass, D., Papakonstantinou, Y., Rajaraman, A., Sagiv, Y., Ullman, J. and Widom, J. (1995a) The TSIMMIS Approach to Mediation: Data Models and Languages. *Next Generation Information Technologies and Systems* (NGITS '95), Second International Workshop, Naharia, Israel.

Goldfarb, C.F. (1990) *The SGML Handbook*. Oxford University Press, Oxford, United Kingdom.

Goldman, R. and Widom, J. (1997) DataGuides: Enabling query formulation and optimization in semistructured databases. *In Proceedings of the Twenty-Third International Conference on Very Large Databases*, Athens, Greece, p 436 -- 445.

Goldman, R., McHugh, J. and Widom, J. (1999). From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania, United State of America.

Grossman, D., Holmes, D. Frieder, O. and Roberts, D. (1997) Integrating Structured Data and Text: A Relational Approach. *Journal of the American Society of Information Science.*

Hammer, J., Garcia-Molina, H., Ireland, K., Papakonstantinou, Y., Ullman, J. and Widom, J. (1995). Information Translation, Mediation, and Mosaic-Based

Browsing in the TSIMMIS System. *In Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, California, United States of America, p. 483.

Hammer, J., McHugh, J. and Garcia-Molina, H. (1997) Semistructured Data: The Tsimmis Experience. *Proceedings of the First East-European Symposium on Advances in Databases and Information Systems* (ADBIS'97), St.-Petersburg, p. 1 -- 8.

Han, W., Lee, K. and Lee, B. (2003) An XML Storage System for Object-Oriented/Object-Relational DBMSs. Journal of Object Technology, Volume 2 (3).

Harding, P., Li, Q. and Moon, B. (2003) XISS/R: XML Indexing and Storage System Using RDBMS. *Proceedings of the 29th VLDB Conference*, Berlin, Germany.

HTML - Hypertext Markup Language [online]. Available from: http://www.w3c.org/MarkUp [Accessed 1.10.2002].

Information Manager - Standard-Based Content Management Interleaf [online]. Available form: http://www.interleaf.com/ [Accessed 10.06.2006]

Khan, L. and Rao, Y. (2001) A Performance Evaluation of Storing XML data in Relational Database Management *Systems. In proceedings of the international workshop on the web information and data management*, Atlanta, Georgia, United States of America.

Khan, L., Chen, Q. and Rao, Y. (2002) A Comparative Study of Storing XML data in Relational and Object-Relational Database Management Systems. *In proceedings of the international conference on internet computing*, Las Vegas, Nevada, United States of America, p 277 -- 282.

Kim, S., Shin, P., Kim, Y., Lee, J. and Lim, H. (2002) A Data Model and Algebra for Document-Centric XML Document. *Information Networking, Wireless Communications Technologies and Network Applications, International Conference.* Cheju Island, Korea, Volume 2, p. 714 -- 723

Klettke, M. and Mayer, H. (2001) XML and Object-Relational Database Systems Enhancing Structural Mappings Based on Statistics *WebDB*, Lecture Notes in Computer Science 1997, Springer-Verlag, p. 151 -- 170.

Krishnaprasad, M. Liu, Z., Manikutty, A., Warner,J. and Arora, V. (2005)

Towards an Industrial Strength SQL/XML Infrastructure. *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005*. Tokyo, Japan, p. 991--1000.

Kuckelberg, A. and Krieger, R. (2003) Efficient Structure Oriented Storage of XML Documents Using ORDMS. *In Proceedings of Very Large Data Bases 2002 Workshop EEXTT and DIWEB*, LNCS 2590, p. 131--142.

Kudrass, T. (2002) Management of XML Documents without schema in relational database systems *Information and Software Technology*, Volume 44, p. 269--275.

Kudrass, T. and Conrad, M. (2002) Management of XML Documents in Object-Relational Databases *EDBT 2002 workshops*, Lecture Notes in Computer Science 2490, Springer-Verlag, p. 210--227.

Lacoude, P. (2006) Pushing SQL Server 2005 Limits, Dealing with Oversized XML Documents [online] Available from:

http://www.lacoude.com/Docs/public/public.aspx?doc=SQL90XML.PDF [Accessed 25.09.2006]

Lahiri, T., Abiteboul, S. and Widom, J. (1999) Ozone: Integrating Structured and Semistructured Data. *In Proceedings of the Seventh International Conference on Database Programming Languages*, Kinloch Rannoch, Scotland.

Leavitt, N. (2000) Whatever Happened to Object-Oriented Databases? *IEEE Computer* Volume 33 , p. 16—19

Leonov, A. V. and Khusnutdinov, R. R. (2004) Construction of an Optimal Relational Schema for Storing XML Documents in an RDBMS without Using DTD/XML Schema *Programming and Computer Software*, Volume 30 (6), p. 323--336.

Ley, M. (2002) The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. *SPIRE*, p. 1-10

Ley, M. and Reuther, P. (2006) Maintaining an Online Bibliographical Database: The Problem of Data Quality. *Extraction et gestion des connaissances EGC 2006*, Lille, France, p.5--10.

Low, W., Tok, W., Lee, M. and Ling, T. (2002) Data Cleaning and XML: The DBLP Experience. *ICDE 2002*, p. 269

Lu, E. J., Wu, B., and Chuang, P. (2006) An empirical study of XML data management in business information system. *The Journal of Systems and Software* Volume (79), p. 984 -- 1000

Lu, H., Xu Yu, J., Wang, G., Zheng, S., Jiang, H., Yu, G. and Zhou, A. (2005) What Makes the Differences: Benchmarking XML Database Implementations. *ACM Transactions on Internet Technology* (ACM TOIT), Volume 5 (1), p. 154 -- 194.

Lu, S., Sun, Y., Atay, M. and Fotouhi, F. (2003) A new inlining algorithm for mapping XML DTDs to relational schemas. *In Proceedings of the 1st International Workshop on XML Schema and Data Management.* Lecture Notes in Computer Science, Chicago, Illinois, United States of America.

Lv, T. and Yan, P. (2006) Mapping DTDs to relational schemas with semantic constraints. *Information and Software Technology*, Volume 48 (4), p. 245--252

Madria, S., Chen, Y., Passi, K. and Bhowmick, S. (2007) Efficient processing of XPath queries using indexes. *Information Systems,* Volume 32 (1), p. 131 – 159.

McHugh, J. and Widom, J. (1999) Query Optimization for XML. *In Proceedings 25th International Conference on Very Large Databases*, Morgan Kaufmann, p. 315--326.

McHugh, J., Abiteboul, S., Goldman, R., Quass, D. and Widom, J. (1997). Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, Volume 26 (3), p 54 -- 66.

Microsoft SQL Server Database Management System [online]. Available from: http://www.microsoft.com/sql/default.mspx [Accessed 1.7.2005]

Milo, T. and Suciu, D. (1999) Index Structures for Path Expressions. *International Conference on Database Theory ICDT*, p. 277 -- 295

Murthy, R., Liu, Z. H., Krishnaprasad, M., Chandrasekar, S., Tran, A., Sedlar, E., Florescu, D., Kotsovolos, S., Agarwal, N., Arora, V., and Krishnamurthy, V. (2005). Towards an enterprise XML architecture. *In Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data,* Baltimore, Maryland,

United States of America, p. 14--16.

Na, G. and Lee, S. (2005) A Relational Nested Interval Encoding Scheme for XML Storage and Retrieval Information retrieval technology. *Second Asia information retrieval symposium, AIRS* , Jeju Island, Korea, LNCS 2689, p. 715 -- 720.

Namespaces in XML World Wide Web Consortium 14-January-1999 [online]. Available from: http://www.w3.org/TR/REC-xml-names/ [Accessed 22.05.2006]

Nestorov, S., Abiteboul, S., and Motwani, R. (1997). Inferring structure in semistructured data. *SIGMOD Record*, Volume 26 (4), p. 39 -- 43.

Neven, F., Schwentick, T., Suciu, D. (Eds.) (2005) Foundations of Semistructured Data. *Dagstuhl Seminar Proceedings 05061* Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany/IBFI, Schloss Dagstuhl, Germany.

Novak, L. and Zamulin, A. (2005) Algebraic Semantics of XML Schema. *Proceding ADBIS 2005*, LNCS, Volume 3631, p. 209 -- 222.

Novak, L. and Zamulin, A. (2006) An XML Algebra for XQuery. *ADBIS 2006*, p. 4 -- 21.

ORACLE Database 10G Release 2 XML DB & XML DB Repository [online]. Available from:

http://www.oracle.com/technology/tech/xml/xmldb/Current/xmldb_datasheet.pdf [Accessed 15.06.2006].

Oracle Database Management System [online]. Available from: http://www.oracle.com/index.html [Accessed 1.10.2003]

Özcan, F., Chamberlin, D., Kulkarni, K., and Michels, J. (2006). Integration of SQL and XQuery in IBM DB2. *IBM System Journal.* Volume 45 (2), p. 245 -- 270.

Pal, S., Cseri, I., Seeliger, O., Rys, M., Schaller, G., Yu, W., Tomic, D., Baras, A., Berg, B., Churin, D., and Kogan, E. (2005). XQuery implementation in a relational database system. *In Proceedings of the 31st international Conference on Very Large Data Bases.* Trondheim, Norway, p. 1175--1186.

Pal, S., Cseri, I., Seeliger, O., Schaller, G., Giakoumakis, L. and Zolotov, V. (2004) Indexing XML Data Stored in a Relational Database. Proceeding of the 30th

VLDB conference, Toronto, Canada.

Pal, S., Tomic, D., Berg, B. and Xavier, J. (2006) Managing Collections of XML Schemas in Microsoft SQL Server 2005. *EDBT 2006*, p. 1102--1105.

Papakonstantinou, Y., Abiteboul, S. and Garcia-Molina, G. (1996). Object fusion in mediator systems. *In Proceedings of the International Conference on Very Large Data Bases* (VLDB), Bombay, India.

Papakonstantinou, Y., Garcia-Molina, H. and Widom, J. (1995) Object exchange across heterogeneous information sources. *In Proceedings of the Eleventh International Conference on Data Engineering*, Taipei, Taiwan, IEEE Computer Society Press, p. 251 -- 260.

Paparizos, S and Jagadish, H.V. (2006) The Importance of Algebra for XML Query Processing. *In EDBT 2006 Workshop on XML Data Management*, Munich, Germany.

Paparizos, S. et al (2003) TIMBER: A Native System for Querying XML. *SIGMOD Conference*, Volume 2003, p. 672.

Paparizos, S., Al-Khalifa, S., Jagadish, H. V., Niermann, A. and Wu, Y. (2002). A physical algebra for XML. *Technical report*, University of Michigan.

Pardede, E., Rahayu, J. and Taniar, D. (2005) Preserving Composition in XML Object Relational Storage. *19th International Conference on Advanced Information Networking and Applications AINA*, Fukuoka, Japan, p. 695 -- 700.

Pardede, E., Rahayu, J. and Taniar, D. (2006) Object-relational complex structures for XML storage. *Information & Software Technology* Volume 48 (6), p. 370 -- 384

Penna, G;, Marco, A., Intrigila, B., Melatti, I. and Poerantonio, A. (2006) Interoperability mapping from XML schemas to ER diagrams. *Data & Knowledge Engineering*, Volume 59 (1), p. 166 -- 188.

Psaila , G (2002) ERX: An Experience in Integrating Entity-Relationship Models, Relational Databases, and XML Technologies. *EDBT Workshops* Lecture Notes in Computer Science 2490, Springer-Verlag, p. 242 -- 265.

Psaila, G (2003) From XML DTDs to Entity-Relationship Schemas. *ER*

Workshops, Chicago, IL, United States of America, p. 378 -- 389.

Qin, J., Zhao, S., Yang, S. and Dou, W. (2005) XPEV: A Storage Model for Well-Formed XML Documents. *FSKD,* Volume *1*, p. 360 -- 369

Reuther, P. Walter, B., Ley, M., Weber, A. and Klink, S. (2006) Managing the Quality of Person Names in DBLP. *Research and Advanced Technology for Digital Libraries, 10th European Conference, ECDL 2006*, Alicante, Spain, p. 508 -- 511.

Rizzolo, F. and Mendelzon, A. (2001) Indexing XML Data with ToXin. *WebDB 2001*, p. 49 -- 54

Robie, J., Lapp, J. And Schach, D. (1998) XML Query Language (XQL) [online]. Available from http://www.w3.org/TandS/QL/QL98/pp/xql.html [Accessed 22.05.2006].

Runapongsa, K, Patel, J. M. (2002) Storing and Querying XML Data in Object-Relational DBMSs. *EDBT 2002 Workshop on XML-Based Data Management (XMLDM'02)*, Prague, Czech Republic.

Runapongsa, K., Jignesh M. Patel, J. M. and Al-Khalifa, S. (2002a). The Michigan benchmark: A microbenchmark for XML query processing systems. *In Proceedings of Very Large Data Bases 2002 Workshop EEXTT*. Lecture Notes in Computer Science Volume 2590, p. 160 -- 161.

Runapongsa, K., Patel, J. M., Jagadish, H. V., Chen, Y. and Al-Khalifa, S. (2006). The Michigan benchmark: towards XML query performance diagnostics. *Information Systems*, Volume 31 (2), p. 73 -- 97.

Rys, M. (2005). XML and relational database management systems: inside Microsoft® SQL Server™ 2005. *In Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data* Baltimore, Maryland, United States of America, p. 14--16.

Rys, M., Chamberlin, D. and Florescu, D. (2005). XML and relational database management systems: the inside story. *In Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data*. Baltimore, Maryland. SIGMOD '05. ACM Press, New York, NY, United States of America, p. 945 -- 947.

Salminen, A. and Tompa, F. (2001): Requirements for XML document database

systems. *ACM Symposium on Document Engineering.* p. 85--94.

SAX [online]. Available from: http://www.saxproject.org/ [Accessed 18.05.2006]

Schmidt, A. R., Kersten, M. L., Windhouwer, M. A., Waas, F. (2000) Efficient Relational Storage and Retrieval of XML Documents. *Workshop on the Web and Databases (WebDB)*, Dallas, United States of America.

Schmidt, A., A., Waas, F., Kersten, M., Carey, M. J., Manolescu, I. And Busse, R. (2002). XMark: A benchmark for XML data management. *In Proceedings of the 28th International Conference on Very Large Data Bases.* Hong Kong, China, p. 974 - - 985.

Schoning, H. and Wasch, J. (2000). Tamino - An Internet Database System" in Advances in Database Technology. *EDBT 2000, 6th International Conference on Extending Database Technology.* Konstanz, Germany, Proceedings (C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, eds.), Volume·1777 of Lecture Notes in Computer Science, Springer, p. 383 -- 387.

Shanmugasundaram et al. (1999) Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proceeding of the 25th VLDB Conference*, Edinburgh, Scotland.

Shanmugasundaram, J., Shekita, E., Kiernan, J., Krishnamurthy, R., Viglas, S., Naughton, J. and Tatarinov, I. (2001) A General Techniques for Querying XML Documents using a Relational Database System. *SIGMOD Record,* Volume 30(3). p. 20 -- 26.

Shimura, T., Yoshikawa, M., Uemura, S. (1999) Storage and Retrieval of XML Documents using Object-Relational Databases. *International Conference on Database and Expert Systems Applications (DEXA).* Springer-Verlag, LNCS 1677, Florence, Italy.

Siméon, J. and Cluet, S. (1998) Using YAT to build a web server. *In International Workshop on the Web and Databases (WebDB'98)*, Volume 1590 of Lecture Notes in Computer Science, Valencia, Spain, p. 118 -- 135.

Singh, A. et al. (2005) NXS: Native XML processing in Sybase RDBMS. *ICDE Workshops 2005.* p. 1280

SQL Server 2005 XML [online]. Available from:

http://msdn.microsoft.com/sql/learning/prog/xml/default.aspx [Accessed 15.06.2006].

SQL:2003. International Organization for Standardization (ISO). *Information Technology-Database Language SQL.* Standard No. ISO/IEC 9075:2003. (Available from American National Standards Institute, New York, NY 10036, United States of America.)

SQL:2006. International Organization for Standardization (ISO). *Information Technology-Database Language SQL.* Standard No. ISO/IEC 9075-14:2006. Part 14: XML-Related Specifications (SQL/XML) (Available from American National Standards Institute, New York, NY 10036)

http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=38647&ICS1=35&ICS2=60&ICS3

Staken, K. (2001) Introduction to Native XML Database [online]. Available from: http://www.xml.com/lpt/a/2001/10/31/nativexmldb.html [Accessed 29.11.2002]

Suciu, D. (1998) Semistructured Data and XML. *In Proceedings of the international conference on Foundations of Data Organization.* p. 9 -- 30.

Suciu, D. (2002) Distributed query evaluation on semistructured data. *ACM Trans. Database System,* Volume 27 (1), p. 1 -- 62.

Tamino - Software Age [online]. Available from:

http://www.softwareag.com/corporate/products/tamino/default.asp [Accessed 08.06.2006].

TeraText DBS [online]. Available from: http://www.teratext.com.au/index.html [Accessed 12.06.2006].

The Transaction Processing Performance Council [online]. Available from: http://www.tpc.org/ [Accessed 24.11.2005]

Tian, F., DeWitt, D.J., Chen, J. & Zhang, C. (2002). The design and performance evaluation of alternative XML storage policies. *ACMSIGMOD Record*, Volume 31 (1), p. 5 -- 10.

Timber - Tree-structured native XML database Implemented at the University of Michigan by Bright [online]. Available form: http://www.eecs.umich.edu/db/timber/

[Accessed 08.06.2006]

Vianu, V. (2003) A Web Odyssey: from Codd to XML. *Symposium on Principles of Database Systems archive, proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Santa Barbara, California, United States of America.

Vianu, V. (2003a) XML: From Practice to Theory. *SBBD 2003.* p.11--25.

Vlist, E. (2002) *XML Schema*. O'Reilly. ISBN 0-596-00252-1

W3C XML web site [online]. Available form: http://www.w3.org/XML/ [Accessed 30.05.2006]

Wang, K. and Liu, H. Q. (1999) Discovering Structural Association of Semistructured Data. *IEEE Transactions on Knowledge and Data Engineering*, Volume 12 (3), p. 353 -- 371.

XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition) A Reformulation of HTML 4 in XML 1.0 W3C Recommendation 26 January 2000, revised 1 August 2002 [online]. Available from: http://www.w3.org/TR/xhtml1/#xhtml [Accessed 22.05.2006]

XML Information Set (Second Edition) W3C Recommendation 4 February 2004 Available form: http://www.w3.org/TR/xml-infoset/ [Accessed 10.06.2006]

XML Path Language (XPath) Version 1.0 [online]. Available from: http://www.w3.org/TR/xpath [Accessed 10.06.2006]

XML Query Use Cases [online] Available from: http://www.w3.org/TR/2005/WD-xquery-use-cases-20050915/ [Accessed 1.11.2006]

XML W3C Recommendation (Fourth Edition) 16 August 2006 edited in place 29 September 2006 [online]. Available form: http://www.w3.org/TR/2006/REC-xml-20060816/ [Accessed 22.01.2007]

XML W3C Working Draft 14-Nov-96 Available form:

http://www.w3.org/TR/WD-xml-961114 [Accessed 10.06.2006]

XPath - XML Path Language [online]. Available from:

http://www.w3.org/TR/xpath [Accessed 1.8.2003]

SOAP [online] Version 1.2 Part 1: Messaging Framework, W3C Recommendation 24 June 2003. Available from: http://www.w3.org/TR/soap12-part1/ [Accessed 2.1.2007]

XQuery 1.0 An XML Query Language, W3C Recommendation 23 January 2007 [online]. Available from: http://www.w3.org/TR/xquery/ [Accessed 1.3.2007]

XQuery 1.0 and XPath 2.0 Data Model (XDM) W3C Candidate Recommendation 11 July 2006 [online]. Available from: http://www.w3.org/TR/xpath-datamodel/ [Accessed 07.11.2006]

XSLT: 1999 [online]. Available form: http://www.w3.org/TR/xslt [Accessed 30.05.2006]

Yao, B., Özsu, M. and Keenleyside, J. (2002) XBench A Family of Benchmarks for XML DBMSs", *In Proceedings of EEXTT 2002 and DiWeb 2002*, Lecture Notes in Computer Science, Volume 2590, Springer-Verlag, p. 162 -- 164.

Yao, B., Özsu, M. and Keenleysidem J. (2002a) XBench A family of benchmarks for XML DBMSs. *In Proceedings of Very Large Data Bases 2002 Workshop EEXTT*. Lecture Notes in Computer Science, Volume 2590, p. 162 -- 163.

Yao, B., Özsu, M. and Keenleysidem J. (2003) XBench--A family of benchmarks for XML DBMSs. *Technical Report, CS-2002-39* University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

Yao, B., Özsu, M. and Keenleysidem J. (2004) XBench Benchmark and Performance Testing of XML DBMSs. *In Proceedings of 20th International Conference on Data Engineering*, Boston, MA, United States of America, p. 621 -- 632.

Yoshikawa, M. and Amagasa, T. (2001) Xrel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. *ACM Transaction on Internet Technology*, Volume 1 (1), p. 110--141.

Appendix A Examples of Formal XML Data Model

In this appendix, three examples show how to model XML into a formal data model.

Example 1: A Data-Centric XML document

Figure A.1 shows an example of a data-centric XML document. Then figure A.2 shows the graph representation of this XML document and finally, table A.1 shows how this document is mapped using the previous data model. This document was taken - with slight modifications - from the department of Information Studies, University of Sheffield web site (Online).

```
<University>
  <Department>
    <Name>Information Studies</Name>
    <ResearchGroup>
      <Name>Computational Informatics Research Group</Name>
      <Director>Peter Willett</Director>
      <Focus>database management systems</Focus>
      <AcademicStaff HeadofResearchArea="Barry Eaglestone">
        <StaffName>Barry Eaglestone</StaffName>
        <StaffName>Angela Lin</StaffName>
        <StaffName>Miguel Nunes</StaffName>
      </AcademicStaff>
    </ResearchGroup>
  </Department>
</University>
```

Figure A.1: Sample Data-Centric XML Document

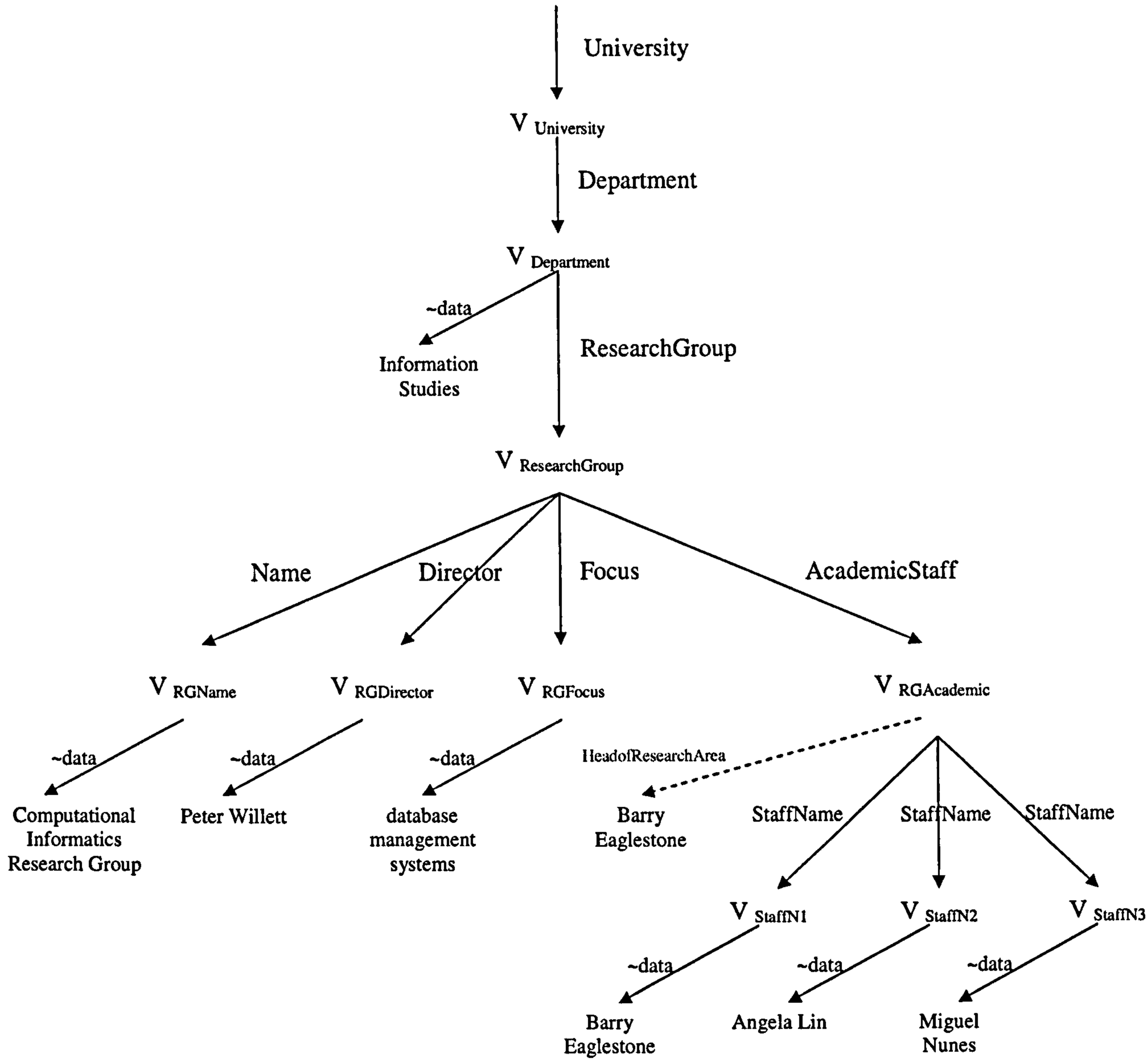


Figure A.2: Graph Representation of Data-Centric XML Document

E			
Edge	Name	Parent	Child
e_1	"University"	V_{root}	$V_{University}$
e_2	"Department"	$V_{University}$	$V_{Department}$
e_3	~data	$V_{Department}$	"Information Studies"
e_4	"ResearchGroup"	$V_{Department}$	$V_{ResearchGroup}$
e_5	"Name"	$V_{ResearchGroup}$	V_{RGName}
e_6	~data	V_{RGName}	Computational Informatics Research Group
e_7	"Director"	$V_{ResearchGroup}$	$V_{RGDirector}$
e_8	~data	$V_{RGDirector}$	Peter Willett
e_9	"Focus"	$V_{ResearchGroup}$	$V_{RGFocus}$
e_{10}	~data	$V_{RGFocus}$	database management systems
e_{11}	"AcademicStaff"	$V_{ResearchGroup}$	$V_{RGAcademic}$
e_{13}	"StaffName"	$V_{RGAcademic}$	$V_{StaffN1}$
e_{14}	~data	$V_{StaffN1}$	Barry Eaglestone
e_{15}	"StaffName"	$V_{RGAcademic}$	$V_{StaffN2}$
e_{16}	~data	$V_{StaffN2}$	Angela Lin
e_{17}	"StaffName"	$V_{RGAcademic}$	$V_{StaffN3}$
e_{18}	~data	$V_{StaffN3}$	Miguel Nunes
e_{12}	"HeadofResearch Area"	$V_{RGAcademic}$	Barry Eaglestone

O	
e	Succ
e_1	null
e_2	null
e_3	e_4
e_4	null
e_5	e_7
e_6	null
e_7	e_9
e_8	null
e_9	e_{11}
e_{10}	null
e_{11}	null
e_{12}	e_{13}
e_{13}	e_{15}
e_{14}	null
e_{15}	e_{17}
e_{16}	null
e_{17}	null
e_{18}	null

Table A.1: Data Model for Data-Centric XML Document

Example 2: A Document-Centric XML document

Figure A.3 shows an example of a document-centric XML document. Then figure A.4 shows the graph representation of this XML document and finally, table A.2 shows how this document is mapped using the previous data model. This document was also taken - with slight modifications - from the department of Information Studies, University of Sheffield web site.

```
< Department webaddress="http://www.shef.ac.uk/uni/academic/I-M/is/">
        <Name>Department of Information Studies </Name>
        <Description>
          <Para>Welcome to the Department of Information Studies World
        Wide Web pages where you will find information about the
        Department. Information about
            <List>
              <Item xlink::HREF="../people/people.html"> the staff </Item>
              <Item xlink::HREF="../courses/index.html"> our degree
        courses</Item>
            </List>
                Thank you for visiting our web site
          </Para>
        <Description>
</Department>
```

Figure A.3: Sample Document-Centric XML Document

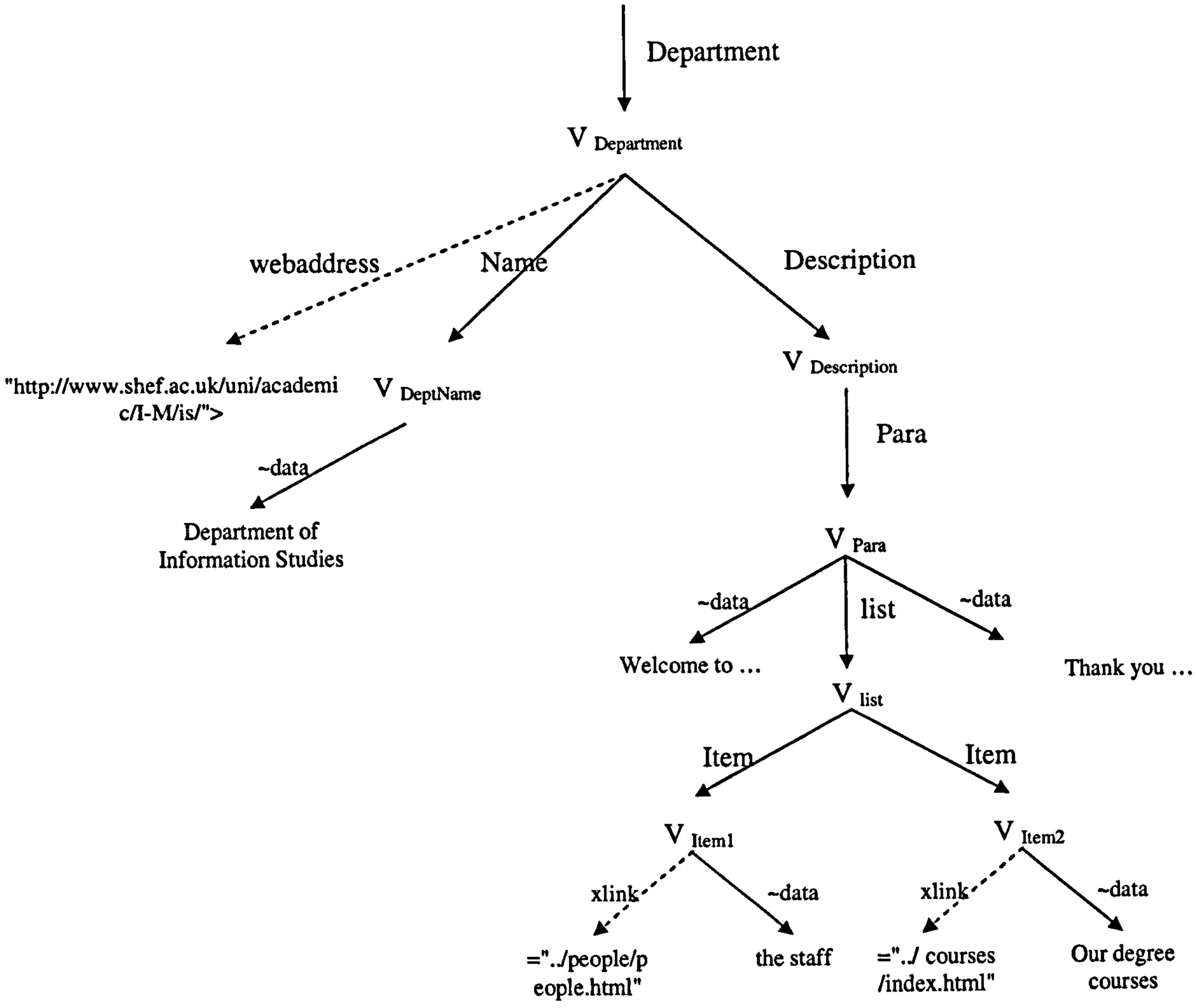


Figure A.4: Graph Representation of Document-Centric XML Document

E			
Edge	Name	Parent	Child
e_1	"Department"	V_{root}	$V_{Department}$
e_3	"Name"	$V_{Department}$	$V_{DepartmentName}$
e_4	~data	$V_{DepartmentName}$	"Department of Information Studies"
e_5	"Description"	$V_{Department}$	$V_{Description}$
e_6	"Para"	$V_{Description}$	V_{para}
e_7	~data	V_{para}	"Welcome to ..."
e_8	"list"	V_{para}	V_{list}
e_9	"Item"	V_{list}	V_{Item1}
e_{11}	~data	V_{Item1}	"the staff"
e_{12}	"Item"	V_{list}	V_{Item2}
e_{14}	~data	V_{Item2}	"our course data"
e_{15}	~data	V_{para}	"Thank you..."
e_2	"webaddress"	$V_{Department}$	"http://www.shef.ac.uk/uni/academic/I-M/is/">
e_{10}	"xlink"	V_{Item1}	"../people/people/html"
e_{13}	"xlink"	V_{Item2}	"../courses/index.html"

O	
e	Succ
e_1	null
e_2	e_3
e_3	e_5
e_4	null
e_5	null
e_6	null
e_7	e_8
e_8	e_{15}
e_9	
e_{10}	e_{11}
e_{11}	null
e_{12}	null
e_{13}	e_{14}
e_{14}	null
e_{15}	null

Table A.2: Data Model for Document -Centric XML Document

Example 3: A hybrid XML document

The following example is a combination of the previous two examples showing a link between the structured part of the document to the unstructured part (such as <Description>) and showing also a link between the unstructured part to the structured part (such as <Head of Department>). Figure A.5 shows the XML document. Then figure A.6 shows the graph representation of this XML document and finally, table A.3 shows how this document is mapped using the previous daa model.

```
<University>

    <Department webaddress="http://www.shef.ac.uk/uni/academic/I-
      M/is/">
      <Name>Information Studies</Name>
        <Description>
          <HeadofDepartment IDREF="PW"/>
          <Para>Welcome to the Department of Information Studies World
      Wide Web pages where you will find information about the
      Department. Information about
              <List>
                <Item xlink::HREF="../people/people.html"> the staff </Item>
              </List>
               Thank you for visiting our web site
            </Para>
          <Description>
            <Staff">
              <Name Id="PW">Peter Willett</Name>
              <Name Id="BE">Barry Eaglestone</Name>
              <Name Id="MN">Miguel Nunes</Name>
             </Staff>
        <ResearchGroup>
          <Name>Computational Informatics Research Group</Name>
          <Director IDREF ="PW"/>
          <Focus>database management systems</Focus>
          <AcademicStaff>
            <HeadofResearchArea IDREF="BE"/>
            <StaffName IDREFS="BE,MN"/>
          </AcademicStaff>
       </ResearchGroup>
    </Department>
</University>
```

Figure A.5: Sample Hybrid XML Document

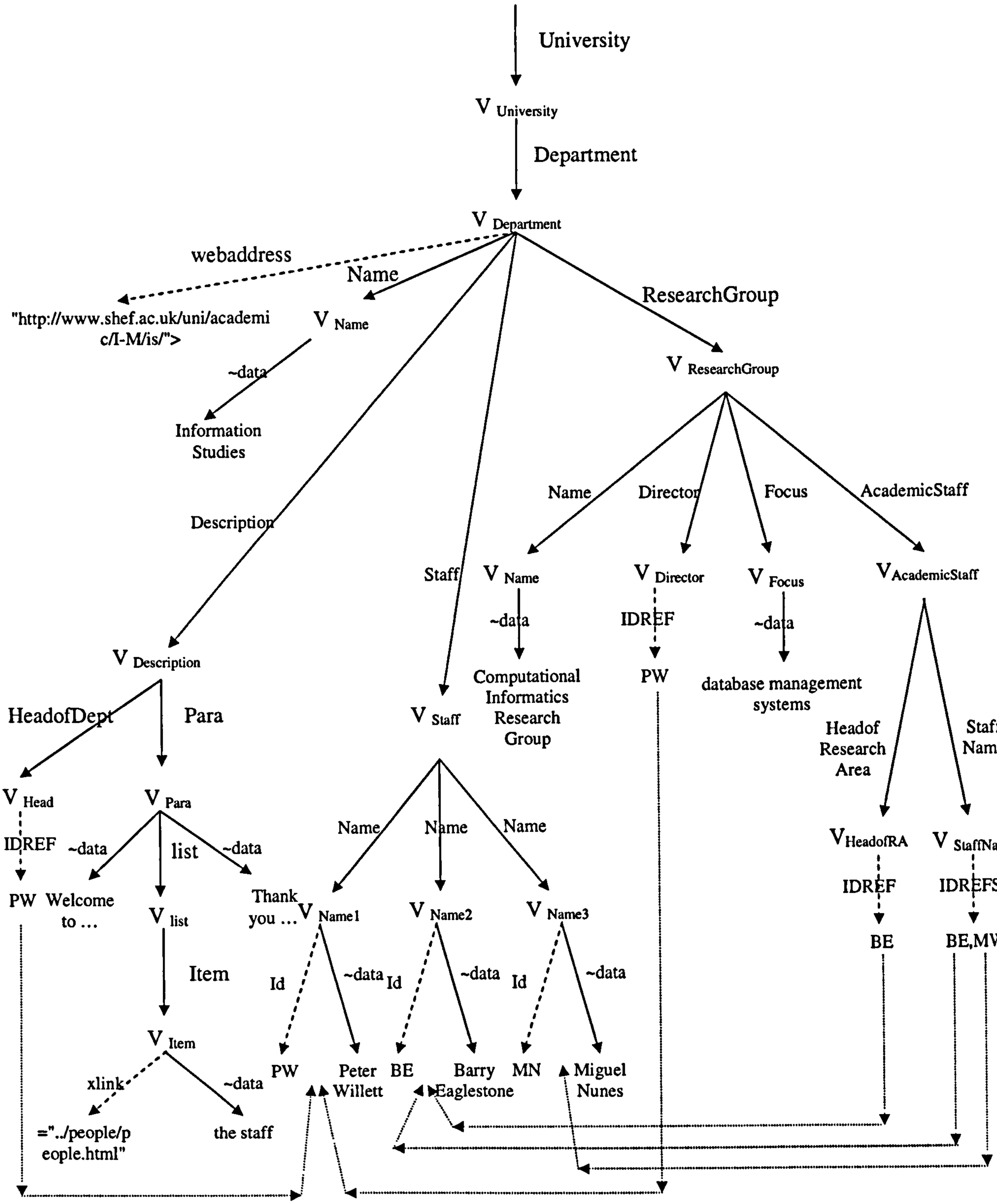


Figure A.6: Graph Representation of Hybrid XML Document

E			
Edge	Name	Parent	Child
e_1	"University"	V_{root}	$V_{University}$
e_2	"Department"	$V_{University}$	$V_{Department}$
e_4	"Name"	$V_{Department}$	V_{Name}
e_5	~data	V_{Name}	"Information ..."
e_6	"Description"	$V_{Department}$	$V_{Description}$
e_9	"Para"	$V_{Description}$	V_{Para}
e_{10}	~data	V_{Para}	"Welcome to..."
e_{11}	"list"	V_{Para}	V_{list}
e_{12}	"item"	V_{list}	V_{item}
e_{14}	~data	V_{item}	"the staff"
e_{15}	~data	V_{Para}	"Thank you ..."
e_{16}	"Staff"	$V_{Department}$	V_{staff}
e_{17}	"Name1"	V_{staff}	V_{Name1}
e_{19}	~data	V_{Name1}	"Peter Willett"
e_{20}	"Name2"	V_{staff}	V_{Name2}
e_{22}	~data	V_{Name2}	"Barry ..."
e_{23}	"Name3"	V_{staff}	V_{Name3}
e_{25}	~data	V_{Name3}	"Migual Nunes"
e_{26}	"ResearchGroup"	$V_{Department}$	$V_{ResearchGroup}$
e_{27}	"Name"	$V_{ResearchGroup}$	V_{Name}
e_{28}	~data	V_{Name}	"Computational .."
e_{29}	"Director"	$V_{ResearchGroup}$	$V_{Director}$
e_{31}	"Focus"	$V_{ResearchGroup}$	V_{Focus}
e_{32}	~data	V_{Focus}	"database .."
e_{33}	"AcademicStaff"	$V_{ResearchGroup}$	$V_{AcademicStaff}$
e_{34}	"HeadofResearch Area"	$V_{AcademicStaff}$	$V_{HeadofResearchArea}$
e_{36}	"StaffName"	$V_{AcademicStaff}$	$V_{StaffName}$
e_3	"webaddress"	$V_{Department}$	"http://www. ...
e_7	"HeadofDept"	$V_{Description}$	("PW",IDREF)
e_{13}	Xlink::HREF	V_{item}	"../people/ ..."
e_{18}	"Id"	V_{name1}	"PW"
e_{21}	"Id"	V_{name2}	"BE"
e_{24}	"Id"	V_{name3}	"MN"
Edge	Parent	refedges	child
e_8	V_{Head}	{ e_7 }	V_{Name1}
e_{30}	$V_{Director}$	{ e_{29} }	V_{Name1}
e_{35}	$V_{HeadofRA}$	{ e_{34} }	V_{Name2}
e_{37}	$V_{StaffName}$	{ e_{36} }	V_{Name2}
e_{38}	$V_{StaffName}$	{ e_{36} }	V_{Name3}

O	
e	Succ
e_1	null
e_2	null
e_3	e_4
e_4	e_6
e_5	null
e_6	e_{16}
e_7	e_9
e_8	null
e_9	null
e_{10}	e_{11}
e_{11}	e_{15}
e_{12}	null
e_{13}	e_{14}
e_{14}	null
e_{15}	null
e_{16}	e_{26}
e_{17}	e_{20}
e_{18}	e_{19}
e_{19}	null
e_{20}	e_{23}
e_{21}	e_{22}
e_{22}	null
e_{23}	null
e_{24}	e_{25}
e_{25}	null
e_{26}	null
e_{27}	e_{29}
e_{28}	null
e_{29}	e_{31}
e_{30}	null
e_{31}	e_{33}
e_{32}	null
e_{33}	null
e_{34}	e_{36}
e_{35}	null
e_{36}	null
e_{37}	e_{38}
e_{38}	null

Table A.3: Data Model for Hybrid XML Document

Appendix B Database Scripts

Figure B.1 shows the SQL Scripts to create the tables used in the experiments. Some of the tables are omitted because they have similar structures. For example, all the 'D_' tables have the exact same structure as 'C_' tables.

```
CREATE TABLE [A_DocType](
      [DocTypeId] [int] NOT NULL,
      [DocType] [nvarchar](20) NULL,
 CONSTRAINT [PK_A_DocType] PRIMARY KEY CLUSTERED
(
      [DocTypeId] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [B_XMLDocument](
      [DocId] [int] NOT NULL,
      [XMLDoc] [xml] NULL,
 CONSTRAINT [PK_B_XMLDocument] PRIMARY KEY CLUSTERED
(
      [DocId] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_Author](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [Author] [varchar](100) NOT NULL,
 CONSTRAINT [PK_A_Author] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_crossref](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [crossref] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_crossref] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_isbn](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [isbn] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_isbn] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_series](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [series] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_series] PRIMARY KEY CLUSTERED
(
      [Id] ASC
```

```
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_school](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [school] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_school] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_chapter](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [chapter] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_chapter] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_cdrom](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [cdrom] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_cdrom] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
SET ANSI_NULLS ON
CREATE TABLE [A_Title](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [Title] [varchar](700) NOT NULL,
 CONSTRAINT [PK_A_Title] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_EE](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [EE] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_EE] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_editor](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [Editor] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_editor] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_BookTitle](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [BookTitle] [varchar](200) NOT NULL,
```

```
 CONSTRAINT [PK_A_BookTitle] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_Pages](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [Pages] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_Pages] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_year](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [Year] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_year] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_address](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [Address] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_address] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_note](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [note] [varchar](500) NOT NULL,
 CONSTRAINT [PK_A_note] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_Journal](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [Journal] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_Journal] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_Volume](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [DocId] [int] NOT NULL,
      [Volume] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_Volume] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_number](
      [Id] [int] IDENTITY(1,1) NOT NULL,
```

```
	[DocId] [int] NOT NULL,
	[Number] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_number] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_Month](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[DocId] [int] NOT NULL,
	[Month] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_Month] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_URL](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[DocId] [int] NOT NULL,
	[URL] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_URL] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_cite](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[DocId] [int] NOT NULL,
	[cite] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_cite] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_publisher](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[DocId] [int] NOT NULL,
	[publisher] [varchar](200) NOT NULL,
 CONSTRAINT [PK_A_publisher] PRIMARY KEY CLUSTERED
(
	[Id] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [C_Doc](
	[DocId] [int] NOT NULL,
	[DocTypeId] [int] NOT NULL,
	[MDate] [datetime] NULL,
	[DocKey] [varchar](150) NOT NULL,
	[ReviewId] [varchar](50) NULL,
	[Rating] [varchar](10) NULL,
	[XMLExtract] [xml] NULL,
 CONSTRAINT [PK_C_Doc] PRIMARY KEY CLUSTERED
(
	[DocId] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_C_Doc] UNIQUE NONCLUSTERED
(
	[DocKey] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
CREATE TABLE [A_Doc](
```

```
      [DocId] [int] IDENTITY(1,1) NOT NULL,
      [DocTypeId] [int] NOT NULL,
      [MDate] [datetime] NULL,
      [DocKey] [varchar](150) NOT NULL,
      [ReviewId] [varchar](50) NULL,
      [Rating] [varchar](10) NULL,
 CONSTRAINT [PK_A_Doc] PRIMARY KEY CLUSTERED
(
      [DocId] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_A_Doc] UNIQUE NONCLUSTERED
(
      [DocKey] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
ALTER TABLE [A_Author]  WITH CHECK ADD  CONSTRAINT
[FK_A_Author_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_Author] CHECK CONSTRAINT [FK_A_Author_A_Doc]
ALTER TABLE [A_crossref]  WITH CHECK ADD  CONSTRAINT
[FK_A_crossref_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_crossref] CHECK CONSTRAINT [FK_A_crossref_A_Doc]
ALTER TABLE [A_isbn]  WITH CHECK ADD  CONSTRAINT [FK_A_isbn_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_isbn] CHECK CONSTRAINT [FK_A_isbn_A_Doc]
ALTER TABLE [A_series]  WITH CHECK ADD  CONSTRAINT
[FK_A_series_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_series] CHECK CONSTRAINT [FK_A_series_A_Doc]
ALTER TABLE [A_school]  WITH CHECK ADD  CONSTRAINT
[FK_A_school_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_school] CHECK CONSTRAINT [FK_A_school_A_Doc]
ALTER TABLE [A_chapter]  WITH CHECK ADD  CONSTRAINT
[FK_A_chapter_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_chapter] CHECK CONSTRAINT [FK_A_chapter_A_Doc]
ALTER TABLE [A_cdrom]  WITH CHECK ADD  CONSTRAINT [FK_A_cdrom_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_cdrom] CHECK CONSTRAINT [FK_A_cdrom_A_Doc]
ALTER TABLE [A_Title]  WITH CHECK ADD  CONSTRAINT [FK_A_Title_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_Title] CHECK CONSTRAINT [FK_A_Title_A_Doc]
ALTER TABLE [A_EE]  WITH CHECK ADD  CONSTRAINT [FK_A_EE_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_EE] CHECK CONSTRAINT [FK_A_EE_A_Doc]
ALTER TABLE [A_editor]  WITH CHECK ADD  CONSTRAINT
[FK_A_editor_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_editor] CHECK CONSTRAINT [FK_A_editor_A_Doc]
ALTER TABLE [A_BookTitle]  WITH CHECK ADD  CONSTRAINT
[FK_A_BookTitle_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_BookTitle] CHECK CONSTRAINT [FK_A_BookTitle_A_Doc]
ALTER TABLE [A_Pages]  WITH CHECK ADD  CONSTRAINT [FK_A_Pages_A_Doc]
FOREIGN KEY([DocId])
```

```
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_Pages] CHECK CONSTRAINT [FK_A_Pages_A_Doc]
ALTER TABLE [A_year]  WITH CHECK ADD  CONSTRAINT [FK_A_year_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_year] CHECK CONSTRAINT [FK_A_year_A_Doc]
ALTER TABLE [A_address]  WITH CHECK ADD  CONSTRAINT
[FK_A_address_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_address] CHECK CONSTRAINT [FK_A_address_A_Doc]
ALTER TABLE [A_note]  WITH CHECK ADD  CONSTRAINT [FK_A_note_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_note] CHECK CONSTRAINT [FK_A_note_A_Doc]
ALTER TABLE [A_Journal]  WITH CHECK ADD  CONSTRAINT
[FK_A_Journal_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_Journal] CHECK CONSTRAINT [FK_A_Journal_A_Doc]
ALTER TABLE [A_Volume]  WITH CHECK ADD  CONSTRAINT
[FK_A_Volume_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_Volume] CHECK CONSTRAINT [FK_A_Volume_A_Doc]
ALTER TABLE [A_number]  WITH CHECK ADD  CONSTRAINT
[FK_A_number_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_number] CHECK CONSTRAINT [FK_A_number_A_Doc]
ALTER TABLE [A_Month]  WITH CHECK ADD  CONSTRAINT [FK_A_Month_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_Month] CHECK CONSTRAINT [FK_A_Month_A_Doc]
ALTER TABLE [A_URL]  WITH CHECK ADD  CONSTRAINT [FK_A_URL_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_URL] CHECK CONSTRAINT [FK_A_URL_A_Doc]
ALTER TABLE [A_cite]  WITH CHECK ADD  CONSTRAINT [FK_A_cite_A_Doc]
FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_cite] CHECK CONSTRAINT [FK_A_cite_A_Doc]
ALTER TABLE [A_publisher]  WITH CHECK ADD  CONSTRAINT
[FK_A_publisher_A_Doc] FOREIGN KEY([DocId])
REFERENCES [A_Doc] ([DocId])
ALTER TABLE [A_publisher] CHECK CONSTRAINT [FK_A_publisher_A_Doc]
ALTER TABLE [C_Doc]  WITH CHECK ADD  CONSTRAINT [FK_C_Doc_C_DocType]
FOREIGN KEY([DocTypeId])
REFERENCES [C_DocType] ([DocTypeId])
ALTER TABLE [C_Doc] CHECK CONSTRAINT [FK_C_Doc_C_DocType]
ALTER TABLE [A_Doc]  WITH CHECK ADD  CONSTRAINT [FK_A_Doc_A_DocType]
FOREIGN KEY([DocTypeId])
REFERENCES [A_DocType] ([DocTypeId])
ALTER TABLE [A_Doc] CHECK CONSTRAINT [FK_A_Doc_A_DocType]
```

Figure B.1: Database Scripts

Appendix C Full Experiments' Results

The following table shows all the results of the experiments.

	Std3	DB 1/3		DB 2/3		DB 3/3	
Query No	System Code	Avg	Std	Avg	Std	Avg	Std
Q01A	100%R	91	18.95	121	24.90	156	73.50
Q01A	100%X	313	42.34	395	91.75	626	19.37
Q01A	100%XW	221	26.54	383	64.06	522	100.41
Q01A	37%X	45	14.29	46	4.66	71	10.39
Q01A	37%XW	38	10.18	42	6.28	63	20.42
Q01A	60%X	60	18.62	75	12.18	83	13.26
Q01A	60%XW	39	9.42	50	8.83	74	13.65
Q01A	PSD	24	8.19	55	8.27	65	30.92
Q01A	PSDA						
Q01A	PSDT						
Q02A	100%R	148	26.94	196	94.89	262	40.93
Q02A	100%X	43097	5570.93	58705	6427.60	73285	6044.47
Q02A	100%XW	210	59.34	379	90.90	580	144.30
Q02A	37%X	78440	6435.08	193910	27243.26	335752	13621.31
Q02A	37%XW	181	75.33	207	54.65	293	121.92
Q02A	60%X	183616	23301.65	411405	16212.19	450089	22445.58
Q02A	60%XW	237	102.97	274	73.94	301	124.83
Q02A	PSD	215	35.99	276	13.80	317	168.16
Q02A	PSDA	70	23.98	75	19.27	120	63.36
Q02A	PSDT						
Q03A	100%R	139	23.98	169	39.29	241	18.13
Q03A	100%X	8538	11.31	22718	165.57	69272	268.23
Q03A	100%XW	166	63.49	300	66.31	441	96.34
Q03A	37%X	81499	4437.62	161943	10669.71	294896	23259.57
Q03A	37%XW	77	1.92	82	8.05	93	2.57
Q03A	60%X	127313	11448.45	308190	22140.57	389509	25956.89
Q03A	60%XW	70	8.07	78	5.63	83	9.10
Q03A	PSD	94	17.81	111	23.09	116	3.25
Q03A	PSDA	12.00	0.44	12.00	0.45	14.00	1.48
Q03A	PSDT						
Q04A	100%R	314	141.63	323	102.72	592	124.93
Q04A	100%X	8766	150.61	42660	4941.41	70586	649.60
Q04A	100%XW	188	57.45	316	81.70	454	97.89

Q04A	37%X	57820	24774.23	174662	10191.38	282379	5817.15
Q04A	37%XW	90	5.65	95	9.81	100	12.35
Q04A	60%X	134342	13021.34	313742	10124.09	371523	11107.27
Q04A	60%XW	83	7.53	84	7.42	86	10.77
Q04A	PSD	81	22.73	92	14.20	97	7.23
Q04A	PSDA	12	0.51	13	1.83	14	1.57
Q04A	PSDT						
Q05A	100%R	195	36.85	232	34.19	241	80.36
Q05A	100%X	22552	1390.93	53622	743.89	75761	6366.67
Q05A	100%XW	190	66.03	315	84.30	444	99.89
Q05A	37%X	17471	1525.08	46501	6408.51	51709	20247.12
Q05A	37%XW	89	6.57	92	4.32	96	8.02
Q05A	60%X	23787	2020.37	66800	9578.58	89296	1348.45
Q05A	60%XW	81	7.08	89	6.56	96	6.44
Q05A	PSD	93	1.08	94	4.09	99	7.60
Q05A	PSDA	12.00	1.07	17.00	1.04	19.00	1.57
Q05A	PSDT						
Q06A	100%R	365	72.24	398	40.06	468	171.06
Q06A	100%X	9144	324.40	52225	2299.58	69593	264.10
Q06A	100%XW	206	76.64	387	72.81	459	164.03
Q06A	37%X	54762	18426.13	148952	56531.47	278190	5949.33
Q06A	37%XW	349	29.72	359	32.42	367	59.20
Q06A	60%X	137073	11138.42	315410	19450.43	361517	10351.30
Q06A	60%XW	356	66.94	373	43.64	438	31.49
Q06A	PSD	360	121.43	377	68.03	396	44.31
Q06A	PSDA	61	31.01	138	63.38	238	165.46
Q06A	PSDT						
Q07A	100%R	509	32.59	1076	28.28	1209	164.76
Q07A	100%X	14101	5135.72	63671	4089.20	78963	9576.67
Q07A	100%XW	227	87.64	249	86.76	308	51.33
Q07A	37%X	82866	15721.11	167265	7894.11	277844	5372.91
Q07A	37%XW	440	79.62	537	211.12	553	77.03
Q07A	60%X	161802	6685.96	373685	189.23	440694	3850.19
Q07A	60%XW	413	52.97	493	99.64	532	60.43
Q07A	PSD	472	107.79	484	79.72	511	139.74
Q07A	PSDA	236.00	147.69	261.00	129.82	294.00	190.84
Q07A	PSDT						
Q08A	100%R	2557	347.21	4866	146.37	5855	1341.39
Q08A	100%X	6663	720.85	57807	4999.97	74909	5251.58

Q08A	100%XW	3121	959.70	24962	1803.61	42579	1185.69
Q08A	37%X	156992	16775.18	288789	49676.01	404319	28077.93
Q08A	37%XW	816	83.08	2174	1075.81	18810	5005.49
Q08A	60%X	262185	35895.58	480492	46241.96	561375	9856.71
Q08A	60%XW	1316	285.40	2595	642.92	29199	7401.66
Q08A	PSD	1822	634.72	23193	3497.09	61481	9052.65
Q08A	PSDA						
Q08A	PSDT	1734.00	837.26	4509.00	1354.14	7855.00	3082.74
Q09A	100%R	117	8.46	124	15.04	142	13.11
Q09A	100%X	417	58.86	563	95.60	788	11.31
Q09A	100%XW	108	16.21	188	43.60	222	49.97
Q09A	37%X	50	18.23	83	6.50	102	37.75
Q09A	37%XW	15	1.46	15	2.28	17	0.98
Q09A	60%X	74	31.13	125	8.96	137	15.65
Q09A	60%XW	15	2.52	15	1.14	21	8.31
Q09A	PSD	14	5.54	15	0.62	18	1.08
Q09A	PSDA						
Q09A	PSDT						
Q10A	100%R	290	134.70	351	12.02	393	21.66
Q10A	100%X	51475	9303.97	102641	464.57	142944	1911.31
Q10A	100%XW	303	16.69	548	55.64	703	66.32
Q10A	37%X	87260	3825.26	194477	6768.67	287414	2237.20
Q10A	37%XW	170	18.44	227	49.83	238	76.46
Q10A	60%X	140460	4587.69	328393	8541.09	380233	925.05
Q10A	60%XW	231	85.10	238	31.56	276	78.46
Q10A	PSD	71	4.47	75	5.57	80	11.73
Q10A	PSDA	14	1.35	14	1.51	14	1.58
Q10A	PSDT						
Q11A	100%R	213	53.84	234	12.02	280	34.41
Q11A	100%X	61993	4470.36	104538	4847.92	148703	1414.21
Q11A	100%XW	1438	675.21	4384	1535.07	8672	1976.47
Q11A	37%X	83105	717.28	194539	4447.27	285851	715.11
Q11A	37%XW	96	20.16	102	25.59	111	6.46
Q11A	60%X	135101	1665.82	327929	7566.20	380807	705.33
Q11A	60%XW	121	7.63	125	23.16	139	13.28
Q11A	PSD	75	6.08	86	8.04	90	7.81
Q11A	PSDA	12.00	1.63	13.00	1.17	15.00	1.06
Q11A	PSDT						
Q12A	100%R	227	9.81	256	90.74	259	47.26

Q12A	100%X	453	218.90	3280	55.64	5364	1337.15
Q12A	100%XW	231	57.75	507	51.37	686	52.76
Q12A	37%X	46	21.01	145	24.57	150	17.32
Q12A	37%XW	98	35.84	100	34.52	107	19.20
Q12A	60%X	90	30.39	125	21.21	239	53.31
Q12A	60%XW	112	14.11	115	9.37	124	21.87
Q12A	PSD	36	6.68	37	8.17	44	13.61
Q12A	PSDA						
Q12A	PSDT						
Q13A	100%R	420	36.06	515	24.18	739	78.69
Q13A	100%X	1160	195.92	4728	1616.62	5726	122.33
Q13A	100%XW	3082	992.17	6882	1419.22	13138	2761.02
Q13A	37%X	68	9.57	111	11.53	145	22.48
Q13A	37%XW	55	18.45	57	7.12	79	32.56
Q13A	60%X	93	17.00	114	7.51	140	42.15
Q13A	60%XW	72	23.67	80	5.85	102	36.87
Q13A	PSD	12	1.57	13	1.17	15	1.21
Q13A	PSDA						
Q13A	PSDT						
Q14A	100%R	19	7.53	76	13.50	219	23.33
Q14A	100%X	118	11.31	188	45.25	522	158.19
Q14A	100%XW	193	51.00	331	121.47	447	221.15
Q14A	37%X	15	1.41	19	9.07	19	1.53
Q14A	37%XW	14	1.55	19	7.00	20	8.52
Q14A	60%X	41	7.51	43	4.24	51	18.25
Q14A	60%XW	14	1.30	15	1.50	17	1.51
Q14A	PSD	13	1.63	13	0.87	14	1.44
Q14A	PSDA						
Q14A	PSDT						
Q15A	100%R	2519	144.27	2630	90.00	2772	27.85
Q15A	100%X	145	22.48	181	36.69	256	75.73
Q15A	100%XW	112	15.63	187	68.54	271	61.72
Q15A	37%X	14	1.73	17	1.41	18	1.00
Q15A	37%XW	15	1.39	15	1.71	20	7.60
Q15A	60%X	12	1.15	14	1.73	15	1.50
Q15A	60%XW	14	2.12	16	2.00	18	4.90
Q15A	PSD	13	1.36	13	1.44	15	1.45
Q15A	PSDA						
Q15A	PSDT						

Q16A	100%R	17721	570.33	25304	1042.22	25938	313.36
Q16A	100%X	146	21.37	344	157.78	1118	264.84
Q16A	100%XW	107	40.36	163	36.46	263	107.13
Q16A	37%X	39	5.32	41	7.55	81	19.19
Q16A	37%XW	19	7.49	24	1.73	37	18.28
Q16A	60%X	42	2.65	53	19.65	63	10.25
Q16A	60%XW	23	8.78	34	2.08	54	16.60
Q16A	PSD	13	1.00	14	1.56	16	1.38
Q16A	PSDA						
Q16A	PSDT						
Q17A	100%R	11288	2523.41	30252	2647.40	34549	3981.95
Q17A	100%X	23620	3375.15	51883	1931.74	64787	957.22
Q17A	100%XW	3315	1286.83	7408	1485.64	30749	2392.47
Q17A	37%X	116991	7127.86	225209	13107.09	333455	1625.73
Q17A	37%XW	820	132.63	1452	286.89	16635	2283.69
Q17A	60%X	182695	3916.78	393269	4773.90	443788	2927.84
Q17A	60%XW	1223	255.90	2159	288.54	25848	2136.88
Q17A	PSD	1298	92.07	23844	1392.61	52799	2350.26
Q17A	PSDA						
Q17A	PSDT	999.00	231.92	4645.00	1729.06	7140.00	2529.06
Q18A	100%R	1832	76.54	4573	226.27	4791	913.81
Q18A	100%X	6364	866.95	52873	782.95	74619	9859.38
Q18A	100%XW	2913	1318.59	7148	1525.97	30355	3798.85
Q18A	37%X	171813	35117.69	329583	10173.51	396239	21116.28
Q18A	37%XW	695	7.99	1269	45.44	15187	915.10
Q18A	60%X	242382	19105.73	509806	48219.55	584999	57239.11
Q18A	60%XW	1054	199.85	1926	35.79	23960	1284.36
Q18A	PSD	1290	97.35	23687	1659.29	50928	3895.13
Q18A	PSDA						
Q18A	PSDT	1098.00	39.61	2983.00	462.78	3602.00	243.65
Q19A	100%R	261	7.07	270	17.32	277	110.44
Q19A	100%X	12671	2849.64	48664	2020.20	69508	251.73
Q19A	100%XW	143	52.88	220	39.47	285	59.26
Q19A	37%X	70127	1494.46	165332	4084.58	276913	3070.43
Q19A	37%XW	69	10.33	99	2.12	103	14.62
Q19A	60%X	125646	4073.28	294526	6584.09	370151	10995.98
Q19A	60%XW	84	10.74	99	11.09	104	7.30
Q19A	PSD	14	1.52	15	1.29	16	1.83
Q19A	PSDA						

Q19A	PSDT						
Q20A	100%R	184	22.26	387	72.31	566	104.07
Q20A	100%X	21115	792.58	42624	1339.62	58719	1064.98
Q20A	100%XW	2706	306.89	6272	1409.53	34161	7450.52
Q20A	37%X	92866	2230.11	200355	3959.68	327124	12867.55
Q20A	37%XW	7906	408.88	8506	737.72	97616	5672.99
Q20A	60%X	161818	3329.09	361084	9034.18	429261	20668.15
Q20A	60%XW	7242	777.86	16091	718.23	155156	7462.53
Q20A	PSD	2331	497.50	53715	5678.67	91635	18660.45
Q20A	PSDA						
Q20A	PSDT	314.00	40.62	33312.00	2807.87	50176.00	1796.30

Table C.1: The Average and Standard Deviation of All the Experiments' Run