School of Electrical Engineering and Computing

Department of Computing

An Effective Technique and Practical Utility for Approximate Query Processing

Tomohiro Inoue

This thesis is presented for the Degree of

Master of Philosophy (Computer Science)

of

Curtin University

November 2015

DECLARATION

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature: ……..……

Date:

ABSTRACT

With the explosive increase in data, query processing times often become troublesome in decision support systems. It takes a much longer time to obtain results - testing the patience of decision makers. Approximate query processing with small random samples is an efficient and effective method for dealing with many queries with huge databases. However, small random samples of highly selective queries or skewed data distribution might miss data which are relevant to the query conditions because of insufficient coverage.

To support an effective random sampling from a database, the ACE tree has been proposed as a novel file organisation mechanism. It facilitates, statistically, returning a random sample of the database records satisfying the query condition. However, the height of the index tree becomes much higher for high-dimensional data. To tackle this problem, the k-MDI expanding the ACE tree has been proposed as an efficient index scheme to draw a random sample of records from a database. Since the height of the index tree is very shallow, it achieves a more efficient index search than the ACE tree, but the feasibility of its implementation has not been demonstrated.

In this thesis, a hybrid view with an index called the Multidimensional Cluster Sampling View based on the k-MDI is developed to support efficient and effective approximate query processing in large databases. The view is designed to enable it to be simply implemented on database tables and manipulated in SQL. This view also facilitates drawing a random sample of records which are more relevant to the query condition even for a highly selective query.

The feasibility of the Multidimensional Cluster Sampling View is shown by the development of utility software. The software can build a view in database tables with a few parameters and execute approximate query processing. The software design, a construction method of the view, and an execution method for approximate query processing in the view are presented in this thesis. Also, it is demonstrated that approximate query processing in the view provides a high degree of accuracy on four-dimensional data even in highly selective queries. Moreover, it is shown that the query processing is faster than full scans and the cutting edge database management system as columnar databases.

The practicality of the Multidimensional Cluster Sampling View is examined and evaluated by many experiments in terms of dimensionality. The evaluations demonstrate that the number of dimensions does not have an impact on the level of accuracy of the approximation, query processing time and construction time of the view.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Dr Aneesh Krishna, and my co-supervisor, Dr Raj Gopalan, for their support, guidance and faith in me. Without them, this research would have been very hard to endure both technically and mentally. They have been extremely patient and gentle despite my initial ignorance of various subjects required for this work. Thanks to Aneesh for being a great source of motivation and good advice. Thanks to Raj for his invaluable discussions and ideas which helped keep the research significant and relevant. I am forever grateful for their comments and countless hours spent to assist me in preparing the thesis. Also, I would like to express my sincere thanks to Associate Professor Mihai Lazarescu for serving on the research committee.

In addition, I would like to thank the administrative staff in the Department of Computing for various paperwork and the arrangement of equipment. I am very thankful to Curtin University.

Finally, this work would not have been possible without the constant encouragement and support of my family and friends. In particular, special thanks to my wife, Sakurako, for her mental support.

PUBLICATIONS

This is a list of referred papers that is related to this research work.

- Inoue, T., Krishna, A., & Gopalan, R. P. (2015). *Multidimensional Cluster Sampling View on Large Databases for Approximate Query Processing.* Paper presented at the Enterprise Distributed Object Computing Conference (EDOC), 2015 IEEE 19th International.

- Inoue, T., Krishna, A., & Gopalan, R. (2016). Approximate Query Processing on High Dimensionality Database Tables Using Multidimensional Cluster Sampling View. *Journal of Software, Academy Publisher, ISSN 1796-217X, 11*(1), 80-93.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background

As a consequence of the rapidly changing world, organisations began to utilise computers to support the decision-making and planning process, namely the use of decision support systems proposed by Bonini (1963). A few decades ago, operational or transactional database systems were optimised to endorse daily business operations (Vaisman & Zimányi, 2014). In other words, these systems were not suitable as a data analysis platform. Although the operational systems may contain detailed data of a certain period, they do not contain historical data in large volumes (Inmon, 2002; Kimball, Ross, & Thornthwaite, 2011). For this reason, such an operational or transactional database could not support the decision-making and planning process effectively. As a result, a concept called data warehousing emerged as one of the solutions to decision support systems containing historical data. This is known as data-oriented or data-driven decision support systems.

Data analysis in data warehouses requires complex queries including many tables, join operations and aggregations. Therefore, it takes a long time to execute ad-hoc decision support queries into the accumulated data. For fast analytical query processing, data cubes with pre-computing are used as one of the solutions. The number of data cubes depends on the number of dimensions, namely the number of key attributes. It is almost impossible to create full materialised data cubes in a data set with high dimensions in the real world because of the substantial combination of the key attributes (Li, Han, Yin, Lee, & Sun, 2008). Thus, the number of dimensions in data warehouses, especially high-dimensional data, often causes a serious problem in practice.

From the Digital Universe Study, the amount of digital data in the world increased from 130 exabytes in 2005 to 1,227 exabytes in 2010 (Gantz & Reinsel, 2012), which is a growth of 219 exabytes per year. In addition, digital data reached 4.4 zettabytes (4,400 exabytes) by

2013 (International Data Corporation, 2014), which is a growth of 1,058 exabytes per year from 2010. Furthermore, International Data Corporation (2014) estimates that all data in the world doubles every two years. While such a massive amount of data can be utilised for decision support, the explosive increase of data has caused further problems in data warehouse applications. The traditional query processing for analysis has required a further long running time because the processing must access huge volumes of data (Babcock, Chaudhuri, & Das, 2003; Chakrabarti, Garofalakis, Rastogi, & Shim, 2001; Garofalakis & Gibbon, 2001; Liu, 2009; Ruoming, Glimcher, Jermaine, & Agrawal, 2006; Wang, Wang, & Li, 2013). Moreover, useful data for analysis were at 22% of the information in 2013, and only 5% of data in the world was actually analysed (International Data Corporation, 2014). The valuable data is set to grow to 37% of the information by 2020 (International Data Corporation, 2014). This means that a further increase of data utilised in analysis is projected for the total amount of digital data in the world. Therefore, alternative techniques, methods, or architecture are needed to deal with the massive amounts of generated data efficiently in data warehouses.

There are currently some solutions available to address the problem. The latest database platforms that include in-memory databases, such as SAP High-Performance Analytic Appliance (HANA) or Oracle Exadata, can provide the fastest query response times. Also, query processing on large scale columnar databases, such as Apache Cassandra or Amazon Redshift, is also considerably fast. This capability comes from distributed processing architecture and many hardware resources such as many CPUs and large memories; these computer systems are very expensive and complicated. The columnar databases, which organise data by column units instead of row units, are one of the options available to deal with a large amount of data efficiently. The databases with novel architecture do not depend on hardware resources, where it is difficult to handle high-dimensional queries within a reasonable response time. Another analytical approach to compute huge volumes of data efficiently is the use of approximate query processing. Basically, it is not necessary for decision makers to obtain perfect answers (Agarwal et al., 2014; Chakrabarti et al., 2001; Cormode, Garofalakis, Haas, & Jermaine, 2012; Rudra, Gopalan, & Achuthan, 2012; Wu, Ooi, & Tan, 2013). For instance, consider the monthly sales analysis in a retail company. On the one hand, the absolute answer for sales in a month is $103,092, and it takes two minutes to obtain the result. On the other hand, the approximate answer in the same month is $100,000 with a 3% error, and it takes only two seconds. It is clear that approximate query processing with fast processing is preferred in the decision-making process. The decision

makers do not care about the small relative error (Liu, 2009). It is important to identify trends or patterns in such a statistical analysis. Thus, fast processing with good approximation and acceptable accuracy is one of the valuable ideas to support the decision-making process in the Big Data era.

Although there are some approaches to the approximate query processing in databases, this thesis focuses on the use of random sampling. Random sampling is a very efficient method for dealing with massive volumes of data (Joshi & Jermaine, 2008a; Olken & Rotem, 1986; Rudra et al., 2012; Webb & Wang, 2014). However, small samples obtained through random sampling methods might lack the appropriate data relevant to the query conditions because the samples do not adequately represent the entire dataset. Large samples obtained through random sampling methods can improve the quality of the results, but it takes a much longer time. As a result, the advantage of random sampling becomes impaired. Furthermore, another serious issue of drawing random sample records from a database is that picking database records at random forces the reading of almost distinct data blocks, which is equivalent to reading the entire dataset (Olken & Rotem, 1990). This is also a significant challenge in using random sampling methods in databases efficiently.

1.2 Aim and Scope of the Thesis

This research program is a study of approximate query processing with a fast response time and acceptable level of accuracy. The main aims for the research are as follows:

The first aim is to develop an effective and efficient method based on random sampling for approximate query processing in databases. The method enables the provision of both fast query processing and approximation with acceptable accuracy simultaneously.

The second is to develop a special software tool enabling the implementation of the method into a large database with a few parameters in order to demonstrate the feasibility of the method. Also, the tool enables execution of approximate query processing with a user-friendly interface.

The final aim is to assess and evaluate the effectiveness and efficiency of the method in order to demonstrate the capability and practicality of the method.

This thesis makes valuable contributions in the field of approximate query processing based on random sampling in databases.

1.3 Thesis Structure

The remainder of this thesis is organised as follows.

Chapter 2 provides concepts and definitions of relevant terms used in this thesis. Also, it reviews previous studies regarding approximate query processing based on random sampling, which are directly linked to this research.

In Chapter 3, a method to achieve effective and efficient approximate query processing, even in highly selective queries, is proposed. The method is considered through implementation into common database tables and is manipulated by SQL statements. The accuracy of the approximations using the method is examined on a large database with three-dimensional data.

Chapter 4 describes how the practical utility which was developed for this research builds the proposed method on database tables and executes approximate query processing in SQL. Also, the effectiveness of approximate query processing by the method is examined and evaluated on a large database with four dimensions using the utility. Moreover, query performance of the proposed method is empirically examined and evaluated in detail.

In Chapter 5, the effects of dimensionality on accuracy in the proposed method are examined and evaluated in detail. The impact of dimensionality on construction time of the method is also examined and evaluated in detail.

Finally, Chapter 6 concludes the thesis along with some suggestions for future research.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Over a few decades, data warehouses have become a more important tool to support decision-making processes. Many academic researchers and database vendors such as Oracle, IBM and Microsoft have been looking for useful techniques for data warehouses. In recent years, an explosive increase of information has brought about slow query processing increasingly in decision support systems. Approximate query processing is a fascinating method for dealing with huge volumes of data, whereas there are some problems which must be solved in practice. This thesis aims to study an effective and practical technique for approximate query processing, which enables the provision of a fast query response time and high accuracy of approximation in any highly selective query.

This chapter reviews recent studies, published in the literature, regarding approximate query processing based on random sampling. The studies are directly linked to this thesis due to their useful techniques for approximate query processing. The rest of this chapter is organised as follows. In Section 2.2, relevant terms and concepts regarding this thesis are presented. Section 2.3 explains and discusses approximate query processing. Section 2.4 describes the ACE Tree, which is a novel index structure to support efficient approximate query processing based on random sampling. In Section 2.5, the k-MDI tree, which extends the ACE tree, is described in detail due to it being a basis of this thesis. Finally, this chapter is summarised in Section 2.6.

2.2 Decision Support System

In 1963, Charles Bonini introduced the use of the computer to support decision makers, then a term called 'decision support system' was created by Scott Morton in 1971 (Eom & Kim, 2006). The ***decision support system*** (***DSS***) is an information system focusing on supporting decision-making processes (Arnott & Pervan, 2014). It can be applied to a wide range of areas such as sales planning, marketing and products planning. For example, a sales manager can use the decision support system to predict future demands from the data of last year and the year before. Nowadays, various systems, applications and methods are included in the decision support systems; for instance, data warehouses, online analytical processing, data mining, approximate query processing, parallelism (or distributed computing), columnar databases, in-memory databases, and so forth. In this Section, relevant terms and concepts regarding this thesis are given.

2.2.1 Data Warehouse

Data warehouses, which form a main part of decision support systems, are described in this section. The ***data warehouse*** (***DWH***) is characterised as a collection of subject-oriented, integrated, non-volatile, and time-variant data that supports management's decisions (Inmon, 2002).

The first term, called subject-oriented, indicates several targets or subjects of analysis at a variety of levels of the decision-making process (Inmon, 2002). For instance, the major subject areas of a retail company may be sales or inventory of products.

The second term, called integrated, expresses the result data which are converted, reformatted and summarised from various operational and external systems (Inmon, 2002). As shown in Figure 2.1, the system B has a gender data as 'male' and 'female'. Another system C has a gender data as '1' meaning male and '0' meaning female. When these data are collected from different sources into a data warehouse, they are encoded into common expressions such as 'M' and 'F'. As another example, attribute measurements are integrated such as from inches to centimetres. A similar consideration of consistency for descriptions and key structures is also included.

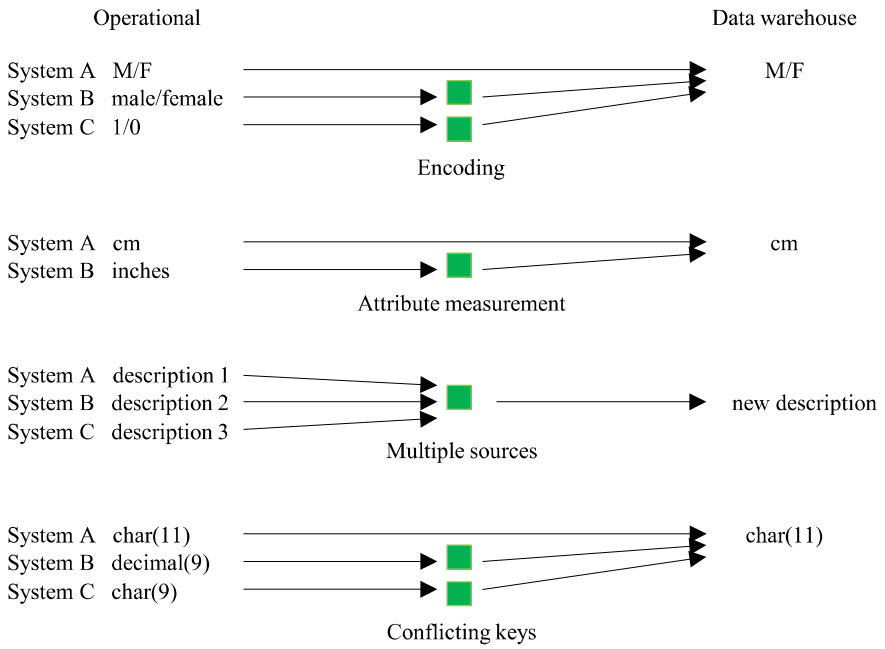


Figure 2.1 Examples of integration

The third term, called non-volatile, means that data are loaded into the data warehouse and kept (Inmon, 2002). The data in the data warehouse are not updated or deleted (in the general sense). As a result, a history of data is maintained in the data warehouse as shown in Figure 2.2.

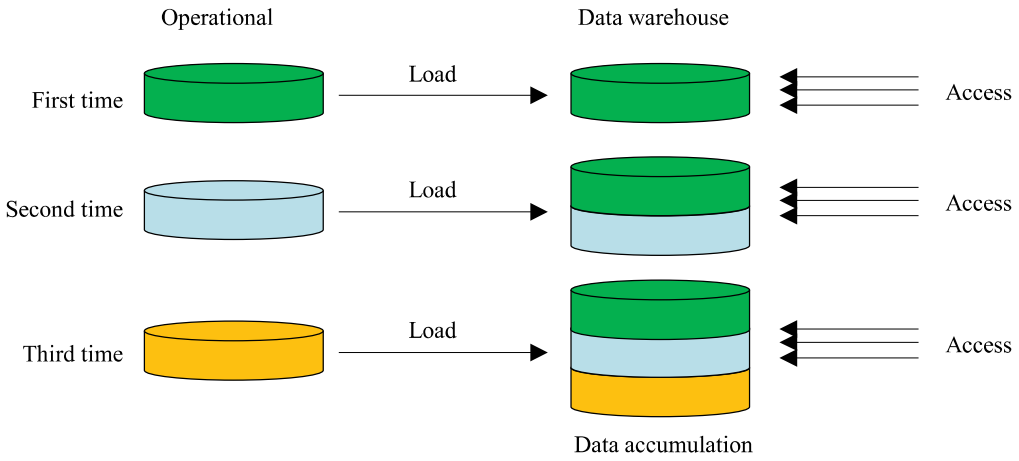


Figure 2.2 Examples of non-volatility

Finally, the term called time-variant implies that data in the data warehouse can be tracked as to how its data has changed over time (Inmon, 2002). The data warehouses, for instance, provide the sales trend in a store over the last years. As just described, some elements of time such as year, month and day, are always contained in the data warehouse. In operational systems, the elements of time may not be included.

Thus, the data warehouse is a repository which is created to provide useful information collected from various sources.

2.2.2 Data Warehouse Architecture

In this section, a general architecture model of data warehouse systems is presented to establish the big picture. Figure 2.3 depicts a data warehouse architecture model introduced by Kimball et al. (2011). The model can be considered as one of the decision support systems. From a high-level view, there are two rooms in the data warehouse architecture model.

On the one hand, the ***back room*** shows the flow of input data from source systems. The data in the source systems are integrated by the ***extract-transformation-load*** (***ETL***) system and then are stored in the presentation server, which is generally called a ***DWH server***.

On the other hand, the ***front room*** shows the flow of output data to users. The accumulated data in the presentation server are used by ***business intelligence*** (***BI***) applications, which are computer software that enable the design and building of the reports and analyses.

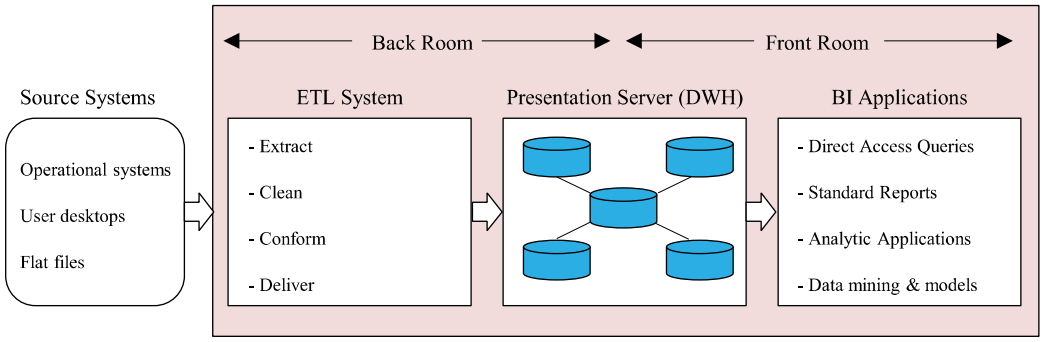


Figure 2.3 A data warehouse architecture model

2.2.3 Dimensional Modeling

In a logical design of data warehouses, dimensional modeling is widely accepted to achieve fast query processing as a data structure. The analysis data are divided into measurements and context. The measurements referred to as ***facts*** are captured by the business processes and operational data sources (Kimball et al., 2011). In other words, they are interesting aspects for data analysis and usually numeric values. The context is a set of logical and independent elements for analysis called ***dimensions*** (Kimball et al., 2011). Figure 2.4 illustrates an example of dimensional models for sales data with three dimensions. The dimensional model is composed of a fact table named SALES and three dimension tables named DATE, PRODUCT and STORE. The fact table contains the numeric measurements called ***measure attributes*** and the keys called ***key attributes***. The dimension tables contain the textual context including labels forming full words, descriptions, and depth of the attributes like year-month-day.

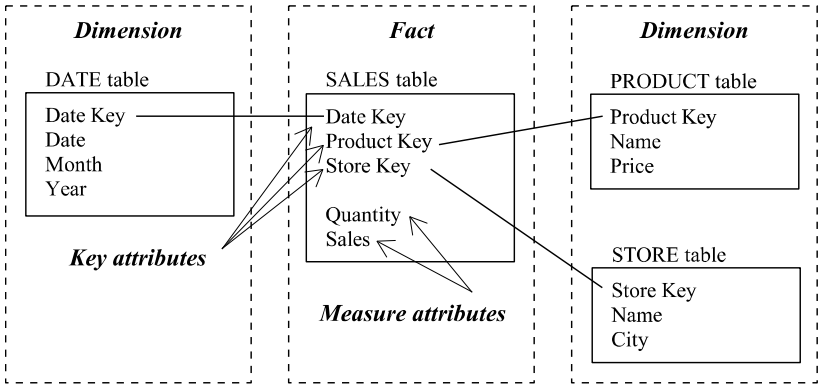


Figure 2.4 A dimensional model for sales data with three dimensions (date, product and store)

The star-like structure centred in the fact table is called a ***start join***. Typically, dimensional models stored in relational databases are called ***star schemas***, and dimensional models stored in n-dimensional spaces are called ***cubes,*** which are usually used for OLAP applications that are a part of the BI area (Kimball et al., 2011). The ***online analytical processing*** (***OLAP***) facilitates interactive query execution, automatic aggregation and analysis from a variety of viewpoints in a data warehouse (Vaisman & Zimányi, 2014). In the data warehouse architecture model, the OLAP tools (applications) are one of the BI applications. The common aggregate functions are COUNT (*), SUM (*), AVERAGE (*), MIN (*) and MAX (*). The DWH server sometimes contains pre-aggregating data made by ETL systems and cubes made by OLAP applications. The pre-aggregating data are useful in improving the analytic query performance. At the implementation level, ***materialised views***, which define view definitions that materialise from or the storing of result sets (Kimball et al., 2011), are one of the pre-aggregation examples. Figure 2.5 illustrates an example of a three-dimensional cube for sales data with dimensions Time, Store, and Product, and a measure quantity.

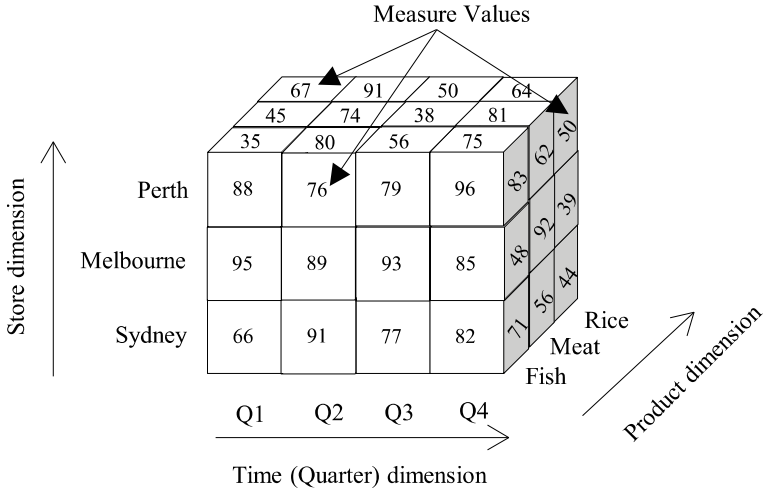


Figure 2.5 A three-dimensional cube for sales data with dimensions Time, Store, and Product, and a measure quantity

The cube is created by aggregating the original data in advance in terms of dimensions. For example, if a manager investigates the sales of quarter one at a Sydney store, the cube can immediately return a precomputed value of 66 without aggregation. However, there are two disadvantages in the cube model. The first one is the need to decide the analysis axis (dimensions) and the level of detail in advance. If the analysis axis created is lacking, the cube must be rebuilt. Another disadvantage is that it is often impossible for all possible queries to create the full materialisation of the cube in the real world because of the many combinations (Babcock et al., 2003; Li et al., 2008). Therefore, it is essential to design the cube models with careful consideration of what lies ahead.

At this point, many terms and concepts have been presented. They can be classified as shown in Figure 2.6.

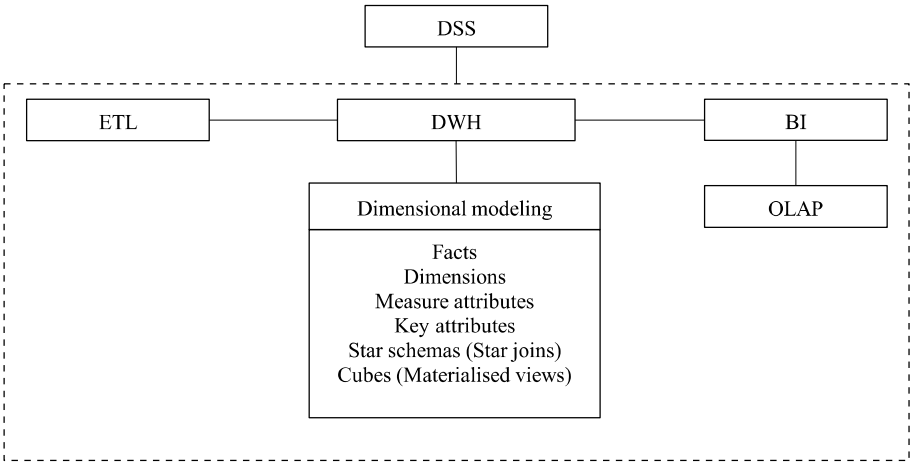


Figure 2.6 Basic terms surrounding data warehouses

2.3 Approximate Query Processing

According to Garofalakis and Gibbon (2001), with the growth of the decision support systems, the following three motivations brought about approximate query processing. First, decision support analysis by OLAP applications is usually exploratory. With a massive increase of information data, even one aggregation query of decision support systems has been consuming the precious time of decision makers. Since many queries are generally attempted to identify truly interesting data, a useful technique with a short response time is required. Second, in many aggregate queries, precision to the 'last decimal' is not needed for the answers. In other words, approximate answers with some error guarantee are acceptable for decision support. Finally, base data are sometimes unavailable for some reason such as network limitation or disk storage failure. An alternative option is to provide approximate answers using local cached data synopses.

Approximate query processing is defined as providing both approximate answers with acceptable accuracy and faster query processing than the query processing that obtains absolute answers (Liu, 2009). Therefore, it can be considered as a kind of OLAP. It is unnecessary for decision makers to obtain exact answers. In other words, approximation with an acceptable level of accuracy is sufficient for decision making. Also, the approximate query processing is mainly designed for aggregate queries such as using the COUNT, SUM

and AVERAGE function. The approximate answers usually accompany the relative error. The relative ***error*** of a given query Q can be quantified as follows:

$$Error(Q) = \left| \frac{(Exact\ Answer - Approximate\ Answer)}{Exact\ Answer} \right| \quad (2.1)$$

For approximate query processing, there are two fundamental approaches, which are online query processing and pre-computed synopsis as shown in Figure 2.7.

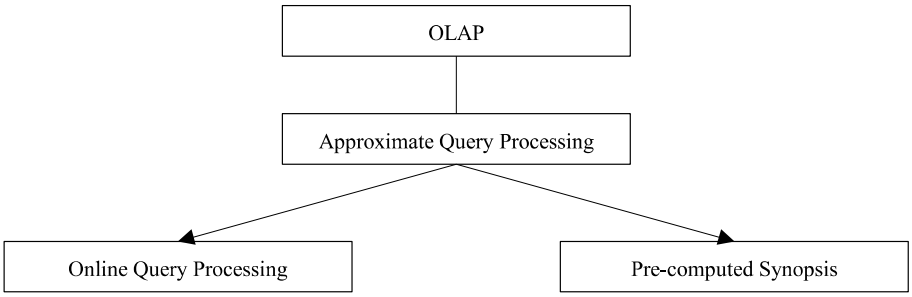


Figure 2.7 Two approaches for approximate query processing

2.3.1 Online Query Processing

The ***online query processing*** is an iterative query processing, which is continually refining the accuracy of answers in increasing samples until it has computed the complete data set or is stopped in the processing (Hellerstein, Haas, & Wang, 1997; Raman & Hellerstein, 2002; Wu et al., 2013). Figure 2.8 depicts the architecture of online query processing. The client requests a query to the DWH server, and then the server returns a partial result of the query to the client. The client displays the approximate answer using the partial result, whereas the approximate answer is not statistical in nature. This process is iterated, and the client updates the approximate answer by combining the holding data and the new partial result until the end of the data or cancellation of the processing. However, this approach conflicts two different goals; it tries to minimise the waiting time for users using small samples, whereas it tries to improve the level of accuracy using large samples (Liu, 2009). Furthermore, in order to support online query processing, relational database engines must be significantly extended (Hellerstein et al., 1997). The online query processing does not require preparation

such as data processing in advance, and users can control the query processing (Wu et al., 2013). However, users require fast query processing and acceptable approximate answers in decision support systems rather than controlling the query processing.

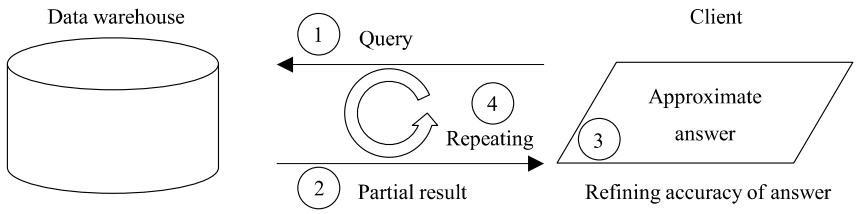


Figure 2.8 Architecture of online query processing

2.3.2 Pre-computed Synopsis

As shown in Figure 2.9, the ***pre-computed synopsis*** is composed of the construction phase and the query answer phase. In the construction phase, data synopsis is created offline and stored for the query answer phase in advance; in the query answer phase, it returns an online approximate answer using the data synopsis (Babcock et al., 2003; Ioannidis, 2003; Liu, 2009). In general, techniques used in the construction phase can be mainly classified into three categories, which are histograms, wavelets and samples.

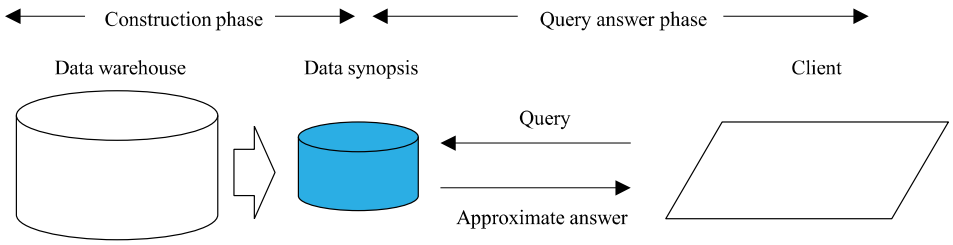
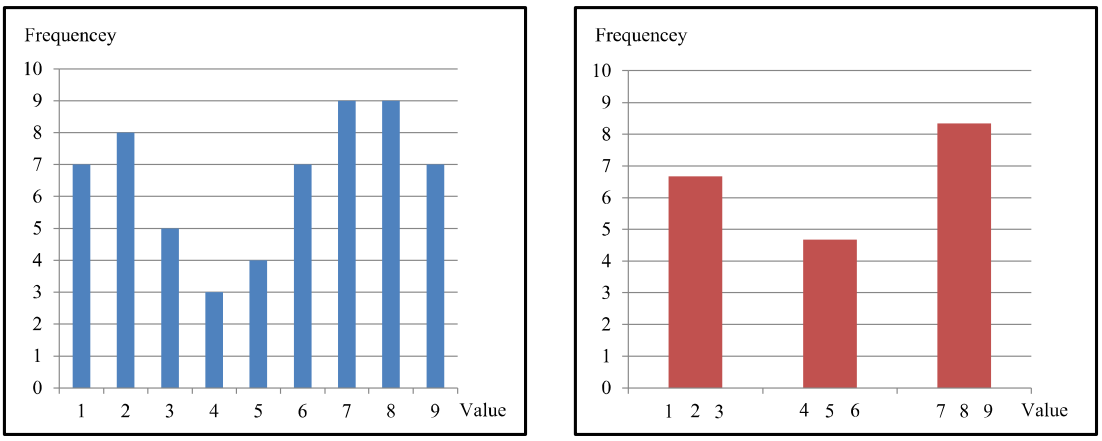


Figure 2.9 Architecture of pre-computed synopsis

2.3.2.1 Histograms

A ***histogram*** is constructed by partitioning the attribute value domain into many disjoint subsets called ***buckets*** (Garofalakis & Gibbon, 2001; Ioannidis, 2003; Liu, 2009). The histogram facilitates the approximation of the frequencies and values in each bucket. Figure 2.10 is an example which partitions the attribute values into three buckets, namely, the first bucket for values between 1 and 3, the second bucket for values between 4 and 6, and the third bucket for values between 7 and 9. It is easy to capture the data distribution. This is the most common technique adopted by major database applications such as the DB2, the Oracle Database and the Microsoft SQL Server because of no run-time overheads, low error estimates, and a requirement of reasonably small space (Liu, 2009).



(a) Data Distribution **(b) Histogram**

Figure 2.10 An example of histogram

The advantage of the histogram technique is in answering a variety of query types including aggregation and non-aggregation. However, the evaluations using the histogram approach usually rely on the experimental results because it is hard to calculate an error bound theoretically; moreover, the histogram technique in the high-dimensional data sets leads to new problems such as storage overheads through an increase in the number of buckets and an increase in construction costs of the histogram (Chakrabarti et al., 2001; Cormode et al., 2012; Liu, 2009). Furthermore, most histogram techniques require several complicated parameters for a better approximation, and such complicated parameters can bring about summaries with poor performance by utilising wrong choices (Cormode et al., 2012).

2.3.2.2 Wavelets

Wavelets can be used for hierarchical decomposition of functions as a useful mathematical tool (Chakrabarti et al., 2001; Garofalakis & Gibbon, 2001; Liu, 2009). ***Haar wavelets*** are the simplest concept in the wavelets and can be easily understood and implemented. Consider a one-dimensional data vector V containing four values $V = [3, 5, 8, 4]$. The Haar wavelets transform is computed from ***higher-resolution*** and ***lower-resolution*** as shown in Table 2.1. For example, the average of the first two values (namely, 3 and 5) is 4, and the average of the next values (namely, 8 and 4) is 6. The computed average array $[4, 6]$ are values used at the next resolution (one). It is clear that some information has been lost. To recover the lost information, detail coefficients are calculated and stored. The ***detail coefficients*** are differences of the second pair of values from the computed average value. The first detail coefficient of computed average values 4 at resolution one is -1 because of $(4 - 5)$. The second detail coefficient of computed average values 6 at resolution one is 2 because of $(6 - 4)$.

Table 2.1 Haar wavelets transform

Resolution	Averages	Detail coefficients
2	[3, 5, 8, 4]	N/A
1	[4, 6]	[-1, 2]
0	[5]	[-1]

The hierarchical decomposition tree of the Haar wavelet is created using the computed last average value ($[5]$) and all detail coefficients ($[-1], [-1, 2]$) which are from lower to higher resolution. The one-dimensional ***Haar wavelet transform*** of vector V is denoted by $W_v = [5, -1, -1, 2]$, and each element is called a ***wavelet coefficient***. Figure 2.11 illustrates the hierarchical decomposition tree of the Haar wavelet in the example. The ***internal nodes*** are composed of the computed last average value and detail coefficients. The ***leaf nodes*** are composed of the original data array. The original data array can be reconstructed from values of the internal nodes by inverting the decomposition exactly.

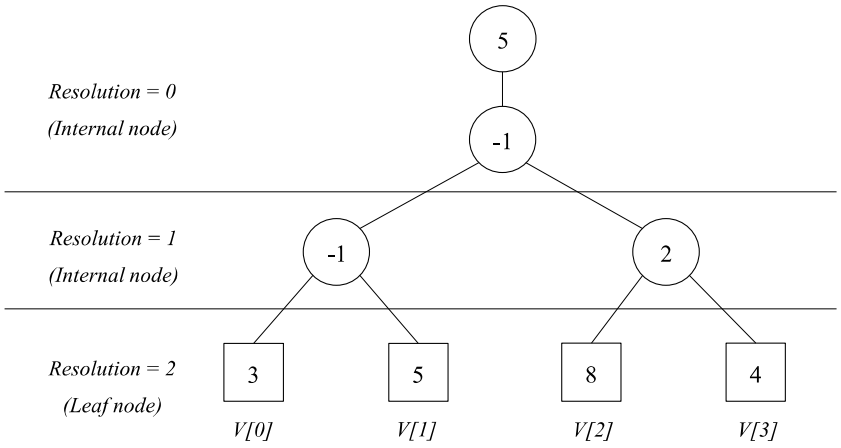


Figure 2.11 An example of hierarchical decomposition tree of Haar wavelet

The primary motivation for the use of Haar wavelet transform W_v instead of the original data is that most detail coefficients (differences) tend to be small values. Therefore, even if such smaller coefficients than a certain ***threshold*** are replaced to zero in the wavelet transform W_v, the effects on the accuracy are small when reconstructing the original data (Chakrabarti et al., 2001; Cormode et al., 2012). By replacing smaller detail coefficients to zero and storing the wavelet transform W_v excepting zero, the original data can be compressed instead of sacrificing accuracy of the data. As shown in Figure 2.12, which is an example of reconstructed data using the inverse Haar wavelet transform, the reconstructed data is similar to the original data. Since the threshold is very small, the compression rate of the data is at 50% in this example. The larger threshold leads to the higher compression rate, whereas the level of accuracy is lower. Thus, even a Haar wavelet transform, which is the simplest in the wavelets, is a highly complicated mechanism.

Original data	5	6	8	2	5	4	7	9	3	5	1	6	8	9	1	3
						Haar wavelet transform										
Wavelet coefficients	5.125	0.625	-0.5	-0.75	0.25	-1.75	0.25	3.25	-0.5	3	0.5	-1	-1	-2.5	-0.5	-1
						Threshold = 1										
Final Wavelet coefficients	5.125	0	0	0	0	-1.75	0	3.25	0	3	0	-1	-1	-2.5	0	-1
						Inverse Haar wavelet transform										
Reconstructed data	5.125	5.125	8.125	2.125	3.375	3.375	5.875	7.875	4.125	6.125	2.625	7.625	8.375	8.375	0.875	2.875

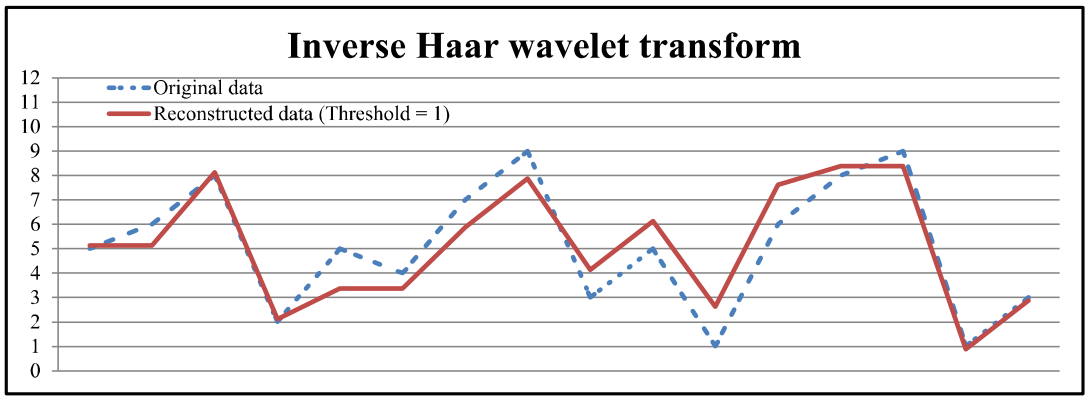


Figure 2.12 An example of reconstructed data using the inverse Haar wavelet transform

A recent study has shown that approximate query processing using wavelets can handle high-dimensional queries with low construction costs and low storage overheads compared with the histogram technique (Liu, 2009). However, in practice, more coefficients must bring about more accurate reconstruction of the data distribution compared with reduced coefficients for high-dimensional queries. Techniques to provide error estimates of the approximation have not yet been well developed in the same manner as the histogram techniques; such as useful feedback for decision makers which is crucial in approximate query processing (Cormode et al., 2012). In addition, the implementation of this approach is highly complicated in practice. Therefore, the implementation must accompany significant modification in database engines.

2.3.2.3 Samples

A ***sample*** is a set of data which are randomly selected through some process, and the ***sampling*** technique, which has a long history of over 30 years, is probably the most basic approach for approximate query processing (Agarwal et al., 2014; Cormode et al., 2012).

The technique is also well-understood and established as a field of scientific study (Ruoming et al., 2006). The main motivation for the use of sampling is that a random sample has a characteristic which often represents the entire data (Garofalakis & Gibbon, 2001; Liu, 2009). Many different methods for sampling have been proposed with ***estimators*** which guess the answer to the query. The most basic method is simple random sampling. There are two types of simple random sampling. The first is ***simple random sampling with replacement*** (***SRSWR***). The process of SRSWR is that an item is chosen at random from the dataset and added to the sample, and then the chosen item is returned into the dataset again; therefore, the same item may be sampled more than once. Another type is ***simple random sampling without replacement*** (***SRSWOR***). Compared with the SRSWR, the SRSWOR does not return the chosen item into the dataset after choosing an item randomly. Since the SRSWOR generally provides more information for a given sample size than the SRSWR (Olken & Rotem, 1990), the SRSWOR is more useful for approximate query processing.

The sampling-based approximation allows both assessment and control of the errors of estimation (Liu, 2009). The ***confidence bound*** (***confidence level***) with 95% probability, namely within $\pm 5\%$ error (***confidence interval***), is commonly used to justify the approximate value (Cormode et al., 2012). For instance, the approximate answer is $95{,}000 \pm 5{,}000$ with at least a 95% probability.

The sampling-based approximation method is a simple process, and it is flexible regarding the queries and suitable for higher dimensional data (Cormode et al., 2012). The simplicity and flexibility is ideal for approximate query processing in the real world. In addition, sampling techniques can easily improve the level of accuracy by increasing the sample size. One of the key issues for sampling from databases is that row-level sampling leads to a large amount of disk I/O because it almost scans the entire disk block to obtain the sampled rows (Cormode et al., 2012; Garofalakis & Gibbon, 2001). In contrast, the block-level sampling approach can significantly reduce the disk I/O, but the sample is not necessarily independent, that is, not unbiased (Cormode et al., 2012; Garofalakis & Gibbon, 2001; Olken & Rotem, 1995). Another key issue is that a small sample may miss relevant data when the query is highly selective or when the dataset has a skewed distribution (Cormode et al., 2012).

2.3.3 Summary of Approximate Query Processing

Approximate query processing has emerged in order to deal with extensive volumes of data efficiently instead of the traditional OLAP. The approach utilised for approximate query processing is mainly divided into online query processing and pre-computed synopsis. Although the online query processing is a useful technique in terms of the control of query processing, decision makers do not need control of query processing but need fast query processing with acceptable accuracy. Therefore, the pre-computed synopsis approach is more suitable rather than online query processing. The pre-computed synopsis is generally classified into three techniques, namely histograms, wavelets and samples.

The first technique, the histograms, is widely adopted in the database engines. However, it does not handle high-dimensional data reasonably. The multidimensional models in data warehouses are generally made up of approximately 8 to 15 tables (Kimball et al., 2011). Consequently, the histogram approach, which does not handle high-dimensional data appropriately, is not suitable for approximate query processing in decision support systems.

The second technique, wavelets, can handle high-dimensional data appropriately. However, even a Haar wavelet, which is the simplest method in wavelets, is a highly complicated mechanism, and useful feedback for decision makers such as an offer of error estimates of the approximation cannot be simply provided. Consequently, the wavelets are also not the best choice for approximate query processing in decision support systems.

The third technique, sampling, can handle high dimensional data appropriately and can also provide error estimates of the approximation. Therefore, compared with other techniques, sampling techniques are the most likely candidate as an effective and practical technique for approximate query processing in decision support systems. However, sampling has some issues in database environments. First, block-level sampling is preferred in terms of efficient disk I/O. Although a sample drawn by block-level sampling is not unbiased, the drawback can be solved if random sample sets are created by pre-computing before executing query processing. Second, it is clear that drawing a small sample is faster processing than a large sample. Although a large sample can improve the level of accuracy under highly selective query conditions, it loses the benefit from a small sample. Therefore, a method that enables an acceptable level of accuracy using small samples in any queries is needed.

The following Section 2.4 describes an index structure that focuses on “fast first” sampling from a range predicate. Also, Section 2.5 presents an indexing scheme to support sampling-based approximate query processing that focuses on efficiency and accuracy in any highly selective query.

2.4 ACE Tree

2.4.1 ACE Tree Structure

The ***ACE tree*** proposed by Joshi and Jermaine (2008a) is a binary tree data structure. It is composed of internal nodes and leaf nodes storing actual data. The internal nodes have several elements to achieve an efficient search. Each internal node contains a range R of key values, a split point k dividing the range R into two partitions, and a left pointer ptr_l and right pointer ptr_r to connect to next level nodes. In addition, the internal node holds a left counter cnt_l and right counter cnt_r which are the number of database records falling in the left and right child nodes.

Figure 2.13 illustrates an example structure of the ACE tree. A jth internal node at level i is denoted by $I_{i,j}$. The root node with a range $I_{1,1}.R = [0-80]$ indicates the key range of the entire data set. The root node divides the $I_{1,1}.R$ into $I_{2,1}.R = [0-40]$ and $I_{2,2}.R = [41-80]$. Likewise, each internal node splits its own range into two descendants. A leaf node consists of $(i+1)$ sections associated with ranges on the path from the root node to the leaf node. As shown by leaf node L_4 in the figure, the range of each section reveals each value encountered on the path, 0-80, 0-40, 21-40 and 31-40. Consequently, L_4S_1 has a random sample set without range restriction, L_4S_2 has a random sample set between 0 and 40, L_4S_3 has a random sample set between 21 and 40, and L_4S_4 has a random sample set between 31 and 40 (the most restricted range). The inclusion relation of sections is always satisfied with $L.S_1 \supset L.S_2 \supset \cdots \supset L.S_{(i+1)}$ from the viewpoint of the data range.

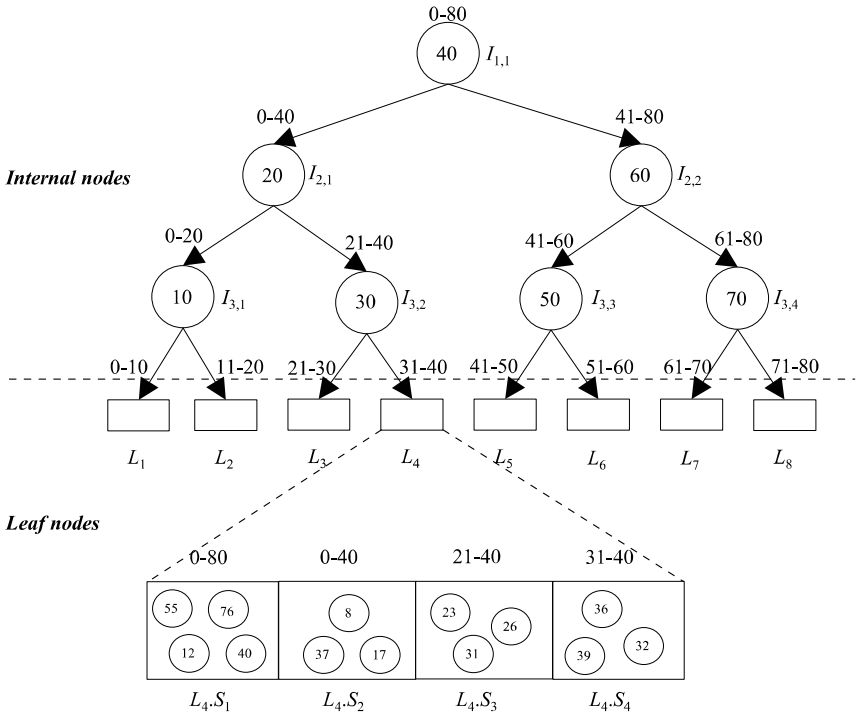


Figure 2.13 Structure of the ACE Tree

2.4.2 Query Execution

The retrieval process of the ACE Tree is to traverse the tree from a root node to leaf nodes recursively. In each internal node, if a query condition overlaps the split point, the processing follows the toggled indicator which points to a left or right child in turn. Otherwise, the processing traverses down to a left or right child which belongs to the query condition. After the processing reaches a leaf node, it recommences from the root node and repeats the above process until sufficient samples are obtained.

For example, consider a query with a range between 33 and 47 denoted by $Q = [33 - 47]$. As shown in Figure 2.14, L_4 is obtained in the first retrieval process. Since $L_4.S_1$ satisfies the query Q, records from section $L_4.S_1$ are filtered by Q and returned to the user immediately. The other sections $L_4.S_2,\ L_4.S_3,\ L_4.S_4$ are stored in a memory for a later retrieval process because the range of each section does not satisfy Q completely. Similarly, L_5 is obtained in the second retrieval process, and then the processing returns records with filtering from section S_1 to the user immediately. Moreover, section $L_5.S_2$ is combined with the section

$L_4.S_2$ to obtain records in the range [0 - 80] that satisfy Q. This combined sample is filtered by Q and returned to the user. The sections S_3, S_4 are also combined and returned by the same approach. Thus, this algorithm can provide fast response and relevant records.

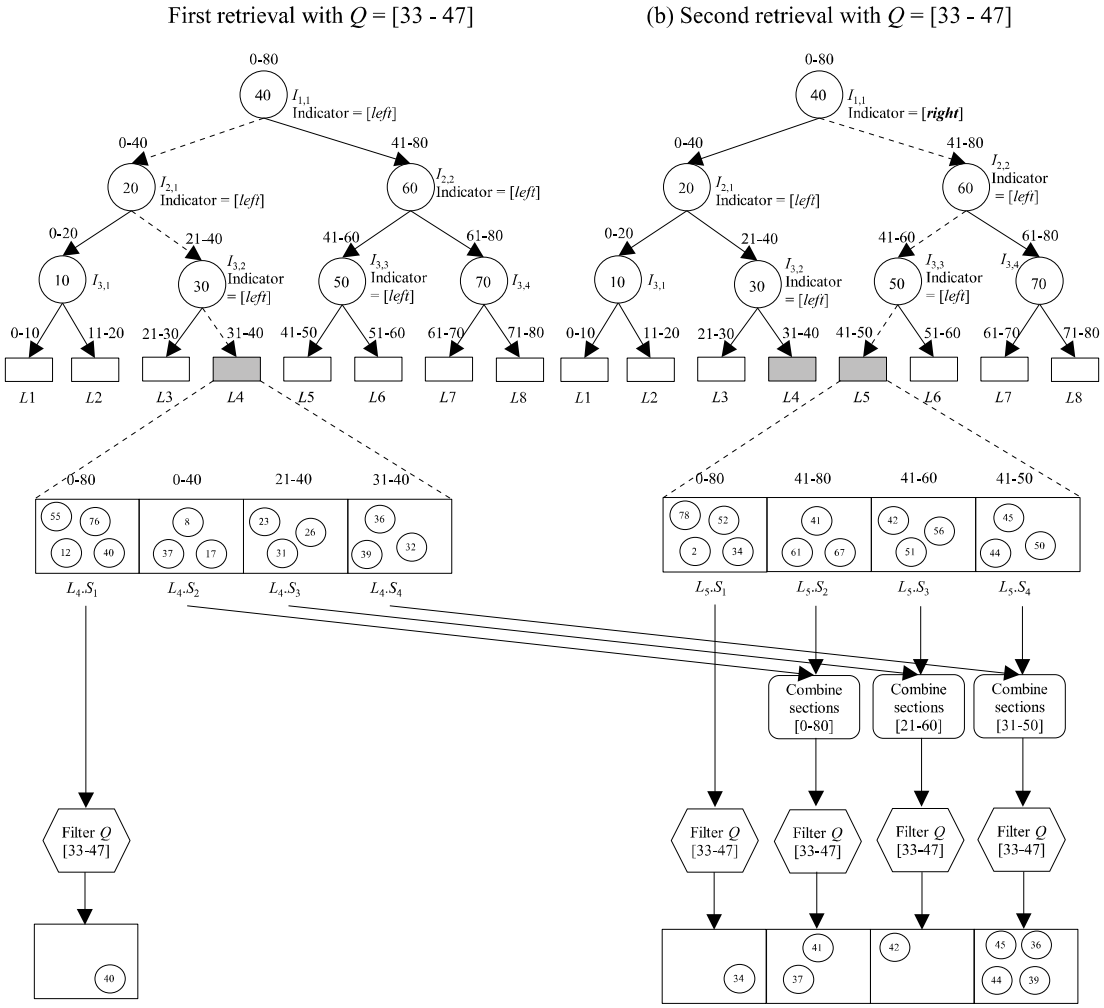


Figure 2.14 Retrieval process of the ACE Tree – An example

2.4.3 Advantages and Disadvantages

There are some advantages in the ACE tree structure. First, this structure can achieve fast block-level sampling from a database file in the process of drawing a random sample because the actual data in each section of each leaf node are packed and stored in multiple

disk pages. Second, an unbiased estimator can be applied because each section that stores actual data is statistically a random dataset of database records. Finally, samples obtained from the ACE tree have relevance to the query conditions if the section number is not one. Therefore, this sampling technique using the ACE tree might not miss relevant data in even small samples.

On the other hand, there are some disadvantages in the ACE tree. First, all data must be stored in the ACE tree. As a result, distributed processing is impossible without data replication elsewhere. This might be a shortcoming for some applications. Second, it is not easy to update the ACE tree structure without rebuilding it when new data are added. Finally, Joshi and Jermaine (2008a) also proposed multidimensional ACE trees using a k-d binary tree instead of a regular binary tree. However, this approach can cause the depth of the tree to become very large, even with a relatively small number of dimensions (Rudra et al., 2012). As a result, the retrieval time will be quite long. This shortcoming is linked to a motivation for a new index structure called the k-MDI tree.

2.5 k-MDI Tree

2.5.1 k-MDI Structure

To address the shortcoming of the ACE tree, Rudra et al. (2012) proposed the k-MDI tree. The ***k-MDI tree*** is a k-ary balanced tree data structure based on the ACE tree. As shown in Figure 2.15, it is also composed of internal nodes and leaf nodes. Each level i in the internal nodes corresponds to a key attribute (dimension) A_i in the index. Therefore, the height of this tree is limited to the number of key attributes. As a result, the k-MDI provides an efficient index search compared with the ACE tree. Also, each dimension A_i divides the data range R_i of the dimensions into P_i partitions which may be different from other dimensions. In addition, internal nodes have P_i descendants. With regard to leaf nodes, there are $(i + 1)$ sections in a leaf node; that is the same as the ACE tree structure apart from a data range concept which is multiple data ranges. Consequently, section 1 contains a random sample set without range restriction. Other sections k $(2 \leq k \leq i + 1)$ contain a random sample set belonging to a certain multidimensional range, which is the path from the root node to the

level k above the leaf node. Thus, all the data in this index tree are divided into $(\prod_{k=1}^{i} P_k) \times (i+1)$.

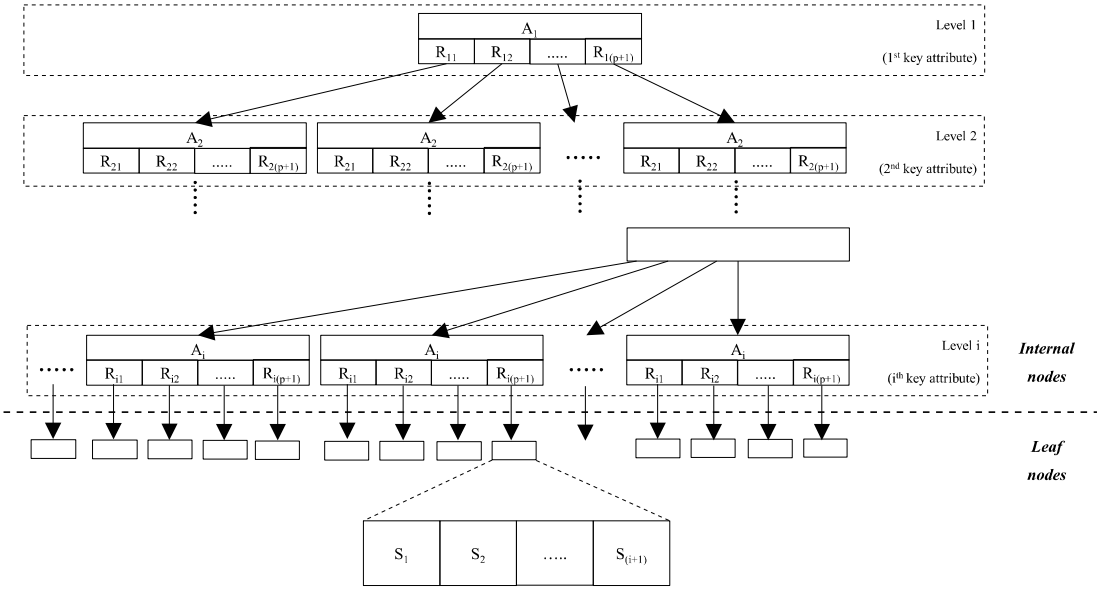


Figure 2.15 Structure of the k-MDI Tree

2.5.2 Query Execution

The primary aim for the use of the k-MDI tree is to achieve fast and efficient processing and to obtain many relevant records to query conditions as samples. Similar to the ACE tree, the retrieval process using the k-MDI tree explores the tree from a root node to leaf nodes recursively. The most important retrieval process is that key attributes for comparison vary at each level. Figure 2.16 depicts an example of the retrieval process of the k-MDI tree by a query $Q = ([Date: 12-18], [Item\ No: 50-150], [Store\ No: 15-20])$. This search algorithm starts from the root node, level 1. The query conditions Q are compared with the *Date* ranges of the first attribute index. The first query condition $[Date: 12-18]$ fits only into a middle range. The processing traverses down to level 2 as indicated by the dashed arrow. The next query condition which is a different key attribute and a value range are compared at level 2. In this example, the query Q overlaps with the first and second range. These ranges are marked for later retrieval. This retrieval explores level 3 in the same way as before, and L_8 is obtained. The retrieval process repeats continuously from the previously

marked point at level 2 until L_{10} is obtained. The sample records contained in these leaves are filtered by Q and returned to the user. The important point in this filtering stage is that valid sections for samples depend on aggregate functions. As mentioned before, although section S_1 consists of random samples without range restriction, other sections are selective random data subsets. Therefore, all sections can be used as samples for mean values; however, only section S_1 can be used as samples for total values or the numbers of records.

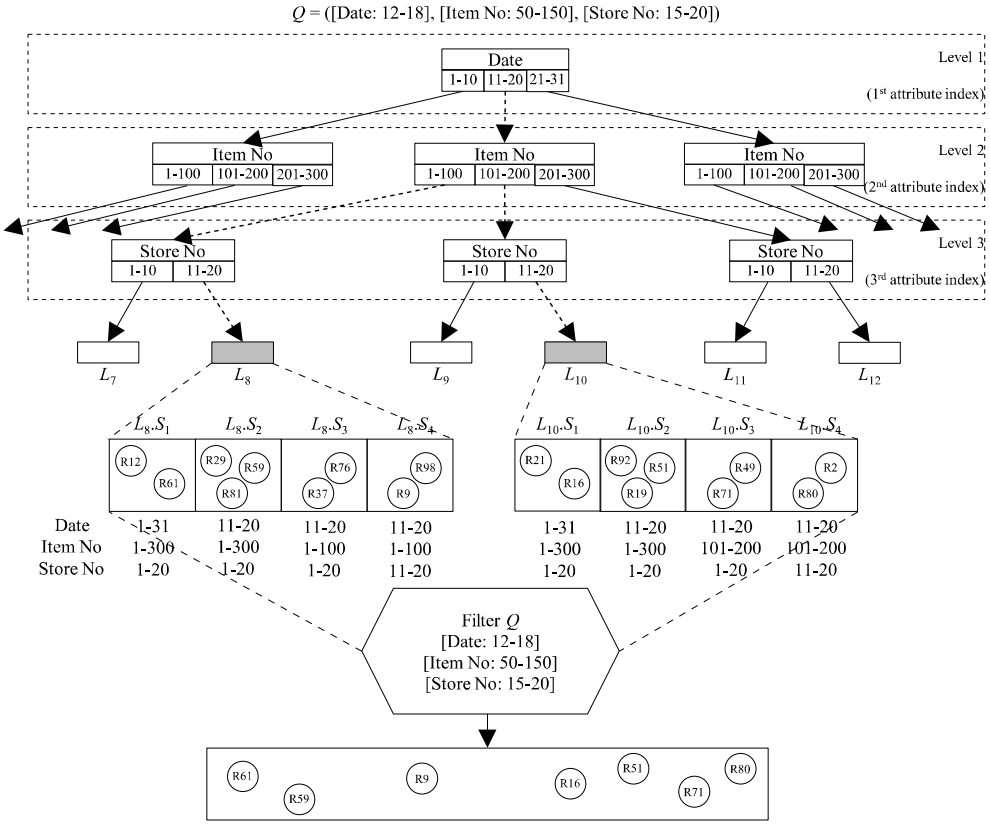


Figure 2.16 Retrieval process of the k-MDI Tree – An example

2.5.3 Advantages and Disadvantages

The k-MDI tree was proposed as it inherited the advantages of the ACE tree. Also, the k-MDI tree overcomes one of the shortcomings of the ACE tree. The k-MDI can be built with the shallow tree structure in even the high dimensions; the number of disk I/O to traverse the internal nodes can be smaller than the ACE tree. As a result, the k-MDI tree can achieve an

efficient index search compared with the ACE tree. Furthermore, the multi-way branching allows flexible range splits. For example, the ACE tree always assigns 2^n split points on each dimension. In contrast, the k-MDI tree can assign the arbitrary number of split points on each dimension.

However, there are unclarified points and disadvantages in the k-MDI tree structure. First, it is not clear that the k-MDI can handle the high dimensional data appropriately. Rudra et al. (2012) empirically demonstrated the capability with up to three dimensions, whereas the number of dimensions is over 10 in many practical situations. Second, Rudra et al. (2012) did not show its feasibility. The implementation methodology for complicated construction and query execution cannot be seen in the paper. Finally, the proposed query processing using the k-MDI cannot draw sufficient samples under a certain query condition. This problem is discussed in detail in Chapter 3.

2.6 Chapter Summary

In this chapter, the basic concepts and definitions of terms to this thesis are provided. Then, the chapter explains the background and terms of approximate query processing, and two approaches are introduced for the approximate query processing in detail. In pre-computed synopsis, there are three main techniques, and these techniques are discussed. After that, the ACE tree, which is a new index structure to support online random sampling, is described and discussed. Finally, the k-MDI tree extending the ACE tree is reviewed.

CHAPTER 3

MULTIDIMENSIONAL CLUSTER SAMPLING VIEW

3.1 Introduction

Due to the ever increasing amount of information, query processing in data warehousing environments can take a long time. However, it is not necessary for decision makers to obtain absolute answers (Chakrabarti et al., 2001; Rudra et al., 2012). For instance, if absolute and approximate answers are $186,164 and $180,670 respectively, the approximate answer is usually an acceptable value for decision support. There are several proposed methods for approximate query processing. One of the most commonly used techniques is sampling, which is a very useful method to obtain results very quickly (Joshi & Jermaine, 2008a; Olken & Rotem, 1986; Rudra et al., 2012; Webb & Wang, 2014). A variety of recent proposals focusing on approximate query processing based on random sampling are studied with significant efforts (Jermaine, Arumugam, Pol, & Dobra, 2008; Joshi & Jermaine, 2008a, 2008b; Li et al., 2008; Rudra et al., 2012; Ruoming et al., 2006). However, the random sampling has a crucial drawback when the data have skewed distributions, or when query conditions are highly selective. The answers from unbiased samples such as by random sampling may miss relevant records in highly selective queries because of insufficient coverage (Joshi & Jermaine, 2008b; Rudra et al., 2012). Although a naïve method to solve the problem is to draw larger samples, the time required to draw the large samples may negate the advantage of sampling as an approximation method.

A new index structure called the Appendability, Combinability, and Exponentiality (ACE) tree has been proposed (Joshi & Jermaine, 2008a). The ACE tree is a very effective index tree structure that draws relevant sample records to query conditions from a database quickly; however, it does not deal with the multidimensional tree adequately. To support multidimensional predicates, they extended the ACE tree, using a k-d binary tree. However,

when this index is constructed with multi-attribute keys, the depth of the index tree becomes very deep (Rudra et al., 2012).

Rudra et al. (2012) proposed a multidimensional index tree called the k-ary multidimensional index (k-MDI) tree, as an extension of the ACE tree. This index has been shown to be an effective sampling scheme for decision support with up to three dimensions. The depth of the index tree is very shallow, being equal to the number of dimensions. Therefore, the amount of disk access needed to traverse internal nodes in the k-MDI tree is small. However, Rudra et al. (2012) do not show the feasibility of the method with an implementation example. It is complicated to implement the k-ary index tree structure in tables of a database and to execute approximate query processing using the k-MDI tree in SQL.

In this chapter, the Multidimensional Cluster Sampling View based on the k-MDI is proposed. The view which includes an index for effective random sampling and clusters storing actual data for efficient extraction of a sample can be implemented with ease using common database tables and can be manipulated by SQL statements. In addition, it enables efficient and effective approximate query processing in SQL. The query processing can draw samples by a cluster unit for efficient data access. At the same time, it can draw sufficient and relevant samples to query conditions. Also, the effectiveness of approximate query processing using the view is empirically evaluated on a large table with three dimensions.

The rest of this chapter is organised as follows. Section 3.2 describes the implementation method of the Multidimensional Cluster Sampling View. In Section 3.3, approximate query processing using the view is described. Section 3.4 presents and discusses results from the experiment. Finally, this chapter is summarised in Section 0.

3.2 Implementation

3.2.1 Our Approach

The k-MDI tree facilitates extraction of relevant sample records to query conditions. However, the feasibility of the k-MDI tree with an implementation mechanism is not

discussed in Rudra et al. (2012). In addition, query processing using the k-MDI has a shortcoming of not being able to draw sufficient sample records under a condition. When a query condition falls in only one leaf, the query processing cannot obtain many sample records.

The ***Multidimensional Cluster Sampling View*** is a hybrid view with an index for approximate query processing, which is based on the k-MDI structure and a set-theoretical approach. All sections of the k-MDI are decomposed into random data subsets with leaf number, section number, data range property and size property called Cluster Samples as shown in Figure 3.1. In this example, there are 48 Cluster Samples which are composed of 12 leaves and four sections similar to leaf nodes of the ACE tree and the k-MDI. Section one ($L.S_1$) is a random data subset without range restriction, and other sections ($L.S_2$, $L.S_3$, $L.S_4$) are a random data subset with specific ranges. The view is composed of an index table and a cluster table. The index table contains identification values configuring the Cluster Samples, namely, leaf number, section number, data range property and size property. The cluster table contains actual data and key attributes linking to the index table, namely, leaf number and section number.

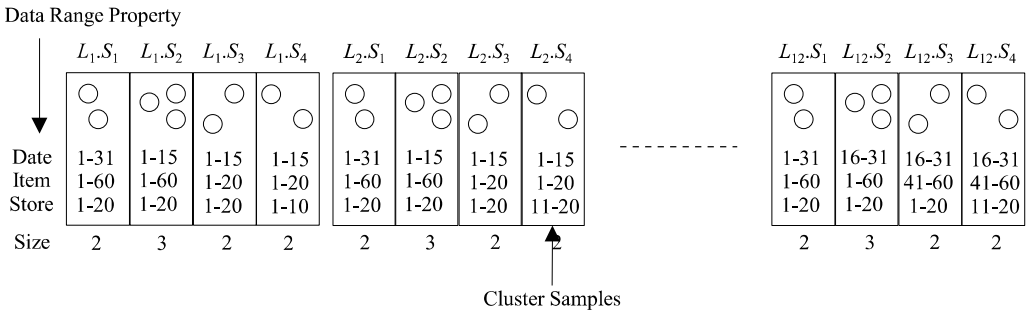


Figure 3.1 An example of Cluster Samples

Online random samples can be easily obtained by scanning the data range property in the index table once and then accessing the cluster table. In addition, more relevant samples to query conditions can be drawn by selecting a larger section number. The Cluster Samples with the larger section number belong to more selective data ranges. This method is an effective online sampling scheme in a highly selective query environment with any given arbitrary range.

The file organisations supporting random sampling in database environments could be one of the interesting themes of research. In order to reduce the cost of disk access from a database for sampling, Hou, Ozsoyoglu, and Taneja (1988) introduce the use of cluster sampling, which uses all of the information on each disk page as a sample unit instead of a tuple. In Olken and Rotem (1995), a comprehensive analysis of various methods for random sampling from databases are discussed. Although the samples from each disk page by cluster sampling are not necessarily independent (Olken & Rotem, 1995), the partitioning strategy is very useful in terms of a method for efficient disk access. Therefore, the method which clusters data is adopted in one of the implementation approaches for efficient query processing.

3.2.2 Construction of Multidimensional Cluster Sampling View

In this section, a conceptual idea for construction of the Multidimensional Cluster Sampling View is presented. The significant point regarding construction of the Multidimensional Cluster Sampling View is that a flat structure not a tree structure is adopted to achieve a simple implementation on database tables and efficient query processing in SQL. The construction proceeds in accordance with the following three steps.

First, as shown in Figure 3.2, the data are sorted on the first key values, and then split points on the key attribute are identified in order that each partition size is approximately the same proportion. This processing is repeated on all targeted key attributes. At this time, the number of partitions affecting the number of Cluster Samples should be limited to some extent because a huge number of Cluster Samples are not efficient for query processing. If there are 1,000,000 Cluster Samples, it is clear that the cost to search the Cluster Samples is expensive.

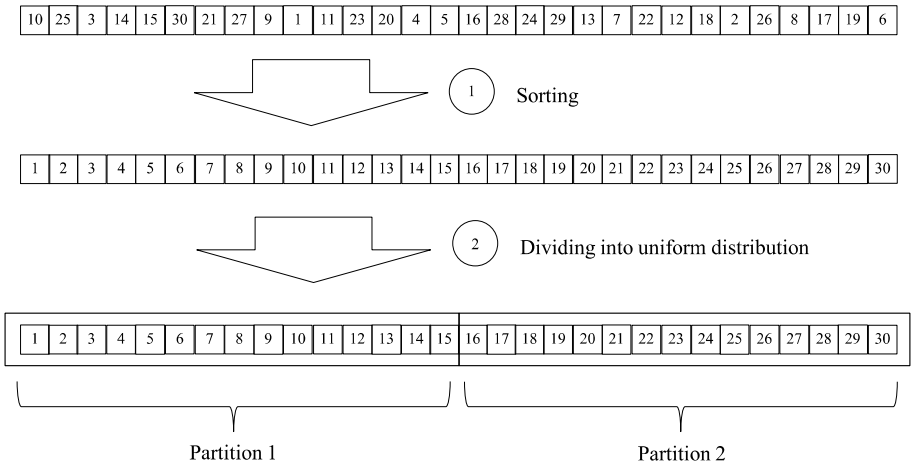


Figure 3.2 First step for construction of the Multidimensional Cluster Sampling View

Second, frames of Cluster Samples are created in the view as shown in Figure 3.3. Property information of the Cluster Samples is initialised, and then necessary sequential spaces to store actual data are allocated. The number of leaves is the product of the number of partitions on each key attribute. Each leaf is divided into the number of sections, one plus the number of key attributes. As is the case with the k-MDI, the largest section numbers possess the most selective data range, namely, section four in this example. Section one is a data set without range restriction, namely, Date range between 1 and 31, Item range between 1 and 60, and Store range between 1 and 20 in this example.

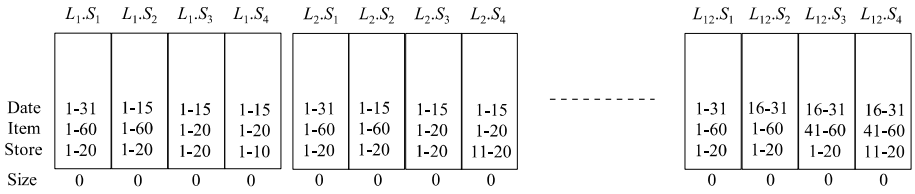


Figure 3.3 Second step for construction of the Multidimensional Cluster Sampling View

Finally, all Cluster Samples are generated from an original table. As shown in Figure 3.4, initially a record is picked from the original table, and then a section number is assigned randomly. Next, the clusters which possess the same section number and the data range property containing the all key values of the picked record are extracted. Then, the record is randomly stored in an extracted cluster. As a result, records in section one of Cluster

Samples are random samples with no data range restriction, and records in other sections are random samples belonging to a specific data range. Lastly, the size property of the cluster is incremented by one. This step is repeated until the last record. However, since one record processing like this is very inefficient in practice, this processing should be executed by bulk processing.

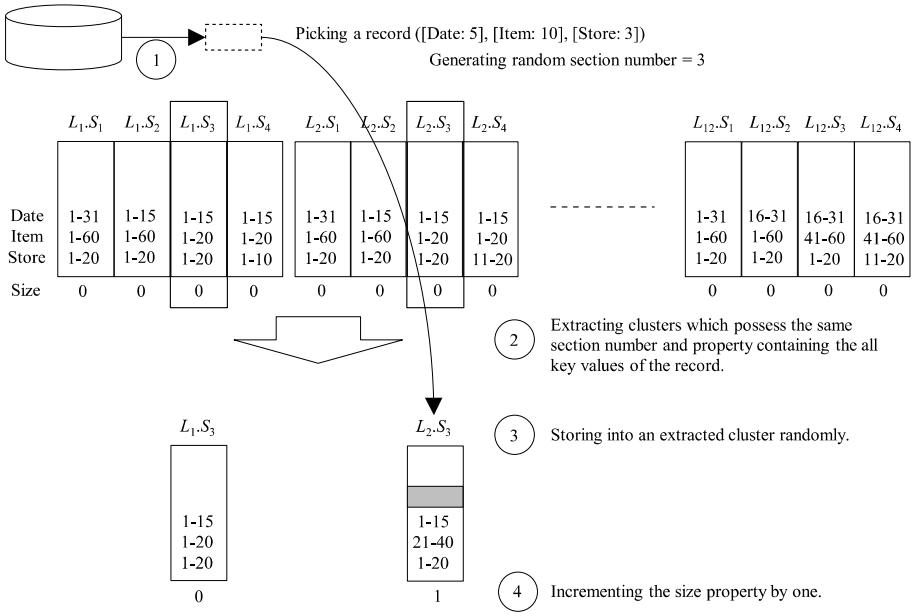


Figure 3.4 Third step for construction of the Multidimensional Cluster Sampling View

3.3 Approximate Query Processing

3.3.1 Query Execution in Multidimensional Cluster Sampling View

The purpose of the Multidimensional Cluster Sampling View is for efficient extraction of samples and effective approximation of query results. The conceptual process of query execution in the Multidimensional Cluster Sampling View with an example is described.

Let $Q = ([Date: 5 - 10], [Item: 15 - 30], [Store: 3])$ be the example query condition. The following SQL statement represents this query condition:

```
SELECT AVG(SALES_AMOUNT) FROM SALES
WHERE (DATE BETWEEN  5 AND 10)
AND   (ITEM BETWEEN 15 AND 30)
AND   (STORE = 3).
```

As shown in Figure 3.5, there are three key attributes, Date attribute, Item attribute, and Store attribute. The Date attribute has two partitions, 1 to 15, and 16 to 31, and the Item attribute has three partitions, 1 to 20, 21 to 40, and 41 to 60. The Store attribute has two partitions, 1 to 10 and 11 to 20. In addition, the Cluster Samples are composed of 12 leaves and 4 sections. In other words, there are 48 Cluster Samples, and the size of one cluster is approximately 2% of whole data.

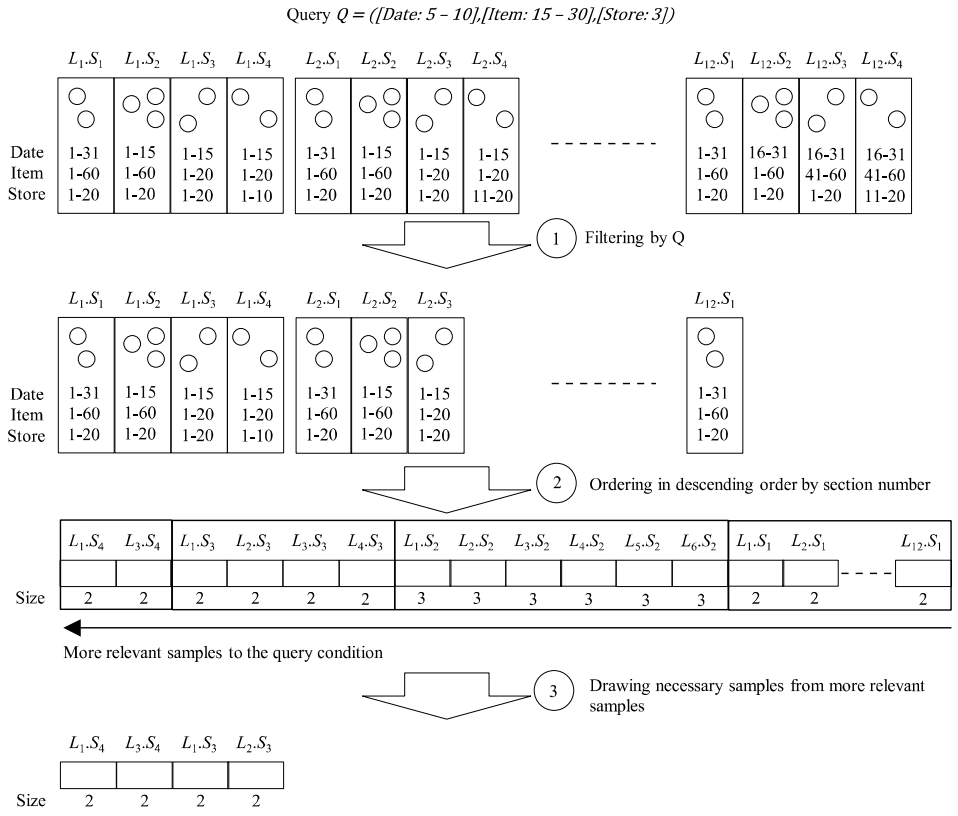


Figure 3.5 Query execution process using the Multidimensional Cluster Sampling View

First, the processing filters the data range property of Cluster Samples by the query condition. Unlike the k-MDI tree, since the data range property of all Cluster Samples is compared, more samples can be obtained even if query conditions are extremely selective. In this example, at least 25% of the whole data, all section one, can be obtained as samples. This

also contributes to solving one of the shortcomings of the k-MDI, which is to draw an insufficient sample in highly selective queries. As a point to be noted, if the approximate query processing is for the total number of records or total values, only section one can be used as samples for the estimator. This will be explained later in Section 3.3.3. Otherwise, all sections can be used as samples.

Second, the query processing rearranges the filtered clusters in descending order by section number. As mentioned before, sample records in the larger section number belong to the more specific data range. Therefore, random samples in the larger section number filtered by the query Q are more relevant to the condition. This can prevent missing relevant records to query conditions in any highly selective queries.

Finally, the query processing draws necessary samples from the head of the sorted clusters by using the size property of the clusters. Obtaining a sample of some pre-determined percent can be easily achieved, just to store the whole data size into a table when the Multidimensional Cluster Sampling View is created.

3.3.2 Sufficient Sample Size

The search algorithm using the k-MDI proposed by Rudra et al. (2012), in fact, has a shortcoming. The number of leaf nodes which is obtained from the k-MDI tree for a sample depends on query conditions. If a query condition does not overlap with any split point, the number of leaves which is obtained from the k-MDI tree is only one. This means that at most $\frac{1}{2^i}$ samples of the whole data with i dimensions are obtained in such a query condition. For instance, the sample size in four dimensions is at most 6.25% under the condition. If the number of dimensions is over seven, the sample size is less than 1% under such a condition. As a consequence, users cannot obtain adequate samples. Needless to say, the sample size diminishes even further if the number of partitions on each attribute is over two.

However, the Multidimensional Cluster Sampling View can provide more samples than the k-MDI because the algorithm described in Section 3.3.1 compares data range properties of all Cluster Samples with query conditions provided by the users. For instance, in the case of Figure 2.16, the algorithm obtains section S_2 and S_3 in leaf L_7 and L_9 as a sample because

these sections include the query condition. Moreover, section S_2 in leaf L_{11} and L_{12} can be sampled for the same reason, and section S_1 in any leaves can be sampled.

It is obvious that this algorithm can provide sufficient samples under any query conditions. When there are two partitions on each key attribute, the sample size can be at least

$$(i+1)+\left(\sum_{x=1}^{i}(i+1-x)2^{x-1}\right) \tag{3.1}$$

where i denotes the number of dimensions. For instance, consider a query condition falling in a leaf of the k-MDI, which is composed of seven dimensions with two partitions on each key attribute. The number of leaf nodes in the k-MDI is 128. When compared with the k-MDI, the query processing using the Multidimensional Cluster Sampling View can improve the maximum sample size from 0.78% to 24.9% on seven dimensions. In the case of 15 dimensions with two partitions on each key attribute, query processing using the Multidimensional Cluster Sampling View can also improve the sample size from 0.003% to 12.8% under certain conditions.

3.3.3 Sampling-Based Estimators

In this section, a calculation method for estimating the approximate answer from samples is provided. Approximation is estimated using samples from Cluster Samples. Since all Cluster Samples are random data sets, unbiased estimators are needed. An average value and the number of records obtained from the samples are required to estimate a total value from unbiased samples. The estimation of average values can be calculated simply from the sampled records. With regard to the estimated number of records, A. Chaudhuri and Mukeriee (1985) studied unbiased estimators for the size, mean values and total values. Let N be the number of records in a whole dataset, and m be the number of sampled records satisfying query conditions. Let n_1 be the number of sampled records in section one, and m_1 be the number of sampled records satisfying query conditions in section one.

An approximation of the number of records is given as follows:

$$\widehat{M} = N\frac{(m_1 - 1)}{(n_1 - 1)} \tag{3.2}$$

An approximation of mean values is given as follows:

$$\bar{x} = \frac{\sum_{k=0}^{m} Value_k}{m} \tag{3.3}$$

An approximation for total values is given as follows:

$$\hat{T} = \widehat{M}\,\frac{\sum_{k=0}^{m_1} Value_k}{m_1} \tag{3.4}$$

3.4 Evaluation

3.4.1 Overview

To evaluate the effectiveness of the Multidimensional Cluster Sampling View, two experiments on a large table with three dimensions in the Oracle Database 11g Release 2 were conducted. A big table was prepared by a tool of the TPC Benchmark Decision Support (TPC-DS) (TPC, 2014), which combines both real world and synthetic data including many tables and columns (Nambiar & Poess, 2006; Poess, Nambiar, & Walrath, 2007).

There were approximately 144 million rows in the table named *CATALOG_SALES*, and the table size was approximately 25 Gigabytes. As shown in Table 3.1, the Multidimensional Cluster Sampling View for the experiments was simulated with six partitions in each key attribute, namely, *CS_SOLD_DATE_SK*, *CS_BILL_ADDR_SK* and *CS_ITEM_SK*; therefore, the view was composed of 864 Cluster Samples with 216 leaves and 4 sections. This means that the size of one Cluster Sample is approximately 0.116% of whole data. Also, the measure attribute named *CS_NET_PAID* is used in this evaluation.

Table 3.1 Partition information of the Multidimensional Cluster Sampling View with 3 dimensions in Section 3.4

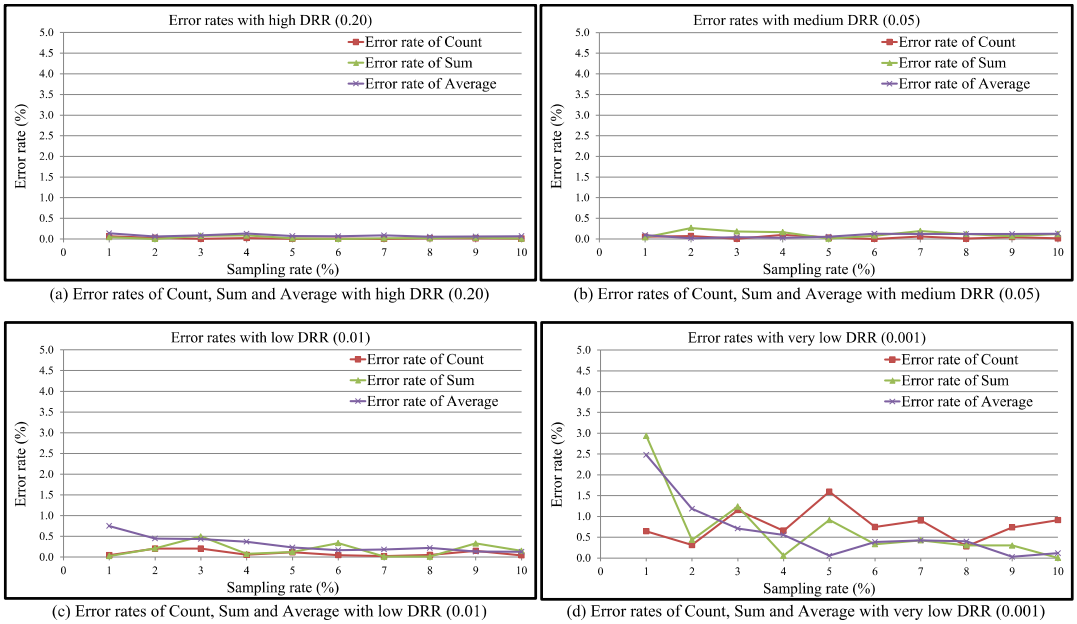
	Minimum value	1st split point	2nd split point	3rd split point	4th split point	5th split point	Maximum value
CS_SOLD_DATE_SK	2,450,815	2,451,151	2,451,487	2,451,809	2,452,131	2,452,405	2,452,654
CS_BILL_ADDR_SK	1	167,401	334,812	502,298	669,895	837,440	1,000,000
CS_ITEM_SK	1	34,002	68,007	102,001	136,004	170,001	204,000

3.4.2 Relevancy Ratio

Rudra et al. (2012) have introduced two measures to evaluate query results by sampling. The first measure called the ***database relevancy ratio*** (***DRR***) is the ratio of the number of records satisfying query conditions in a complete data set. It is represented by $\rho(Q)$ on a query Q. For instance, when there is no query condition, the DRR $\rho(Q)$ is 1. If a query condition narrows down the whole data to two, the DRR $\rho(Q)$ is 0.5. This measure is appropriate to define how much query conditions are selective. The second measure called the ***sample relevancy ratio*** (***SRR***) is not used in this paper.

3.4.3 Query Effectiveness

For the first experiment, the accuracy of approximate query processing using the Multidimensional Cluster Sampling View was examined on a large dataset with three dimensions. This experiment was also conducted with a variety of selectivity for the query, namely, high DRR (0.20), medium DRR (0.05), low DRR (0.01) and very low DRR (0.001). Also, three aggregate functions, which are SUM, COUNT and AVERAGE, were used since other functions such as MAX and MIN are extremely difficult in sampling based estimation. Figure 3.6 includes four line graphs that illustrate the error rates between absolute answers and approximate answers using the Multidimensional Cluster Sampling View with various DRR on 3 dimensions. The mean value of 10 results on each sampling rate is used as the error rate.



(a) Error rates of Count, Sum and Average with high DRR (0.20)

(b) Error rates of Count, Sum and Average with medium DRR (0.05)

(c) Error rates of Count, Sum and Average with low DRR (0.01)

(d) Error rates of Count, Sum and Average with very low DRR (0.001)

Figure 3.6 Error rates between absolute answers and approximate answers using the Multidimensional Cluster Sampling View with various DRR among three aggregate functions on 3 dimensions from a sampling rate of 1% to 10%

Figure 3.6a and Figure 3.6b illustrate error rates on high DRR (0.20) and medium DRR (0.05); all error rates on all sampling size are under 0.2%. Since this result shows almost an exact answer, approximation using the Multidimensional Cluster Sampling View can be reliable on high DRR (0.20).

Figure 3.6c illustrates error rates on low DRR (0.01), the error rates of the number of records are at most 0.2%, whereas the error rates of total value and average value are at most 0.5% and 0.75% respectively. This is also acceptable accuracy for approximation.

Figure 3.6d illustrates error rates on very low DRR (0.001), and the error rates of all functions, which are COUNT, SUM and AVERAGE, with 1% samples are at 0.64%, 2.94% and 2.48% respectively. They are the highest errors except for COUNT. However, this is still a valuable result for approximate query processing.

In summary, the error rates of all aggregation functions with sample size between 1% and 10% are under 3% at all DRRs. These error rates are acceptable values as a result of approximate query processing. Furthermore, the error rates decline gradually with increases

of sampling rates as expected. These results show that the Multidimensional Cluster Sampling View is competent for the practical use of approximate query processing in terms of high quality approximation.

3.4.4 Sample Size Effectiveness

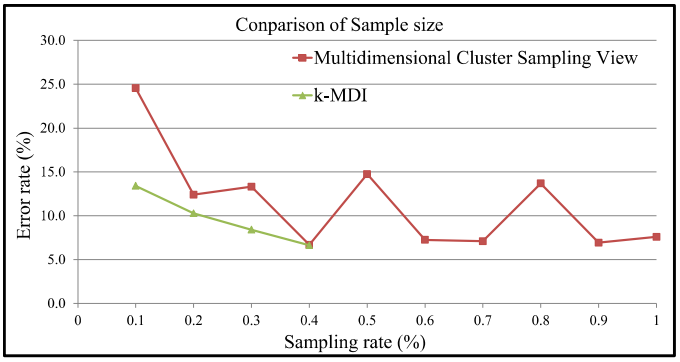
For the next experiment, the improvement of sample size by the Multidimensional Cluster Sampling View was examined compared with the k-MDI. In this experiment, the same Multidimensional Cluster Sampling View was used as for the k-MDI. In terms of leaf nodes and section nodes, both clusters are the same. The difference is query processing. The k-MDI is a tree structure, whereas this experiment focuses on how many samples each method can provide. Therefore, it is not necessary to simulate the exact query processing of the k-MDI. The processing that the k-MDI could provide final samples under a certain query condition was implemented. To explain the processing using Figure 3.5, the Cluster Samples are filtered by the query conditions. By filtering the last section number, the processing can obtain some leaf nodes same as the k-MDI tree. Since the total number of records is targeted in this experiment, pairs of the above leaf numbers and fixed section one are keys for samples of the k-MDI. The subsequent processing, drawing necessary samples, is the same as the Multidimensional Cluster Sampling View.

Moreover, an extremely selective query, 0.0004 DRR, was used with the COUNT function on three dimensions. The following SQL statement is for an absolute answer:

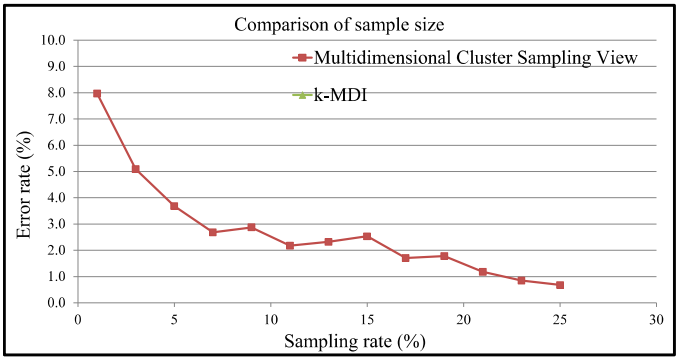
```
SELECT COUNT(*) FROM CATALOG_SALES
WHERE (CS_SOLD_DATE_SK BETWEEN 2451750 AND 2452000)
AND   (CS_BILL_ADDR_SK BETWEEN  500000 AND  550000)
AND   (CS_ITEM_SK      BETWEEN   80000 AND   90000).
```

From Table 3.1, the query condition belonged to two partitions on the first and second key attributes and to one partition on the third key attribute. As a result, in theory, the k-MDI and the Multidimensional Cluster Sampling View can provide samples for the total number of records, at most 4 sections (clusters) and 216+4 clusters respectively.

Figure 3.7 contains two line graphs that depict the obtainable sample size and error rates using the Multidimensional Cluster Sampling View and the k-MDI under the above query condition. On the one hand, the maximum sampling rate using the k-MDI algorithm is at 0.4% in the same manner as theory, and approximate query processing using the k-MDI provides over 5% error at all times in the Figure 3.7a. On the other hand, accuracy of the approximate answers in the Multidimensional Cluster Sampling View tends to improve with increasing the sample size. The query processing in this view draws more samples up to 25%, and it provides less than 5% error with more than 3% samples. It is clear that the Multidimensional Cluster Sampling View is more capable in any queries than the k-MDI as a method of approximate query processing in databases.



(a) Sample size between 0.1% and 1%



(b) Sample size between 1% and 25%

Figure 3.7 Sample size and error rates using the Multidimensional Cluster Sampling View and the k-MDI with extremely selective query on 3 dimensions

3.5 Chapter Summary

In this chapter, a novel view called the Multidimensional Cluster Sampling View was proposed for practical use, which facilitates drawing online random samples on database tables efficiently and effectively. In query processing using the view, many relevant sample records to query conditions can be drawn in any query conditions compared with the k-MDI. In addition, the excellent ability of the Multidimensional Cluster Sampling View for approximate query processing was demonstrated on three key attributes, and it was empirically demonstrated that it provided sufficient samples.

CHAPTER 4

DEVELOPMENT OF PRACTICAL UTILITY

4.1 Introduction

Online analytical processing (OLAP) is one of the essential components for decision support systems. However, the traditional OLAP tools often require a great deal of time (S. Chaudhuri, Das, & Narasayya, 2007; Ruoming et al., 2006). Approximate query processing based on random sampling is one of the most effective techniques to achieve fast query response times in a large dataset (Joshi & Jermaine, 2008a; Olken & Rotem, 1986, 1995; Rudra et al., 2012; Webb & Wang, 2014), whereas there is a serious drawback under a certain condition. Approximations based on random sampling with highly selective queries may provide unreliable results because the samples do not represent the entire data (Joshi & Jermaine, 2008b; Rudra et al., 2012).

For an effective technique of approximate query processing in the real world, the Multidimensional Cluster Sampling View is proposed in Chapter 3. The view facilitates efficient and effective online random sampling for approximate query processing in SQL, whereas its feasibility is not completely shown. In order to demonstrate a practical use, utility software that enables building the Multidimensional Cluster Sampling View with a few parameters and easily executes approximate query processing using the view in SQL was developed. This chapter presents the detailed and practical implementation method of the Multidimensional Cluster Sampling View and describes the approximate query processing using the view. Also, its practicality is empirically demonstrated through two evaluations.

This chapter is organised as follows. In Section 4.2, the overview design of the practical utility that was developed in this research is presented. Section 4.3 describes how the

software creates the Multidimensional Cluster Sampling View on databases, and Section 4.4 describes how the software executes approximate query processing using the view in SQL. In Section 0, the capability of the software which enables the creation of the view and execution of approximate query processing using the view is empirically evaluated on a large database with four dimensions. Section 4.6 summarises this chapter.

4.2 Design for practical software

The practical software that enables building the Multidimensional Cluster Sampling View with a few parameters and executing approximate query processing using the view was developed. In this section, the software design is described. The utility software is programmed in C# with Microsoft Visual Studio 2013 on .NET Framework 4.5 and is focused on the Oracle Database 11g Release 2 in this research.

4.2.1 Data Model

In this section, the data model which is used in the practical software is explained. Basically, data, except for software setting information, are stored in database tables.

Figure 4.1 illustrates relationships among tables that are used in the practical utility. There are 4 types of tables, *tab_repository*, *col_repository*, *leaf_repository* and *clusterN*. The full column definitions of each table are available in Appendix A. In the *tab_repository* table and *col_repository* table, information for user interface is stored; for instance, schema names, cluster table names, size, key and measure attribute names, data types, maximum and minimum values. There is a parent-child relationship between the two tables, and they are linked by a unique table number.

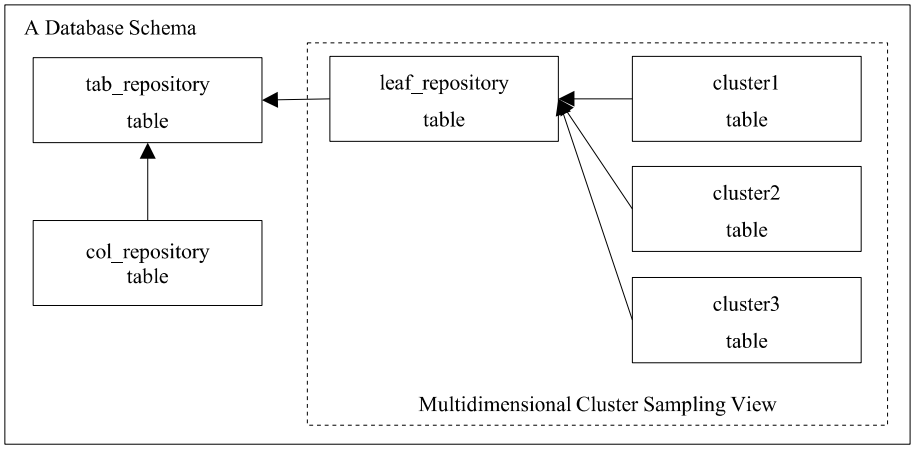


Figure 4.1 Relationships among tables used in practical utility

In the *leaf_repository* table, called 'index table' in the Multidimensional Cluster Sampling View, leaf numbers, section numbers, the number of records in a section, and data range property are stored. As this table is only one in the schema, the table is shared with all cluster tables stored as actual data, called *clusterN* in this example. Between these tables, there is also a parent-child relationship, and it is linked by three attributes of a unique table number, leaf number and section number. The detailed method for construction of the 'index table' and 'cluster table' will be described later in Sections 4.3.2 and 4.3.3 respectively.

4.2.2 Class Design

The practical software is written in C# language on .NET Framework 4.5. Therefore, in this section, class design of the utility software is presented to draw the big-picture view of class that configures the software.

This software adopts a programming model called Model-View-Presenter (MVP), which is proposed by Potel (1996). It is one of the software architectural patterns in the object-oriented programming approach. The class structure of the utility software is depicted in Figure 4.2. It can be seen that there are five core classes except the Program class of entry point. In addition, there are other classes that are not important to understand the software, such as UtilityTraceListener class, ProgressInformation class, ExtensionMethods class and

ListViewItemSorter class. These classes write a log file, encapsulate progress information, extend basic methods or sort item lists. In this section, the focus is upon the core five classes.

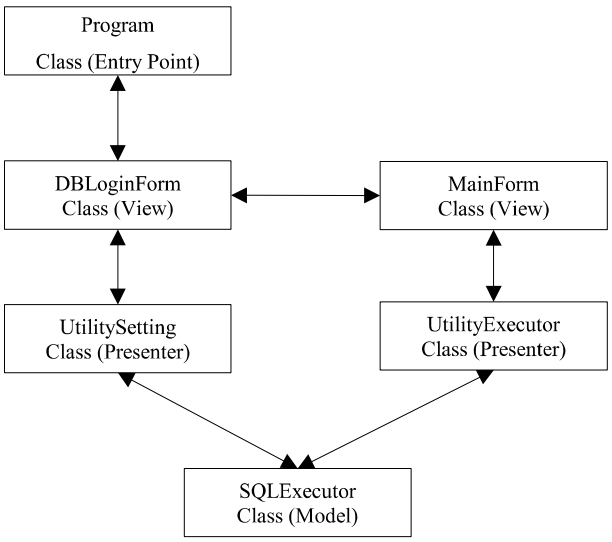


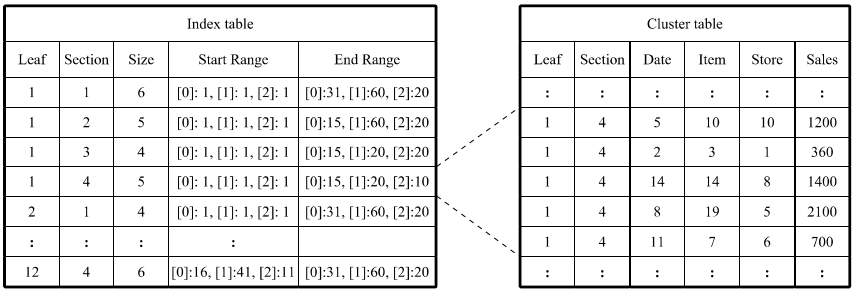
Figure 4.2 Class structure of utility software

The DBLoginForm class is in charge of the view of MVP. This class provides user interfaces for database login. Users input parameters to log in to a database via this form. The inputted parameters are notified to the UtilitySetting class, which is in charge of the presenter of MVP. The UtilitySetting class tests the correctness of the parameters using the SQLExecutor class, which is in charge of the model of MVP. If the parameters pass the check, the MainForm class is loaded. The form provides interfaces to users for creating the Multidimensional Cluster Sampling View and executing approximate query processing using the view. The UtilityExecutor class operates as a presenter, and the presenter class uses the SQLExecutor class to execute two main functions. The detailed behaviour of the two main functions will be presented later in Sections 4.3 and 4.4.

4.3 Creating Multidimensional Cluster Sampling View

4.3.1 Overview

The implementation of the Multidimensional Cluster Sampling View is mainly divided into two stages, index table construction and cluster table construction. The Multidimensional Cluster Sampling View can be simulated as shown in Figure 4.3. Each table has two columns in common, leaf number and section number. The index table has data range columns using variable-length multidimensional arrays. The array's indexes which start at zero indicate the order of key attributes.



Index table

Leaf	Section	Size	Start Range	End Range
1	1	6	[0]: 1, [1]: 1, [2]: 1	[0]:31, [1]:60, [2]:20
1	2	5	[0]: 1, [1]: 1, [2]: 1	[0]:15, [1]:60, [2]:20
1	3	4	[0]: 1, [1]: 1, [2]: 1	[0]:15, [1]:20, [2]:20
1	4	5	[0]: 1, [1]: 1, [2]: 1	[0]:15, [1]:20, [2]:10
2	1	4	[0]: 1, [1]: 1, [2]: 1	[0]:31, [1]:60, [2]:20
:	:	:	:	
12	4	6	[0]:16, [1]:41, [2]:11	[0]:31, [1]:60, [2]:20

Cluster table

Leaf	Section	Date	Item	Store	Sales
:	:	:	:	:	:
1	4	5	10	10	1200
1	4	2	3	1	360
1	4	14	14	8	1400
1	4	8	19	5	2100
1	4	11	7	6	700
:	:	:	:	:	:

Figure 4.3 Mapping the Multidimensional Cluster Sampling View

In order to achieve efficient data access, the cluster table adopts a partitioning technique, which divides a large table into several small tables possessing a specific data range (Lightstone, Teorey, & Nadeau, 2007). The partitioning technique can allow the simple managing of large database objects, and it is completely transparent to applications. Therefore, software or users can use one table and do not need to use specific SQL statements in query processing. In this case, leaf and section number are the partitioning keys, and the table is created using list partitioning because the number of the partitions is finite and fixed when the partitions are created.

4.3.2 Index Table Construction

There is a consideration for construction of the index table. It is the size of one Cluster Sample. Although Rudra et al. (2012) do not strictly discuss the leaf size of the k-MDI,

Rudra et al. (2012) suggested that the leaf size may be the same disk block size as the ACE tree or another suitable size. In general, the disk block size is 4KB, which depends on the file system of the operating system. The block size is very small on large databases. If the table size is 25GB, 6,553,600 leaves will be created on the simplified calculation. Also, the Oracle Database has a database block size, which is a configuration parameter used to determine the size of reading and writing a disk at the one time. However, the I/O size is at most 32KB, and it is still very small in large databases.

As shown in Figure 4.3, one record in the index table indicates one section in a leaf node, namely, one Cluster Sample. Therefore, when there are 6,553,600 leaf nodes with four dimensions, the index table contains approximately 32 billion records. The total number of sections is the product of the total number of leaf nodes and the number of sections in a leaf (one plus the number of dimensions). It is clear that the massive amount of data significantly affects the query response time when query processing retrieves the index table. Consequently, the total number of sections was limited in the present attempt. The index table construction is based on the following five steps.

At the first step, the total number of the records in an original dataset, the number of distinct values (NDV) on each key attribute, and minimum and maximum values on each key attribute are computed in SQL. The NDV of a key attribute, denoted by NDV_i where i denotes the attribute number, can be obtained in SQL as follows:

```
SELECT COUNT(DISTINCT Key1) FROM Table1.
```

The second step is that the number of partitions on each key attribute is computed in C#. The number of partitions on each key attribute denoted by P_i, the total number of leaves denoted by L, and the total number of sections denoted by S are defined as follows:

$$2 \leq P_i \leq NDV_i \tag{4.1}$$

$$L = \prod\nolimits_{k=1}^{i} P_k \tag{4.2}$$

$$S = (i+1)L \leq 1{,}000 \tag{4.3}$$

In the third step, a data range of partitions on each key attribute is determined in SQL. For example, when there are seven partitions on a key attribute, the data range of partitions is computed in SQL as follows:

```
SELECT buckets, MIN(Key1) AS start, MAX(Key1) AS end
FROM
(
    SELECT NTILE(7) OVER(ORDER BY Key1 ASC) AS buckets, Key1
    FROM Table1
)
GROUP BY buckets ORDER BY buckets.
```

The *NTILE* in the above SQL statement is one of the analytic functions in the Oracle Database. This function divides an ordered data set into the given number of buckets and assigns the bucket number.

The fourth step is that leaf number and section number are linked to the data ranges of partitions. As shown in Figure 4.4, all combinations of the data ranges are generated by the direct product. In SQL, a cross join operation can be used. Then, a leaf number is assigned to each combination in order, and a last section number, which is the number of key attributes plus one, is also assigned to all combinations. Next, data sets belonging to other section numbers are generated as shown in Figure 4.5. The data set created just before is copied to another set, and then the section number in the new set is decremented by one. After that, only the data range of the decremented section number in the new set is changed to no restriction, namely, the third key attribute (Store) in this example. The creation process of other section numbers, namely, copy of data set created just before, change of section number, and change of data range property, is iterated until the section number reaches one.

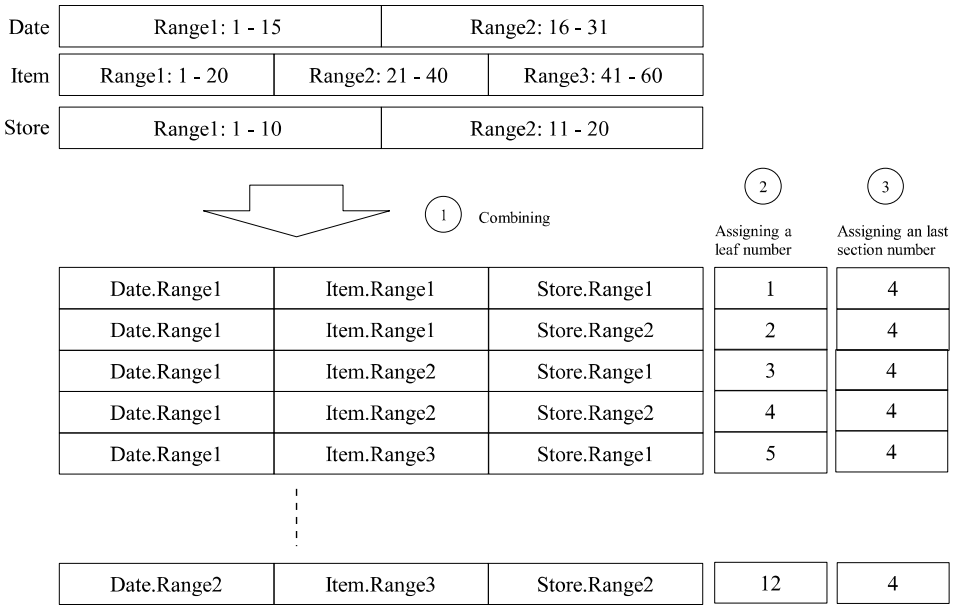


Figure 4.4 Generation process of the last section

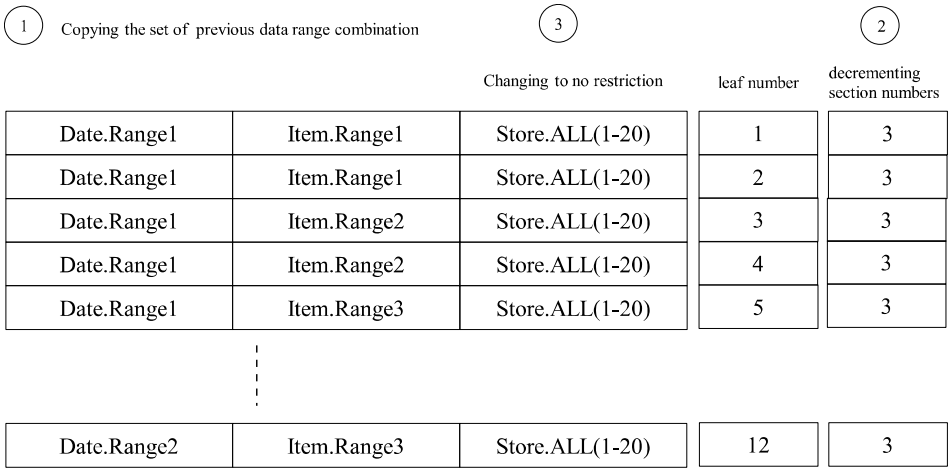


Figure 4.5 Generation process of other sections

As the final step, all generated records with leaf numbers, section numbers, and data range properties are inserted into the index table by using bulk processing.

4.3.3 Cluster Table Construction

In this stage, records with leaf and section numbers are generated from the original table and are stored in the cluster table which is allocated in a partitioned space. In this software, list partitioning is adopted for the cluster table in order to achieve efficient disk I/O of query processing as mentioned in Section 4.3.1. The primary aim of using partitioning is to use an operation called partition pruning, which enables to avoid unnecessary disk access (Fiorillo, 2012). For example, consider a SALES table split in a one-year unit. If a query is for the total amount of last year's sales, only the partition storing last year's sales is used for the computation, not the full scans. This partition pruning is a very useful method to improve the speed of query processing. Furthermore, another point in the partitioning is to specify the initial storage size using the original table size. This can contribute to allocating sequential spaces when the physical data file has sufficient spaces. As a consequence, efficient sequential disk access can be achieved.

After defining the cluster table on the database, actual data records are input. To create the cluster table, the following steps are repeated until the last record is processed. At the first step, a record is selected from the original table, and a random number is generated between 1 and $(i + 1)$ for the section number denoted by y, where i denotes the number of key attributes. The second step is that a leaf number set is made by scanning the index table. The condition to scan the index table is records possessing the same section number y and containing the key attribute values of the record selected from the original table at the first step. After that, one leaf number is randomly picked up from the leaf number set and then is assigned. As the final step, the selected record with leaf number and section number are inserted into the cluster table.

However, such a one-record processing is not efficient in practice; therefore, bulk processing is used for practical use at this data input stage. Also, since the cluster table size is very large, only bulk processing is not sufficient for faster data processing. The Oracle Database has some redo log files, which contain the information to re-execute the changes to the database (Kyte & Kuhn, 2014b). These files play an important role in the Oracle Database. When records are inserted, deleted or updated, the Oracle Database writes the information to the redo log files for performance improvement instead of updating the actual data files directly. However, this architecture could be one of the bottlenecks in such a large volume of transactions. The architecture is not necessary in this cluster table construction because if the

transactions fail, it is only necessary to retry them from the beginning. To turn off redo log generation, the Oracle Database provides an option - a no logging mode (Kyte & Kuhn, 2014a). Consequently, the software which was developed in this research sets the no logging mode to the cluster table.

4.4 Use of Multidimensional Cluster Sampling View

4.4.1 Approximate Query Processing

Detailed approximate query processing in SQL using the Multidimensional Cluster Sampling View is described. One of the differences between the Multidimensional Cluster Sampling View and the k-MDI is being able to execute approximate query processing in SQL. The approximate query processing in the Multidimensional Cluster Sampling View is based on the following seven steps:

1. All pairs of leaf and section numbers which have the data range property that includes query conditions are selected from the index table. All sections can be used for the approximation of mean values, whereas only section one can be used for the approximation of total values or the number of records.
2. The pairs are sorted in descending order on the section number to obtain a sample that is more relevant to the query conditions.
3. The necessary number of pairs for a sampling rate is selected from the higher rank of the sorted pairs. The necessary number of pairs can be calculated in advance because each section is approximately equivalent in size.
4. The result set of the previous step and the cluster table are joined on leaf number and section number to draw a sample.
5. The joined records are filtered by the query conditions.
6. The based values for the approximation are computed from the filtered records by the COUNT and SUM functions.
7. The approximate answer is estimated using the computed values as shown in Section 3.3.3.

4.4.2 Actual SQL

The actual SQL using the Multidimensional Cluster Sampling View is explained. The developed utility software provides easier query executions as illustrated in Figure 4.6. Users input only parameters for queries, and the SQL statement is automatically generated by the utility as demonstrated in the following example:

```
WITH
SelectLeaf AS
(
  SELECT ROW_NUMBER() OVER (ORDER BY DBMS_RANDOM.RANDOM)
         kl_leafno,           -- leaf number
         kl_secno,            -- section number
         kl_elements_cnt      -- the number of records
  FROM kmdi_leaf_repository   -- index table
  WHERE (kl_tab_number = 1)   -- unique table number
  AND   (kl_range_s01 <= 2451500 AND kl_range_e01 >= 2451000)
  AND   (kl_range_s02 <=    8000 AND kl_range_e02 >=    5000)
  AND   (kl_range_s03 <=       4 AND kl_range_e03 >=       2)
  ORDER BY kl_secno DESC
),
GetSampleLeaf AS
(
  SELECT kl_leafno, kl_secno, kl_elements_cnt FROM SelectLeaf
  WHERE ROWNUM <= 10    -- necessary section counts
),
GetSamplesCount AS
(
  SELECT SUM(kl_elements_cnt) AS SampleCount
  FROM GetSampleLeaf
),
GetSamples AS
(
  SELECT COUNT(*)              AS QueryCount,
         SUM(k.CS_SALES_PRICE) AS QuerySum1
  FROM GetSampleLeaf g, CATALOG_SALES_CLUSTER k -- cluster table
  WHERE (g.kl_leafno = k.kmdi_leaf)
  AND   (g.kl_secno  = k.kmdi_section)
  AND   (k.CS_SOLD_DATE_SK BETWEEN 2451000 AND 2451500)
  AND   (k.CS_BILL_ADDR_SK BETWEEN    5000 AND    8000)
```

```
  AND   (k.CS_ITEM_SK      BETWEEN       2 AND       4)
)
SELECT c.SampleCount               AS "SampleSize",
       c.SampleCount*100/143997065 AS "SampleRate(%)",
       s.QuerySum1/s.QueryCount    AS "AVG(CS_SALES_PRICE)"
FROM GetSamplesCount c, GetSamples s;
```

The query processing in SQL is mainly divided into two phases. The example SQL statement includes several temporary result sets, known as a common table expression (CTE). The first phase that identifies sections for samples can be found between *SelectLeaf* and *GetSampleLeaf* CTE in the SQL statement. The second phase that processes actual data can be seen after the first phase.

In the *SelectLeaf* CTE, leaf numbers, section numbers and the numbers of records in a section are selected from the index table, named *kmdi_leaf_repository* in this example, by the condition given by a user. The *kl_tab_number* shows the unique table number in the Multidimensional Cluster Sample View since all cluster tables share the index table. The conditions which start from *kl_range_** indicate data ranges on each level. Although this example does not show a condition to the section number due to the approximate query processing for the mean value, the condition for sections is also one of the conditions ($kl\_secno = 1$) in this CTE when the query is for total value or the total number of records. After the extraction from the index table, the records are randomly sorted by the *ROW_NUMBER* and *DBMS_RANDOM.RANDOM* function, and then the records are again sorted in descending order on section numbers. As a result, section numbers are in descending order while leaf numbers are in random order in each section number unit. This randomness could provide different samples every time while maintaining high relevancy.

In the next CTE, *GetSampleLeaf*, necessary sections are selected from a higher rank. In this example, the total number of Cluster Samples is at 960, and the sampling rate is at 1%. Therefore, the top 10 clusters, approximately 1% of the whole data, are selected from *SelectLeaf* by using the *ROWNUM* pseudo column.

In the *GetSamplesCount* CTE, the total number of records in the sampled clusters is computed. However, if users do not wish to obtain the actual sampling size, it is not necessary to execute this CTE.

The *GetSamples* CTE is a SQL statement to access actual data. By joining the result set of *GetSampleLeaf* and the cluster table, named *CATALOG_SALES_CLUSTER* in this example, on leaf numbers and section numbers, and by filtering actual data with the given query condition, records are selected and computed with the *COUNT* and *SUM* function from the cluster table.

Finally, the approximate answer is estimated using the method described in Section 3.3.3 and returned to the user.

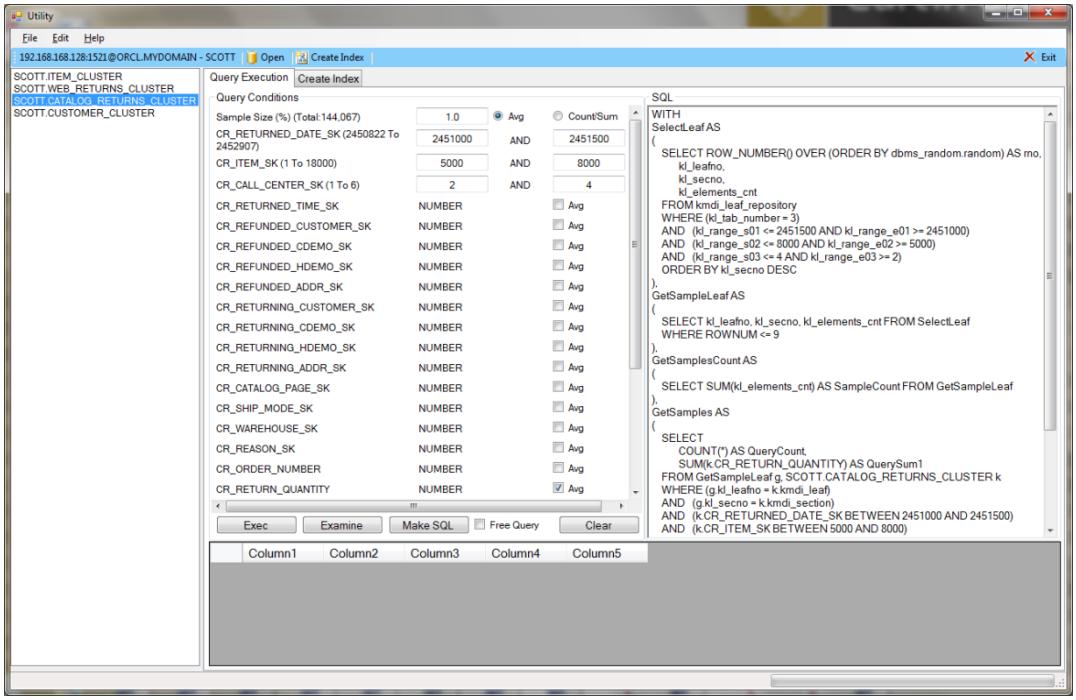


Figure 4.6 Screen image of utility for query execution using the Multidimensional Cluster Sampling View

4.5 Evaluation

4.5.1 Overview

To evaluate the accuracy of the approximation using the Multidimensional Cluster Sampling View and the response time, experiments on large datasets using utility software developed

in this research were conducted. The datasets are generated at approximately 100GB by *dsdgen* which is one of the software packages of TPC-DS (TPC, 2014). This software package can generate data using a hybrid approach of both real-world and synthetic data, comprising many tables and columns (Nambiar & Poess, 2006; Poess et al., 2007). All experiments were performed in CentOS 6.5 x64 with the Oracle Database 11g Release 2 on VMware Player 6.1 having 4 Gigabytes of RAM, four 3.4 GHz clock processor and 1 Terabytes hard disk drive.

In this evaluation, the index table of the Multidimensional Cluster Sampling View has four dimensions, and the view is composed of 192 leaves and 5 sections, namely, 960 Cluster Samples. The split points identifying the partitions on each attribute level are shown in Table 4.1.

Table 4.1 Partition information of the Multidimensional Cluster Sampling View with 4 dimensions in Section 0

	The number of partitions	Minimum value	1st split point	2nd split point	3rd split point	Maximum value
CS_SOLD_DATE_SK	4	2,450,815	2,451,346	2,451,809	2,452,236	2,452,654
CS_BILL_ADDR_SK	4	1	251,145	502,298	753,769	1,000,000
CS_ITEM_SK	4	1	51,010	102,001	153,004	204,000
CS_PROMO_SK	3	1	335	670	N/A	1000

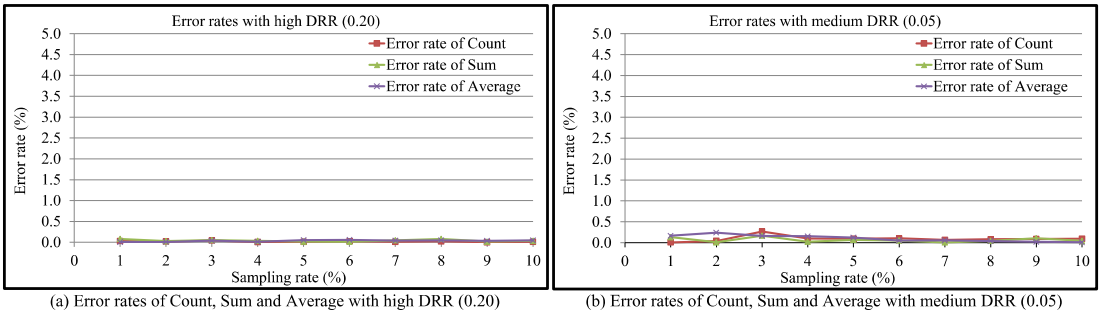
4.5.2 Accuracy Evaluation

For the first evaluation, the accuracy of approximate query processing using the Multidimensional Cluster Sampling View with four dimensions and approximately 144 million rows was examined. This experiment was performed with various database relevancy ratios (DRR) and various sampling rates (1% - 10%). The DRR is composed of high value (0.20), medium value (0.05), low value (0.01) and very low value (0.001). The query conditions of each DRR are determined by queries with absolute answers in advance. The average values of results in each sampling rate are adopted to compare with exact answers. The absolute answers are provided by the following query of the form:

```
SELECT COUNT(*), SUM(CS_SALES_PRICE), AVG(CS_SALES_PRICE)
```

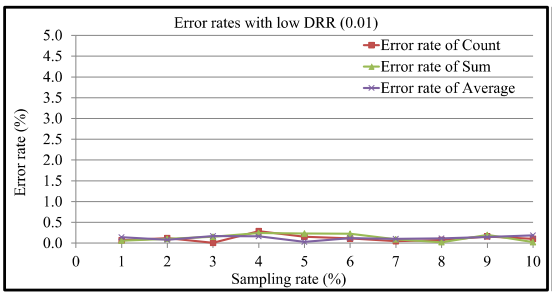
```
FROM   CATALOG_SALES
WHERE (CS_SOLD_DATE_SK BETWEEN s1 AND e1)
AND   (CS_BILL_ADDR_SK BETWEEN s2 AND e2)
AND   (CS_ITEM_SK      BETWEEN s3 AND e3)
AND   (CS_PROMO_SK     BETWEEN s4 AND e4).
```

Figure 4.7, which is composed of four line graphs, depicts the error rates between absolute answers and approximate answers using the Multidimensional Cluster Sampling View with various DRR. It shows the capability of the utility to go beyond three dimensions.



(a) Error rates of Count, Sum and Average with high DRR (0.20)

(b) Error rates of Count, Sum and Average with medium DRR (0.05)



(c) Error rates of Count, Sum and Average with low DRR (0.01)

(d) Error rates of Count, Sum and Average with very low DRR (0.001)

Figure 4.7 Error rates between absolute answers and approximate answers using the Multidimensional Cluster Sampling View with various DRR among three aggregate functions on 4 dimensions from a sampling rate of 1% to 10%

Figure 4.7a illustrates error rates on high DRR (0.20) with the maximum error rate at 0.08% for the SUM function with 1% or 8% samples. From this result, approximation with high DRR (0.20) using the Multidimensional Cluster Sampling View on four dimensions is extremely exact.

Figure 4.7b and Figure 4.7c illustrate error rates on medium DRR (0.05) and low DRR (0.01) with the highest error rates at 0.27% of the COUNT function with 3% samples and at 0.29%

of the COUNT function with 4% samples respectively. The accuracy of approximation using the Multidimensional Cluster Sampling View is still very high.

Figure 4.7d illustrates error rates on very low DRR (0.001) with the error rate of the SUM function with 1% samples being the highest, at 1.39% error. This result also has sufficient level of accuracy for approximate query processing. In addition, error rates of the AVERAGE function with all sampling rates stays under 1%.

In summary, all results show that error rates are below 1.5%. Especially among the aggregations, approximate answers for mean values sustain error rates below 1%. This means that approximate query processing using the Multidimensional Cluster Sampling View with four dimensions can provide high-precision answers even if queries are very selective, such as for 0.1% of the whole dataset.

4.5.3 Performance Evaluation

For the next evaluation, the query response time of approximate query processing using the Multidimensional Cluster Sampling View was measured. First of all, the query performance using the view was compared with full scans on a table with four dimensions and approximately 144 million rows. In order to eliminate the effect of the buffer cache of the operating system and the Oracle Database, this experiment was conducted with configuration changes. The configuration for the Oracle Database can be made possible by changing the parameter of *FILESYSTEMIO_OPTIONS* to *'SETALL'* and then by restarting the database. With this configuration, the writer processes in the Oracle Database always has access to hard disks directly.

As shown in Figure 4.8, it is obvious that the query response time of the Multidimensional Cluster Sampling View is faster than the full scans. The response time using the Multidimensional Cluster Sampling View with 1% samples is at 2.24 seconds, whereas the response time using the full scans is at 220.64 seconds, which is 98.5 times compared with the Multidimensional Cluster Sampling View. Thus, the response ability of the Multidimensional Cluster Sampling View demonstrates improvement between 10 and 99 times compared with the full scans, almost in proportion to the sampling rates. This can save

significant time for users. Users could execute various queries to analyse the massive amount of data from diversified standpoints.

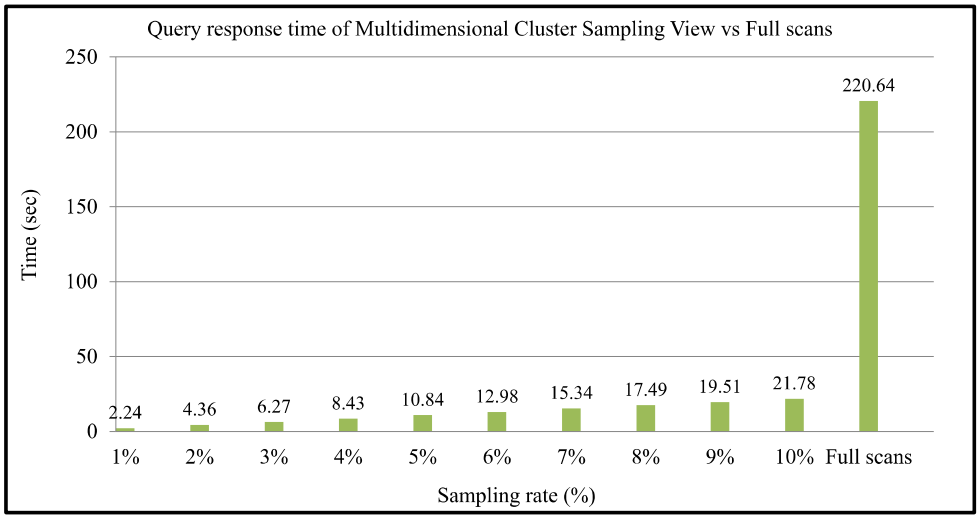


Figure 4.8 Query response time of the Multidimensional Cluster Sampling View and full scans

Next, the query performance using the Multidimensional Cluster Sampling View was compared with a columnar database management system called InfiniDB (version 4.6) on a large table with a variety of dimensions and approximately 288 million rows. The columnar databases, which is based on a column-oriented database management system proposed by Stonebraker et al. (2005), are increasingly used as an infrastructure to analyse huge volumes of data. To remove the effect of the buffer cache, a function to flush out the buffer cache, *calflushcache()*, is always performed before executing queries on the InfiniDB. Since the previous experiment shows that query execution time using the Multidimensional Cluster Sampling View is directly proportional to the sampling rate, the sampling rate using the view is fixed at 1% in this experiment. The response time of the Multidimensional Cluster Sampling View with other sampling rates could be easily projected based on the above observation.

Figure 4.9 shows the query response time of the Multidimensional Cluster Sampling View and the InfiniDB. The response time using the view is approximately 4 seconds regardless of the number of dimensions. In contrast, the response time using the InfiniDB is directly proportional to the number of dimensions because columnar databases access key and measurable values only used in the query. Even the fastest time on one dimension using the

InfiniDB is five times longer than the time using the Multidimensional Cluster Sampling View, and the difference reaches up to 30 times in this experiment. It is clear that query performance using the Multidimensional Cluster Sampling View is faster, whereas the InfiniDB can provide absolute answers.

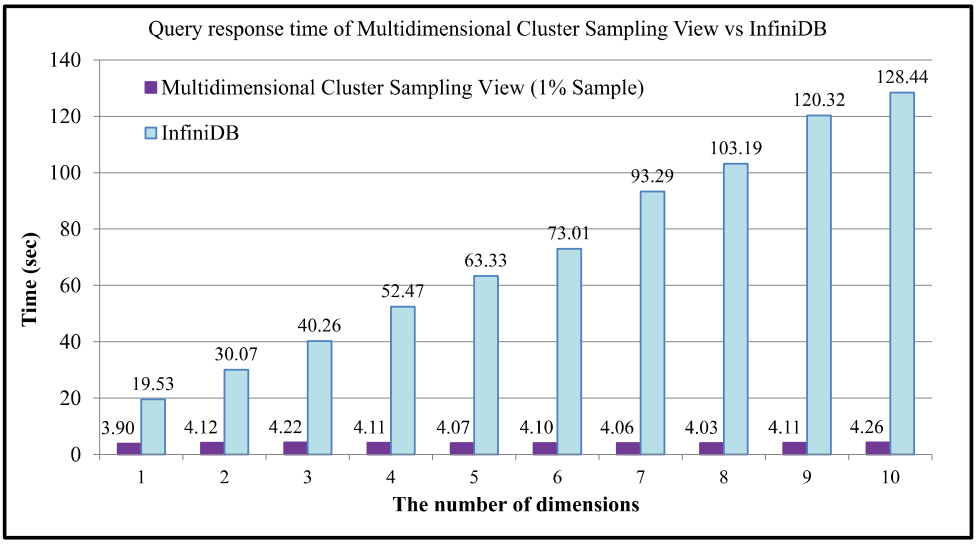


Figure 4.9 Query response time of the Multidimensional Cluster Sampling View and InfiniDB

At first, in order to achieve efficient data access, the clustered index, storing the data which has same key values in the same data blocks (Fiorillo, 2012), had been tried. However, it did not work effectively contrary to the expectations since the clustered size is approximately 32KB maximum in the Oracle Database. This meant that many records could not be stored in one cluster. Moreover, clusters with the same key values were not stored in segments sequentially even if records are reorganised in order with key values. As a result, a considerable amount of time was required for random access to clustered units. Although Oracle partitioning is not a fundamental solution to this problem, sequential initial space allocation by the partitioning could improve the efficiency of data access.

4.6 Chapter Summary

In this chapter, behaviour of the practical utility software that was actually developed was introduced. The software is able to build the Multidimensional Cluster Sample View in common database tables and to execute approximate query processing using the view in SQL. The view can be simply implemented in two tables. In addition, it was found that the Multidimensional Cluster Sampling View has a high-level of accuracy for approximate query processing on four-dimensional attributes and also a faster response time than full scans or a columnar database.

CHAPTER 5

EFFECTS OF DIMENSIONALITY

5.1 Introduction

With an ever-increasing accumulation of enterprise data, traditional query processing, which provides precise answers, is time consuming when used in decision support systems with large quantities of data (Babcock et al., 2003; Chakrabarti et al., 2001; S. Chaudhuri et al., 2007; Garofalakis & Gibbon, 2001; Liu, 2009; Ruoming et al., 2006; Wang et al., 2013). However, it is unnecessary for decision makers to obtain absolute answers; approximate answers which are quite close to the exact answers can be acceptable in the decision-making process (Babcock et al., 2003; Chakrabarti et al., 2001; Rudra et al., 2012). Approximate query processing based on random sampling is a very efficient method for dealing with extensive volumes of data (Joshi & Jermaine, 2008a, 2008b; Li et al., 2008; Olken & Rotem, 1986, 1995; Rudra et al., 2012; Ruoming et al., 2006; Webb & Wang, 2014). The sampling-based approximate query processing can achieve fast data processing in large databases due to only requiring access to a small amount of data. However, a small sample in highly selective queries or skewed data distribution is likely to miss most, if not all, of the relevant data because the small sample may not be fully represented throughout the entire dataset (S. Chaudhuri et al., 2007; Joshi & Jermaine, 2008b; Li et al., 2008; Rudra et al., 2012). A large sample can attain a higher level of confidence; however, its computation is much more time-consuming. As a result, the large sample negates the benefits of sampling.

The Multidimensional Cluster Sample View, proposed in Chapter 3, allows approximate query processing to be performed effectively and efficiently. In Chapter 4, the effectiveness of query processing using the view was empirically demonstrated on a large dataset with four dimensions. Also, the speed of query processing using the view was compared with full scans and a columnar database. However, multidimensional models in data warehouses are generally composed of approximately 8 to 15 tables (Kimball et al., 2011). This means that approximate query processing used in decision support systems must work properly over

ten-dimensional key attributes for practical use. Therefore, the Multidimensional Cluster Sampling View is needed to perform further evaluations in terms of dimensionality including high-dimensional data. To evaluate the Multidimensional Cluster Sampling View with multi-dimensional data, there are three points which must be examined, namely accuracy of approximation, query performance and construction performance. However, the second point, the query performance of approximate query processing with various dimensions, has been already evaluated with a columnar database called InfiniDB in Chapter 4. It has been demonstrated that the number of dimensions does not affect the query performance in the view. Hence, the rest of the points are critical to demonstrate the practicality of the Multidimensional Cluster Sampling View in data warehouses.

In this chapter, the effects of dimensionality on accuracy in the Multidimensional Cluster Sampling View are examined and evaluated. Furthermore, an analysis of the effects of dimensionality on the construction performance of the Multidimensional Cluster Sampling View is also carried out.

The rest of this chapter is organised as follows. Section 5.2 examines and evaluates the effects of the number of dimensions on accuracy in the Multidimensional Cluster Sampling View. In Section 5.3, the impact of the number of dimensions on the construction time of this view is examined and evaluated. Finally, this chapter is summarised in Section 5.4.

5.2 Dimensionality Evaluation for Accuracy

5.2.1 Evaluation Approach

The objective of this section is to evaluate whether the number of dimensions affects the level of accuracy of approximation provided by the Multidimensional Cluster Sampling View. In terms of the effects of dimensionality on accuracy, some experiments were conducted in the Multidimensional Cluster Sampling View with 6, 10 and 13 dimensional key attributes. Detailed information regarding an original table of this view will be presented in Section 5.2.2. To evaluate the level of accuracy of this view, four DRR values were used: medium DRR (0.05), low DRR (0.01), very low DRR (0.001) and extremely low DRR

(0.0001). The approximation was also compared with the absolute answer using three aggregate functions (AVG, COUNT and SUM), with the sampling rate ranging from 1% to 10%. The queries were repeated ten times on each sampling rate, and the mean value of the results was adopted as the approximate answer. Finally, the effect on accuracy of the number of dimensions is discussed in terms of error rates, which are calculated by the difference between the absolute and approximate answers expressed as a percentage of the absolute answers.

5.2.2 Data Sets

The dataset is generated by *dsdgen*, the software package of TPC-DS (TPC, 2014). The Multidimensional Cluster Sampling View was built from a fact table using the utility that was developed. The table named *WEB_SALES* is composed of approximately 72 million records. Table 5.1 depicts column definitions of the *WEB_SALES* table and the target key attributes used in the experiments. There are 18 foreign keys in the table, and some of them are used as dimensions in each experiment. In addition to this, a measure attribute named *WS_WXT_SALES_PRICE*, which ranges 0 to 29,810, was employed in the experiments.

Table 5.1 Column definitions of WEB_SALES table and target key attributes used in experiments for effects of dimensionality on accuracy

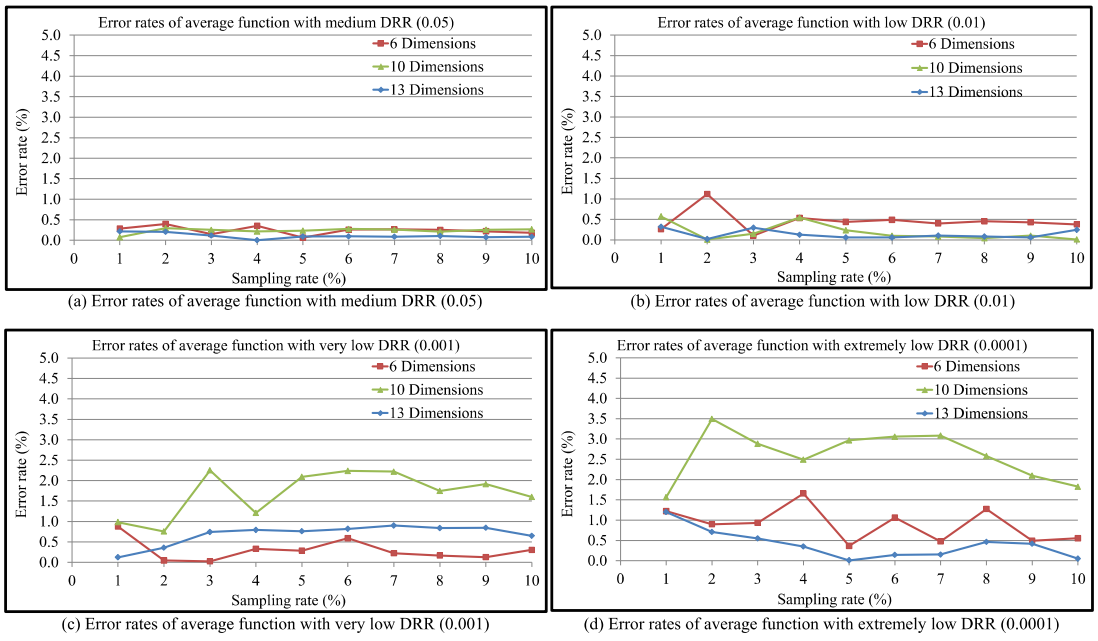
Column	Foreign Key	Dimensional Table	6 Dimensions	10 Dimensions	13 Dimensions
ws_sold_date_sk	d_date_sk	date_dim	✓	✓	✓
ws_sold_time_sk	t_time_sk	time_dim	✓	✓	✓
ws_ship_date_sk	d_date_sk	date_dim		✓	✓
ws_item_sk	i_item_sk, wr_item_sk	item, web_returns	✓	✓	✓
ws_bill_customer_sk	c_customer_sk	customer	✓	✓	✓
ws_bill_cdemo_sk	cd_demo_sk	customer_demographics		✓	✓
ws_bill_hdemo_sk	hd_demo_sk	household_demographics		✓	✓
ws_bill_addr_sk	ca_address_sk	customer_address		✓	✓
ws_ship_customer_sk	c_customer_sk	customer			✓
ws_ship_cdemo_sk	cd_demo_sk	customer_demographics			✓
ws_ship_hdemo_sk	hd_demo_sk	household_demographics			✓
ws_ship_addr_sk	ca_address_sk	customer_address			✓
ws_web_page_sk	wp_web_page_sk	web_page	✓	✓	✓
ws_web_site_sk	web_site_sk	web_site	✓	✓	
ws_ship_mode_sk	sm_ship_mode_sk	ship_mode			
ws_warehouse_sk	w_warehouse_sk	warehouse			
ws_promo_sk	p_promo_sk	promotion			
ws_order_number	wr_order_number	web_returns			
ws_quantity					
ws_wholesale_cost					
ws_list_price					
ws_sales_price					
ws_ext_discount_amt					
ws_ext_sales_price					
ws_ext_wholesale_cost					
ws_ext_list_price					
ws_ext_tax					
ws_coupon_amt					
ws_ext_ship_cost					
ws_net_paid					
ws_net_paid_inc_tax					
ws_net_paid_inc_ship					
ws_net_paid_inc_ship_tax					
ws_net_profit					

5.2.3 Experimental Results and Analysis

In conducting the analysis, the experiments were divided into three sets in terms of different aggregate functions: AVERAGE, COUNT and SUM function.

For the first experiment, an examination was conducted regarding the effects of dimensionality on the accuracy of the AVERAGE function in the Multidimensional Cluster Sampling View. Figure 5.1 includes four line graphs that demonstrate the error rates of AVERAGE function between the absolute and approximate answers obtained through the use of the Multidimensional Cluster Sampling View with various DRR values on three types of dimensions which ranged from a sampling rate of 1% to 10%. As shown in Figure 5.1a, all results indicate very low error rates, all below 0.5% error. In this chart, no significant differences in accuracy are seen due to the effects of dimensionality. In Figure 5.1b, all

results except for the sampling rate of two percent on six dimensions show error rates below 0.5%. In Figure 5.1c, whilst the error rates always remain below 2.5%, it is clearly evident that error rates vary for different numbers of dimensions. The results seen when there were 10 dimensions show the largest error rate of all, and then 13 and 6 dimensions follow suit respectively. Therefore, the dimensionality of queries using the Multidimensional Cluster Sampling View does not correlate with accuracy in this chart. This is due to there being no proportional relationship between the number of dimensions and error rates. Figure 5.1d shows error rates below 3.5% in all results. Similarly, for very low DRR (0.001), it can be clearly seen that there are differences among the three sets of dimensions. The largest error rate among them is seen when 10 dimensions are used. 6 dimensions and 13 dimensions follow respectively. Consequently, the effects of dimensionality on the accuracy of AVERAGE function using the Multidimensional Cluster Sampling View are not evident in the results of this experiment.



(a) Error rates of average function with medium DRR (0.05)

(b) Error rates of average function with low DRR (0.01)

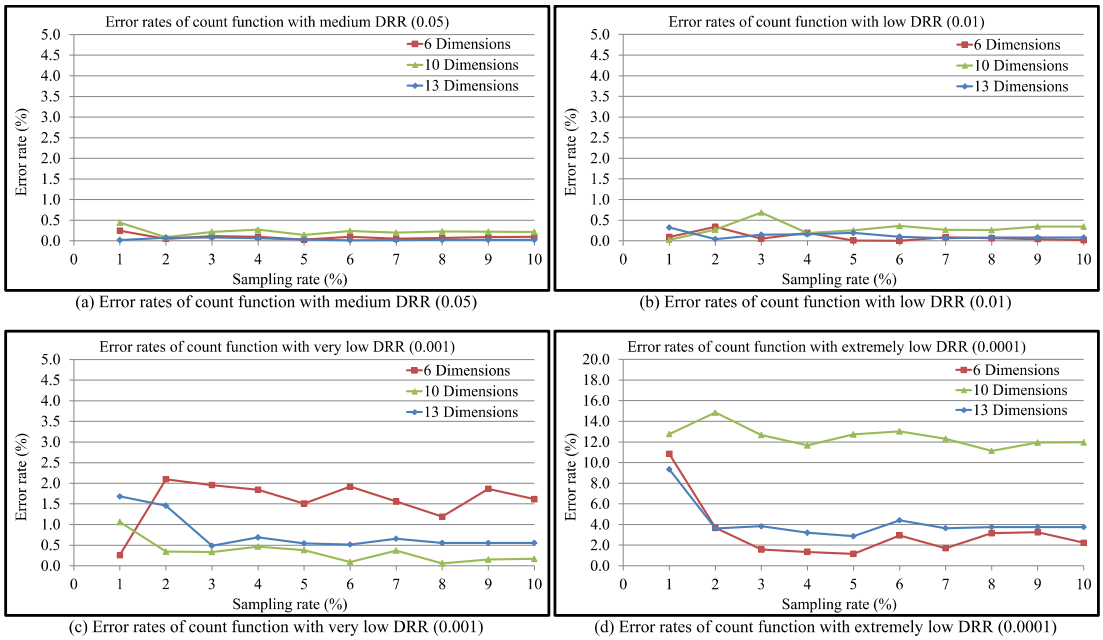
(c) Error rates of average function with very low DRR (0.001)

(d) Error rates of average function with extremely low DRR (0.0001)

Figure 5.1 Error rates of AVERAGE function between absolute answers and approximate answers using the Multidimensional Cluster Sampling View with various DRR values on three types of dimensions from a sampling rate of 1% to 10%

For the next experiment, the effects of dimensionality on the accuracy of COUNT function using the Multidimensional Cluster Sampling View were examined. Figure 5.2 includes four line graphs which show the error rates of COUNT function between absolute and

approximate answers obtained through the use of the Multidimensional Cluster Sampling View, with various DRR values on three types of dimensions ranging from a sampling rate between 1% and 10%. As shown in Figure 5.2a, all results indicate an error rate under 0.5%. No significant differences in accuracy are seen in this graph that could be attributed to the effects of dimensionality. Next, in Figure 5.2b, all error rates except for the 3% sampling show below 0.5%. In Figure 5.2c, all results remain low error rates, below 2.5%. It is clearly seen that error rates vary for different numbers of dimensions. The largest error rate is seen on 6 dimensions, followed in order by 13 dimensions and 10 dimensions. Therefore, the dimensionality of queries using the Multidimensional Cluster Sampling View does not correlate with accuracy in this graph. In Figure 5.2d, error rates on ten dimensions show an approximate percentage of between 11 % and 15%. These error rates are clearly unacceptable for use in approximation. The approximation with 1% samples on 6 and 13 dimensions also show high error rates, approximately 11% and 9% respectively. In contrast, the error rates between 2% and 10% samples on 6 and 13 dimensions are acceptable, under 5%. Hence, the effects of dimensionality on the accuracy of the COUNT function using the Multidimensional Cluster Sampling View are not obvious in terms of this experiment.



(a) Error rates of count function with medium DRR (0.05)

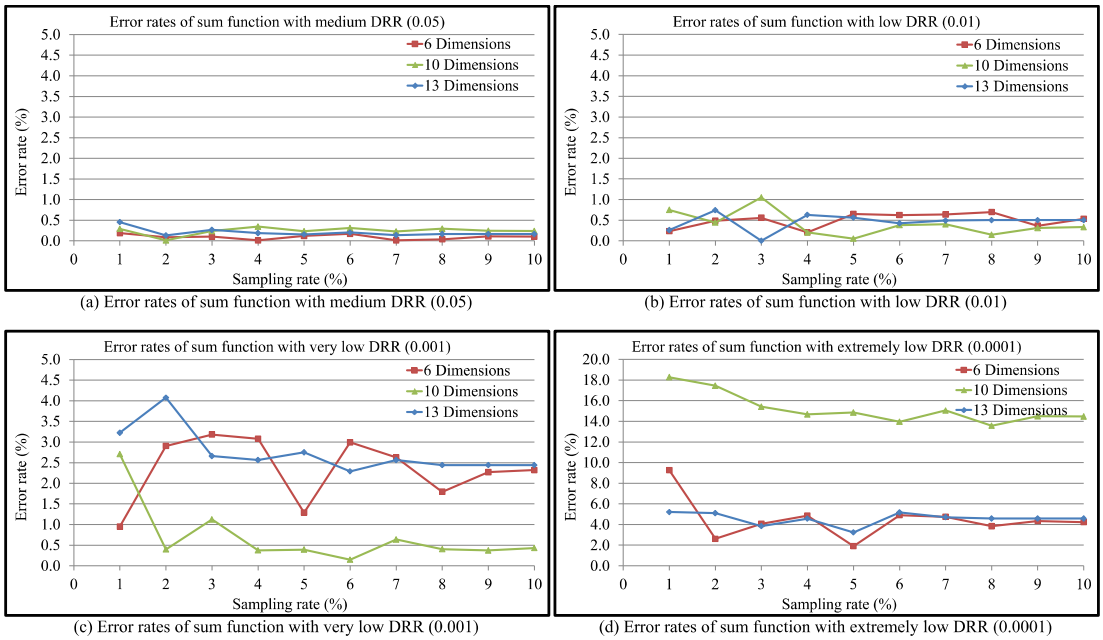
(b) Error rates of count function with low DRR (0.01)

(c) Error rates of count function with very low DRR (0.001)

(d) Error rates of count function with extremely low DRR (0.0001)

Figure 5.2 Error rates of COUNT function between absolute answers and approximate answers using the Multidimensional Cluster Sampling View with various DRR values on three types of dimensions from a sampling rate of 1% to 10%

For the final experiment, the effects of dimensionality on the accuracy of the SUM function using the Multidimensional Cluster Sampling View were examined. Figure 5.3 includes four line graphs that demonstrate the error rates of SUM function between absolute and approximate answers obtained through the use of the Multidimensional Cluster Sampling View, with various DRR values on three types of dimensions ranging from sampling rates between 1% and 10%. As shown in Figure 5.3a and Figure 5.3b, all results maintain low error rates, under 1%. The significant effects of dimensionality on accuracy are not visible in these charts. In Figure 5.3c, error rates on 6 dimensions and 13 dimensions are relatively large on the whole, but below 4%. The error rates on 10 dimensions show the lowest values except for the sampling rate of 1%. In this graph, the effects of dimensionality on accuracy are also unseen. In Figure 5.3d, the results on 10 dimensions show the largest error, between 13% and 18%. The error rates on 6 and 13 dimensions remain almost similar on the whole, around 5%. No significant effects of dimensionality on the accuracy are seen in this graph. Therefore, the number of dimensions does not correlate with the accuracy of SUM function using the Multidimensional Cluster Sampling View.



(a) Error rates of sum function with medium DRR (0.05)

(b) Error rates of sum function with low DRR (0.01)

(c) Error rates of sum function with very low DRR (0.001)

(d) Error rates of sum function with extremely low DRR (0.0001)

Figure 5.3 Error rates of SUM function between absolute answers and approximate answers using the Multidimensional Cluster Sampling View with various DRR values on three types of dimensions from a sampling rate of 1% to 10%

Thus, there are three findings that can be reached from these experiments. Firstly, the number of dimensions in the Multidimensional Cluster Sampling View does not affect the accuracy of the approximation. Secondly, the approximation of the AVERAGE function using the Multidimensional Cluster Sampling View could provide a reliable level of accuracy even in extremely selective queries (0.0001 DRR). Finally, the approximation of the AVERAGE function using the Multidimensional Cluster Sampling View could provide a more accurate result than that provided for the COUNT or SUM functions. In the experiments conducted, the approximation of the COUNT or SUM functions using this view could provide an acceptable level of accuracy within a very low DRR (0.001 DRR), whereas it could not provide an acceptable level of accuracy within an extremely low DRR (0.0001 DRR). This is because Cluster Samples with larger section numbers, which belong to specific data ranges, cannot be used as samples of the total number of records or the total size. Therefore, the approximation using this view is more reliable for mean values than for the number of records or total values.

5.3 Dimensionality Evaluation for Construction Performance

5.3.1 Evaluation Approach

The aim of this section is to evaluate whether dimensionality affects the construction performance of the Multidimensional Cluster Sampling View. In order to compare the effects of dimensionality on construction time, the performance between one and ten dimensions was measured using the utility developed. Detailed information regarding an original table of the view will be presented in Section 5.3.2. All experiments were carried out in CentOS 6.5 (64 bit) with the Oracle Database 11g Release 2 on VMware Player 6.1, having 4 Gigabytes of RAM, four 3.4 GHz clock processors and a 1 Terabyte disk.

5.3.2 Data Sets

The raw data is populated by the data generator of TPC-DS using a scale factor of 1GB (TPC, 2014). A table named *CATALOG_SALES* is used to build the Multidimensional Cluster Sampling View with a variety of dimensions. The table is composed of 34 columns with approximately 1.45 million records. Table 5.2 shows the target key attributes used in this experiment. For example, the top three attributes, *CS_SOLD_DATE_SK*, *CS_SOLD_TIME_SK* and *CS_SHIP_DATE_SK*, are used as key attributes to measure the construction time in regards to the three dimensions.

Table 5.2 Target key attributes used in experiments for effects of dimensionality on construction performance

No.	Column Name	Dimensions
1	CS_SOLD_DATE_SK	✓
2	CS_SOLD_TIME_SK	✓
3	CS_SHIP_DATE_SK	✓
4	CS_BILL_CUSTOMER_SK	✓
5	CS_BILL_CDEMO_SK	✓
6	CS_BILL_HDEMO_SK	✓
7	CS_BILL_ADDR_SK	✓
8	CS_SHIP_CUSTOMER_SK	✓
9	CS_SHIP_CDEMO_SK	✓
10	CS_SHIP_HDEMO_SK	✓

5.3.3 Experimental Results and Analysis

Figure 5.4 shows the construction time of the Multidimensional Cluster Sampling View between one and ten dimensions. Between one and seven dimensions, the performance shows around 2.5 minutes. The effects of dimensionality on the construction time cannot be clearly seen in regards to these dimensions. However, it shows 5 minutes on eight dimensions, approximately double when compared with seven dimensions. Furthermore, it shows 12.5 minutes (five times) and 18.8 minutes (seven and a half times) on nine and ten dimensions respectively. Therefore, it appears that there is another factor which could affect the construction time.

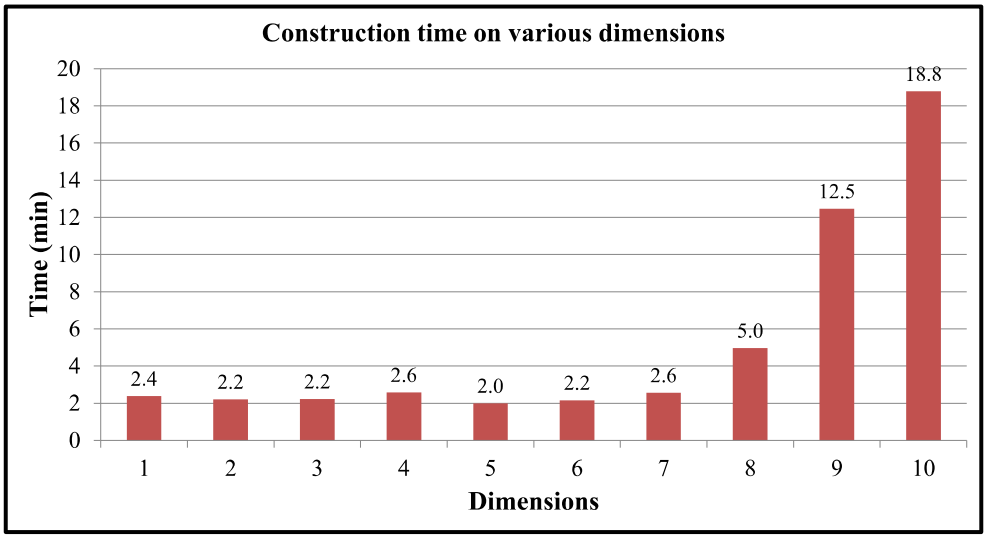


Figure 5.4 Construction time of the Multidimensional Cluster Sampling View between one and ten dimensions

Table 5.3 shows the number of Cluster Samples on each dimension in Figure 5.4. As shown in the table, the number of Cluster Samples between one and seven dimensions, with the exception of 6 dimensions, is around 1,000. Then, the number on eight dimensions and nine dimensions is double and five times respectively. This rate of increase is the same as that of construction time. Consequently, it seems that the number of Cluster Samples could affect the construction time.

Table 5.3 The number of Cluster Samples on each dimension in Figure 5.4

Dimensions	The number of Cluster Samples
1	1,000
2	972
3	864
4	960
5	972
6	672
7	1,024
8	2,304
9	5,120
10	11,264

Consequently, a further experiment was conducted to confirm whether the number of Cluster Samples has an impact on the construction time. The performance and Cluster Sample size was measured on four fixed dimensions. As shown in Figure 5.5, 2.1 minutes are required for the construction with 960 Cluster Samples, and it takes 3.4 minutes for the construction with 2,000 Cluster Samples. It takes 4.6 minutes for the construction with 3,125 Cluster Samples. As expected, the construction time increases almost in direct proportion to the number of Cluster Samples.

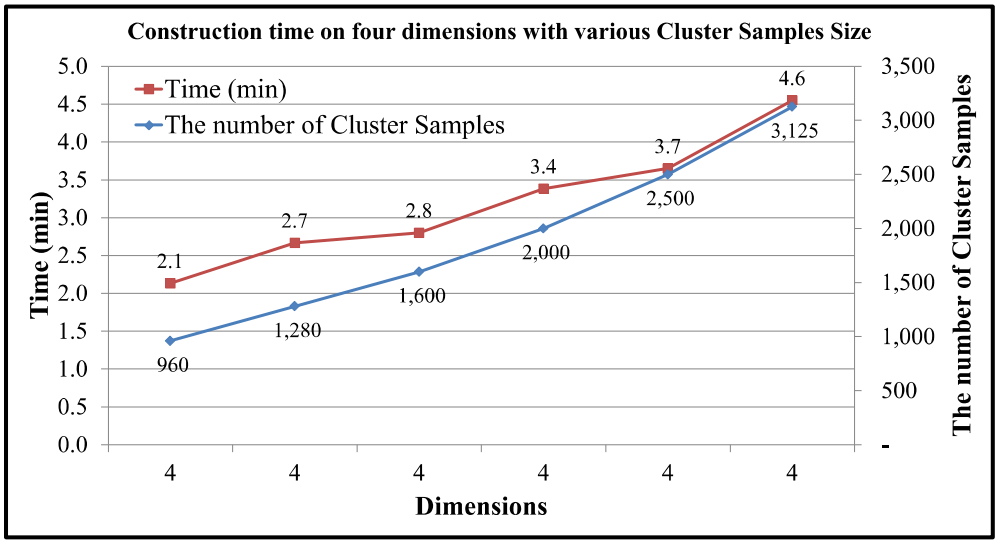


Figure 5.5 Construction time on four dimensions between 960 and 3,125 Cluster Samples

In summary, the factor which has an impact on the construction time of the Multidimensional Cluster Sampling View is not the dimensionality, but rather the Cluster Sample size. In a strict sense, although the dimensionality or other factors might affect the construction time, the effects could be minimal.

5.4 Chapter Summary

In this chapter, the effects of dimensionality on both the level of accuracy of approximation and construction time were examined using the Multidimensional Cluster Sampling View. It was discovered that the level of accuracy of approximation using this view was not affected by the number of dimensions. Also, it was determined that the number of dimensions did not affect the construction time of the view, but the Cluster Sample size affected the construction performance. Hence, the Multidimensional Cluster Sampling View is capable and practical even in high-dimensional data if the number of Cluster Samples is limited to a certain number.

CHAPTER 6

CONCLUSIONS

6.1 Summary

In this thesis, an effective and efficient method that enables both a fast query processing and approximation with acceptable accuracy simultaneously has been investigated and developed. Moreover, the capability and practicality of the method has been demonstrated by ten experiments, evaluations and the development of utility software.

In Chapter 3, a novel view called the Multidimensional Cluster Sampling View to support efficient and effective approximate query processing was developed. The approximate query processing utilises the benefits of random sampling at its maximum in databases. The simple implementation into database tables and manipulation by SQL statements are considered. Also, the view is designed to enable the drawing of random samples which are relevant to the query conditions even in highly selective queries. In addition, approximate query processing in the Multidimensional Cluster Sampling View allows the provision of sufficient random samples compared with the k-MDI.

In Chapter 4, practical software which was developed was presented. The software can build the Multidimensional Cluster Sample View in database tables with a few parameters and execute approximate query processing in SQL using the view. In the thesis, the software design, the implementation method and the execution method of approximate query processing are presented in detail. Also, it was empirically demonstrated that the Multidimensional Cluster Sampling View provides a high-level of accuracy of approximation on four-dimensional data. In addition, it was empirically demonstrated that the view provides a faster response time than full scans or a columnar database called InfiniDB.

In Chapter 5, the effects of dimensionality on the accuracy of approximation in the Multidimensional Cluster Sampling View and on construction time of the view were examined and evaluated. It was empirically shown that the number of dimensions does not affect the accuracy of approximation and construction time, whereas the Cluster Sample size affects the construction time.

Thus, an effective technique and practical utility for approximate query processing have been studied in this thesis. This makes valuable contributions to the field of approximate query processing.

6.2 Future Directions

In this thesis, the practical utility was developed by focusing on a particular database, namely the Oracle Database. The effectiveness and efficiency of the Multidimensional Cluster Sampling View has not yet been demonstrated in other databases. For future work, it could be useful to enhance the utility software which can operate in other databases.

In the evaluation regarding the effects of dimensionality, it was discovered that Cluster Sample size has an impact on construction time of the Multidimensional Cluster Sampling View. Therefore, it is proposed that the suitable size of Cluster Samples be investigated for the best performance. Also, the Multidimensional Cluster Sampling View uses a list partitioning technique provided by the Oracle Database. The effect on construction time by Cluster Sample size might come from the use of a partitioning technique. Therefore, the fundamental reason could be made clear by implementing the view in various databases using other functions of block-level access.

The data which were used for conducting all the experiments reported in this thesis are generated by a data generator supported by the Transaction Processing Performance Council. Although the generation method uses a hybrid approach of both real-world and synthetic data, the data are not real life. Therefore, it is also proposed to further evaluate the Multidimensional Cluster Sampling View for practical use, such as its use in various real life data sets with huge volumes.

There are some drawbacks in the Multidimensional Cluster Sampling View. One of the drawbacks is that it has to be rebuilt for any changes. Since it takes a long time to rebuild the view, it is not efficient. Therefore, it could be useful to investigate how to update the view efficiently without rebuilding.

The Multidimensional Cluster Sampling View assumes that it is built on two tables in a database. This might be a shortcoming for some applications or large scale systems because the view cannot scale out such as in distributed computing. Therefore, it could be useful to investigate how to implement the view on a distributed database system. However, actual data in the Multidimensional Cluster Sampling View are clustered. Therefore, it is possible to distribute the clustered sample data into some databases.

It seems that it is difficult to identify a sufficient sample size that ensures a high degree of accuracy in any skewed data distribution or any query. The suitable sample size depends on the query conditions and the data distribution. However, decision makers do not know the selectivity of the query and data distribution. Therefore, it could be useful to investigate how to identify a sufficient sample size without making a mistake in any data sets and queries. However, the ground of the Multidimensional Cluster Sampling View is simple random sampling without replacement. The method for estimating the errors with random samples has been well developed in a mathematical theory, and it can be applied for calculation of the sufficient sample size. Therefore, the lack of a method for identifying a sufficient sample size may not be a serious drawback.

Thus, there are some shortcomings in the Multidimensional Cluster Sampling View at this stage; however, they are not major drawbacks but rather growth potential.

REFERENCES

Agarwal, S., Milner, H., Kleiner, A., Talwalkar, A., Jordan, M., Madden, S., . . . Stoica, I. (2014). *Knowing when you're wrong: building fast and reliable approximate query processing systems*. Paper presented at the Proceedings of the 2014 ACM SIGMOD international conference on Management of data, Snowbird, Utah, USA.

Arnott, D., & Pervan, G. (2014). A critical analysis of decision support systems research revisited: the rise of design science. *J Inf technol, 29*(4), 269-293.

Babcock, B., Chaudhuri, S., & Das, G. (2003). *Dynamic sample selection for approximate query processing*. Paper presented at the Proceedings of the 2003 ACM SIGMOD international conference on Management of data, San Diego, California.

Bonini, C. P. (1963). *Simulation of information and decision systems in the firm*. Englewood Cliffs, N.J.: Prentice-Hall.

Chakrabarti, K., Garofalakis, M., Rastogi, R., & Shim, K. (2001). Approximate query processing using wavelets. *The VLDB Journal, 10*(2-3), 199-223. doi: 10.1007/s007780100049

Chaudhuri, A., & Mukeriee, B. (1985). DOMAIN ESTIMATION IN FINITE POPULATIONS. *Australian Journal of Statistics, 27*(2), 135-137. doi: 10.1111/j.1467-842X.1985.tb00555.x

Chaudhuri, S., Das, G., & Narasayya, V. (2007). Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst., 32*(2), 9. doi: 10.1145/1242524.1242526

Cormode, G., Garofalakis, M., Haas, P. J., & Jermaine, C. (2012). Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends databases, 4*(1–3), 1-294. doi: 10.1561/1900000004

Eom, S., & Kim, E. (2006). A survey of decision support system applications (1995–2001). *Journal of the Operational Research Society, 57*(11), 1264-1278.

Fiorillo, C. (2012). *Oracle Database 11gR2 Performance Tuning Cookbook* Retrieved from http://CURTIN.eblib.com.au/patron/FullRecord.aspx?p=943418

Gantz, J., & Reinsel, D. (2012). THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadow s, and Biggest Grow th in the Far East. Retrieved July 5, 2015, from http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf

Garofalakis, M. N., & Gibbon, P. B. (2001). *Approximate Query Processing: Taming the TeraBytes (tutorial)*. Paper presented at the Proceedings of the 27th International Conference on Very Large Data Bases.

Hellerstein, J. M., Haas, P. J., & Wang, H. J. (1997). *Online aggregation*. Paper presented at the Proceedings of the 1997 ACM SIGMOD international conference on Management of data, Tucson, Arizona, USA.

Hou, W.-C., Ozsoyoglu, G., & Taneja, B. K. (1988). *Statistical estimators for relational algebra expressions*. Paper presented at the Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, Austin, Texas, USA.

Inmon, W. H. (2002). *Building the Data Warehouse (Third Edition)*: John Wiley & Sons, Inc.

International Data Corporation. (2014). The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. Retrieved July 5, 2015, from http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

Ioannidis, Y. (2003). Approximations in Database Systems. In D. Calvanese, M. Lenzerini, & R. Motwani (Eds.), *Database Theory — ICDT 2003* (Vol. 2572, pp. 16-30): Springer Berlin Heidelberg.

Jermaine, C., Arumugam, S., Pol, A., & Dobra, A. (2008). Scalable approximate query processing with the DBO engine. *ACM Trans. Database Syst., 33*(4), 1-54. doi: 10.1145/1412331.1412335

Joshi, S., & Jermaine, C. (2008a). Materialized Sample Views for Database Approximation. *Knowledge and Data Engineering, IEEE Transactions on, 20*(3), 337-351. doi: 10.1109/tkde.2007.190664

Joshi, S., & Jermaine, C. (2008b). *Robust Stratified Sampling Plans for Low Selectivity Queries*. Paper presented at the Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on.

Kimball, R., Ross, M., & Thornthwaite, W. (2011). *Data Warehouse Lifecycle Toolkit (2nd Edition)*. Hoboken, NJ, USA: Wiley.

Kyte, T., & Kuhn, D. (2014a). Investigating Redo *Oracle Database Transactions and Locking Revealed* (pp. 127-146): Apress.

Kyte, T., & Kuhn, D. (2014b). Redo and Undo *Oracle Database Transactions and Locking Revealed* (pp. 109-126): Apress.

Li, X., Han, J., Yin, Z., Lee, J.-G., & Sun, Y. (2008). *Sampling cube: a framework for statistical olap over sampling data*. Paper presented at the Proceedings of the 2008

ACM SIGMOD international conference on Management of data, Vancouver, Canada.

Lightstone, S. S., Teorey, T. J., & Nadeau, T. (2007). *Physical Database Design : The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Burlington, MA, USA: Morgan Kaufmann.

Liu, Q. (2009). Approximate Query Processing. In L. Liu & M. T. ÖZsu (Eds.), *Encyclopedia of Database Systems* (pp. 113-119): Springer US.

Nambiar, R. O., & Poess, M. (2006). *The making of TPC-DS*. Paper presented at the Proceedings of the 32nd international conference on Very large data bases, Seoul, Korea.

Olken, F., & Rotem, D. (1986). *Simple Random Sampling from Relational Databases*. Paper presented at the Proceedings of the 12th International Conference on Very Large Data Bases.

Olken, F., & Rotem, D. (1990). *Random sampling from database files: a survey*. Paper presented at the Proceedings of the 5th international conference on Statistical and Scientific Database Management, Charlotte, NC.

Olken, F., & Rotem, D. (1995). Random sampling from databases: a survey. *Statistics and Computing, 5*(1), 25-42. doi: 10.1007/BF00140664

Poess, M., Nambiar, R. O., & Walrath, D. (2007). *Why you should run TPC-DS: a workload analysis*. Paper presented at the Proceedings of the 33rd international conference on Very large data bases, Vienna, Austria.

Potel, M. (1996). MVP: Model-View-Presenter, The Taligent Programming Model for C++ and Java. Retrieved. Retrieved Novemver, 2014, from http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

Raman, V., & Hellerstein, J. M. (2002). *Partial results for online query processing*. Paper presented at the Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin.

Rudra, A., Gopalan, R., & Achuthan, N. R. (2012). *An efficient sampling scheme for approximate processing of decision support queries*. Paper presented at the ICEIS 2012 - The 14th International Conference on Enterprise Information Systems, Wroclaw, Poland.

Ruoming, J., Glimcher, L., Jermaine, C., & Agrawal, G. (2006, 03-07 April 2006). *New Sampling-Based Estimators for OLAP Queries.* Paper presented at the Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on.

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., . . . Zdonik, S. (2005). *C-store: a column-oriented DBMS*. Paper presented at the Proceedings of the 31st international conference on Very large data bases, Trondheim, Norway.

TPC. (2014). TPC BENCHMARK ™ DS Standard Specification Version 1.1.0. Retrieved April 30, 2014, from http://www.tpc.org/tpcds/

Vaisman, A., & Zimányi, E. (2014). Introduction *Data Warehouse Systems* (pp. 3-11): Springer Berlin Heidelberg.

Wang, Y. J., Wang, H. H., & Li, H. (2013). A Histogram Based Analytical Approximate Query Processing for Massive Data. *Applied Mechanics and Materials, 411-414*, 362. doi: 10.1145/2465351.2465355

Webb, L. M., & Wang, Y. (2014). Techniques for Sampling Online Text-Based Data Sets *Big Data Management, Technologies, and Applications* (pp. 95-114): IGI Global.

Wu, S., Ooi, B., & Tan, K.-L. (2013). Online Aggregation. In B. Catania & L. C. Jain (Eds.), *Advanced Query Processing* (Vol. 36, pp. 187-210): Springer Berlin Heidelberg.

APPENDIX A

COLUMN DEFINITIONS IN MULTIDIMENSIONAL CLUSTER SAMPLING VIEW

Table 8.1 Column definitions of TAB_REPOSITORY table

No	Column name	Data type	Primary key	Not null	Default value	Description
1	kt_tab_number	Integer	Yes	Yes		Table number
2	kt_fact_schema	varchar2(30)				Original fact schema name
3	kt_fact_tab_name	varchar2(30)				Original fact table name
4	kt_total_cnt	integer			0	Fact table size
5	kt_mcsv_schema	varchar2(30)				schema name
6	kt_mcsv_tab_name	varchar2(30)				CLUSTER table name
7	kt_attrcnt	integer			0	The number of key attributes

Table 8.2 Column definitions of COL_REPOSITORY table

No	Column name	Data type	Primary key	Not null	Default value	Description
1	kc_tab_number	integer	Yes	Yes		Table number
2	kc_level	integer	Yes	Yes		Key attribute number
3	kc_colname	varchar2(30)				Column name of key attributes
4	kc_coltype	varchar2(30)				Column data type of key attributes
5	kc_colndv	integer			0	Number of distinct values
6	kc_colmin	number(*,3)			0	Minimum value
7	kc_colmax	number(*,3)			0	Maximum value
8	kc_colparts	integer			0	The number of paritions

Table 8.3 Column definitions of LEAF_REPOSITORY table

No	Column name	Data type	Primary key	Not null	Default value	Description
1	kl_tab_number	integer	Yes	Yes		Table number
2	kl_leafno	integer	Yes	Yes		Leaf number
3	kl_secno	integer	Yes	Yes		Section number
4	kl_elements_cnt	integer			0	Sample size in the section
5	kl_range_s01	number(*,3)			0	Start value of key attribute 01
6	kl_range_e01	number(*,3)			0	End value of key attribute 01
7	kl_range_s02	number(*,3)			0	Start value of key attribute 02
8	kl_range_e02	number(*,3)			0	End value of key attribute 02
9	kl_range_s03	number(*,3)			0	Start value of key attribute 03
10	kl_range_e03	number(*,3)			0	End value of key attribute 03
11	kl_range_s04	number(*,3)			0	Start value of key attribute 04
12	kl_range_e04	number(*,3)			0	End value of key attribute 04
13	kl_range_s05	number(*,3)			0	Start value of key attribute 05
14	kl_range_e05	number(*,3)			0	End value of key attribute 05
15	kl_range_s06	number(*,3)			0	Start value of key attribute 06
16	kl_range_e06	number(*,3)			0	End value of key attribute 06
17	kl_range_s07	number(*,3)			0	Start value of key attribute 07
18	kl_range_e07	number(*,3)			0	End value of key attribute 07
19	kl_range_s08	number(*,3)			0	Start value of key attribute 08
20	kl_range_e08	number(*,3)			0	End value of key attribute 08
21	kl_range_s09	number(*,3)			0	Start value of key attribute 09
22	kl_range_e09	number(*,3)			0	End value of key attribute 09
23	kl_range_s10	number(*,3)			0	Start value of key attribute 10
24	kl_range_e10	number(*,3)			0	End value of key attribute 10
25	kl_range_s11	number(*,3)			0	Start value of key attribute 11
26	kl_range_e11	number(*,3)			0	End value of key attribute 11
27	kl_range_s12	number(*,3)			0	Start value of key attribute 12
28	kl_range_e12	number(*,3)			0	End value of key attribute 12
29	kl_range_s13	number(*,3)			0	Start value of key attribute 13
30	kl_range_e13	number(*,3)			0	End value of key attribute 13
31	kl_range_s14	number(*,3)			0	Start value of key attribute 14
32	kl_range_e14	number(*,3)			0	End value of key attribute 14
33	kl_range_s15	number(*,3)			0	Start value of key attribute 15

No	Column name	Data type	Primary key	Not null	Default value	Description
34	kl_range_e15	number(*,3)			0	End value of key attribute 15
35	kl_range_s16	number(*,3)			0	Start value of key attribute 16
36	kl_range_e16	number(*,3)			0	End value of key attribute 16
37	kl_range_s17	number(*,3)			0	Start value of key attribute 17
38	kl_range_e17	number(*,3)			0	End value of key attribute 17
39	kl_range_s18	number(*,3)			0	Start value of key attribute 18
40	kl_range_e18	number(*,3)			0	End value of key attribute 18
41	kl_range_s19	number(*,3)			0	Start value of key attribute 19
42	kl_range_e19	number(*,3)			0	End value of key attribute 19
43	kl_range_s20	number(*,3)			0	Start value of key attribute 20
44	kl_range_e20	number(*,3)			0	End value of key attribute 20

Table 8.4 Column definitions of CLUSTER table

No	Column name	Data type	Primary key	Not null	Default value	Description
1	cl_leaf	integer				Partition key
2	cl_section	integer				Sub partition key
3	Same elements as the fact table					