5-2016

Data Partitioning Methods to Process Queries on Encrypted Databases on the Cloud

Osama M. Omran
University of Arkansas, Fayetteville

Data Partitioning Methods to Process Queries on Encrypted Databases on the Cloud

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

By

Osama Ben Omran
University of Garyounis
Bachelor of Science in Computer Science, 1989
University of Teesside
Master of Science in Information Technology, 1998

May 2016
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Dr. Brajendra Panda
Dissertation Director

Dr. Susan Gauch
Committee Member

Dr. Merwin Beavers
Committee Member

Dr. Paul Cronan
Committee Member

ABSTRACT

Many features and advantages have been brought to organizations and computer users by Cloud computing. It allows different service providers to distribute many applications and services in an economical way. Consequently, many users and companies have begun using cloud computing. However, the users and companies are concerned about their data when data are stored and managed in the Cloud or outsourcing servers. The private data of individual users and companies is stored and managed by the service providers on the Cloud, which offers services on the other side of the Internet in terms of its users, and consequently results in privacy concerns [61]. In this dissertation, a technique has been explored to improve query processing performance while protecting database tables on a Cloud by encrypting those so that they remain secure. It shows how to process SQL queries on encrypted databases designed to protect data from any leakage or attack, even from the service providers. The strategy is to process the query on the Cloud without having to decrypt the data, and data decryption is performed only at the client site. Therefore, to achieve efficiency, no more than the exact set of requested data is returned to the client. In addition, four different techniques have been developed to index and partition the data. The indexes and partitions of the data are used to select part of the data from the Cloud or outsource data depending on the required data. The index data can be stored on the Cloud or server with the encrypted database table. This helps in reducing the entire processing time, which includes data transfer time from the Cloud to the client and also data decryption and processing time at the client.

ACKNOLEDGMENTS

Special great thank to my advisor Professor Brajendra Panda for his support and advice in my research, and for his great guidance and help during pursuing my PhD program. I also would like to thank Professor Merwin Beavers, Professor Susan Gauch and Professor Paul Cronan for serving on my committee; it is a complete privilege having you on my dissertation committee. I also would like to thank my mother and my wife for their great support before and during pursuing my PhD program. Special thanks go out to the all faculty and staff at the Computer Science and Computer Engineering department for their effort to learn and give the students great and wonderful knowledge. Special thanks are extended to the all staff of the University of Arkansas for all of their help.

DEDICATION

TO MY LATE FATHER…TO HIS PURE SPIRIT AND HEART.

TO MY MOTHER...TO HER BIG HEART.

TO MY WIFE FOR GIVING SUPPORT AND ADVICE…TO THIS WONDERFUL WOMAN.

TO MY KIDS FOR LISTENING TO ME AND DOING GREAT JOB…TO MY LOVELY BEAUTIFUL KIDS.

TO ALL MY BROTHERS AND SISTERS…TO THIS NICE FAMILY

TO ANYONE WHO SUPPORTING ME...TO THESE KINDLY PEOPLE

TO ALL PEOPLE WHO TRY TO BUILD THE FUTURE…TO THESE GREAT PEOPLE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

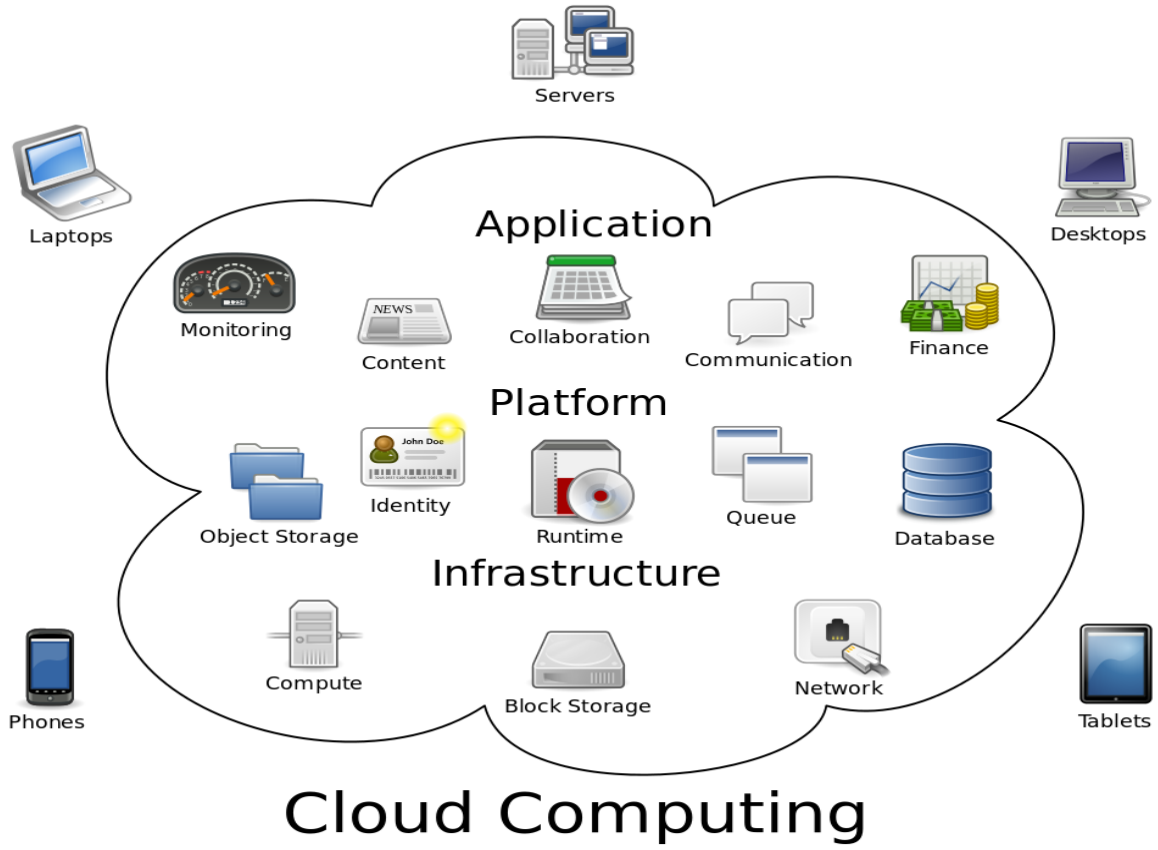
1.1 Cloud Computing

The Internet has been developing very rapidly in the last two decades. There has been a remarkable growth in the information technology use in every aspect of our lives, and this rapid growth has produced more challenges to conquer, including the high consumption of energy and the increased expenses of resources [81]. Therefore, a new technology has to be invented to avoid extra expenses and make work more efficient, which are the purposes for which Cloud computing has been invented. With the potential to significantly decrease the costs and, at the same time, increase productivity, cloud computing is a valuable technology [24]. By improving its cooperation, agility, and scale, Cloud computing could enable a truly global computing model over the Internet infrastructure. Furthermore, Cloud computing with its higher performance, can offer more scalable and fault-tolerant services [61]. Because of its high scalability, Cloud computing offers unlimited computing resources on demand, which is a service that can eliminate the need for Cloud service providers to plan the future on hardware provisioning [61]. In fact, Cloud computing has generated significant interest in the industry, though it is still an evolving paradigm. Cloud computing combines computing technologies and economic benefits through the evolutionary development of several existing approaches. This includes applications and spread services as well as information infrastructures, which include groups of computers, networks, and storage resources [24]. Nevertheless, this potentially revolutionizing computing paradigm could become a huge failure without appropriate security and privacy solutions designed for this technology. Security is one of the most important concerns when moving to the Cloud. Earning users' trust in Cloud providers occurs by providing the security of data in the Cloud [11]. Organizations and individuals fear leakage of their data, especially sensitive data,

when is stored in a Cloud unless they can be certain the Cloud is secure. As current Cloud services typically result in data being present in an unencrypted form on a machine owned and operated by a diverse organization, rather than the data owner, they pose an inherent challenge to data privacy. There are threats of unauthorized usage of the data by service providers, and there is the potential for theft of data from data servers in the Cloud. Thus, fears of sensitive data leakage or loss of privacy is a significant barrier to the adoption of Cloud services. For example, in 2007, criminals targeted the prominent Cloud service provider Salesforce.com, and succeeded in stealing customers' emails and addresses through a phishing attack [7]. As a result, some laws assigned different restrictions on the processing of personal and sensitive information by third parties; therefore, the use of Cloud services as they are currently being designed is constricted. Organizations and individuals want to save their money when they move into a Cloud. They try to reduce the total amount of the expenses when they transfer the data from or to the Cloud. The total expense of the transfer depends on the workflow execution time, the total amount of data transmitted from the consumer to the storage resource, the total amount of data transferred from the storage resource to the consumer, and the storage used at the resource in terms of GB-hours [63].

There are three Cloud delivery models. These models are software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). In SaaS, the Cloud providers enable and provide application software as on-demand services. Therefore, the provider’s applications can run on a Cloud infrastructure, and it is accessible from various client devices. PaaS enables programming environments to access and develop additional application building blocks. Such programming environments have an observable impact on the application architecture, such as constraints on which services the application can request from an OS [24]. In addition,

consumer-created applications installed on the Cloud infrastructure use programming languages and tools supported by the provider. Finally, in IaaS, the Cloud provider contributes a set of virtualized infrastructural components such as virtual machines and storage on which customers can build and run applications. The application will eventually reside on the VM and the virtual operating system. Figure 1.1 explains the idea of Cloud Computing.



Source: http://www.brighthub.com/environment/green-computing/articles/127086.aspx#imgn_1

Figure 1.1 Cloud computing concept

1.2 Advantages of Cloud Computing

Cloud computing has brought many advantages to people and companies throughout the world. The following list clarifies some important points in Cloud computing:

- Services or data are hosted on remote infrastructure

- Services or data are available from anywhere
- As Services or data are used, the user pays
- Use of software and hardware in economic ways
- Offers unlimited computing resources on demand
- Supports parallel and distributed computing
- Offers unlimited storage capacity
- Improves performance on the user's computer
- Increases data reliability
- Supports virtualization

1.3 Data Outsourcing

Information outsourcing and dissemination services have recently seen widespread diffusion because of the growing costs of in-house storage and management of large collections of sensitive data and the requests for both storage capacity and skilled administrative personnel [14]. Even though data outsourcing offers many important advantages like reducing management costs, higher availability, and more effective disaster protection than in-house operations provide, when data are not under the control of their owners, their confidentiality and integrity can therefore be at risk. Consequently, a solution to these data protection issues would then permit users and companies to utilize a dissemination service, offering strong guarantees about the protection of user privacy against both adversaries breaking into the system and the server itself. The servers need to perform a computation on their data without revealing the data used in the computation. Therefore, an effective and safe execution plan for a query computation, with which the servers can protect their information, has to be determined. Authors in [14] illustrate that data should be protected also from the server itself. The paper suggests some points: 1) the

server must not have any visibility on the data when it executes queries, and the DBMS must execute queries in encryption form. If the data is split into different fragments, we must have a technique to join them not on the server, but at the client site to answer any query. 2) We must have private access to the data, and it should not be possible for an observer or the server to infer two queries' aims at accessing the same or different data. 3) We must ensure the server does not improperly modify data, and that the server provides a correct response to queries. 4) A technique must be developed for enforcing access control in a reliable way to avoid causing a possible bottleneck and affecting the performance in the system. 5) It is easy to share information among multiple servers because of communication technologies. Indeed, confidentiality and privacy cares are still active problems that call for effective and efficient solutions.

1.4 Information Security

Security is an important field in computer science and information systems [83]. It is defined as feeling safe and protecting from thoughtful or accidental threats on the information, and the information is protected from access by unauthorized users. At the same time, authorized users must be able to access the data at any time. Information security means that the procedures and measures taken to protect data, hardware, software, networks, people, and procedures, which are involved in generating information [83]. The notion of information security is being established on Confidentiality, Integrity, and Availability, according to the National Security Telecommunication and Information Systems Security Committee. Confidentiality means the prevention of unauthorized users from knowing or accessing secret information and revealing secret information only to authorized users. Integrity means the data must be consistent and accurate throughout the system, and we must prevent the system from having invalid data, redundant data, inconsistent data, and data anomalies. Availability means that the system must be

available to authorized users to access the information at any time, and we must secure the system from external attacks, lack of system protection, lack of disaster recovery because of system failure, and faulty implementation of authentication processes [83].

1.5 Database Security

Information systems allow business corporations and government institutions to become more productive and efficient [83]. Organizations use the information to produce and get the right decisions to improve their work and tasks. Therefore, the organizations must have consistency and integrity in the data to make the right decisions, and the database administrator must implement and enforce security at all levels of the database. The database administrator must know the diverse security access points that can make the database vulnerable in order to protect valuable data stored in the database [83]. A security access point is a place where database security must be protected and applied. The security access points include people, applications, networks, operating systems, database management systems, data files, and data. In summary, database security is a collection of data constraints, security methods, security policies and procedures, and security tools blended to implement all necessary measures to secure the Integrity, Accessibility, and Confidentiality of every access point [83].

1.6 Database-as-a-Service

The previous explanation describes Cloud computing as the use of the internet to host computer resources instead of maintaining them on local computers, and accessing remote resources and using them via the internet. A company like Salesforce.com offers a Multi-tenancy technique. Multi-tenancy is an optimization mechanism for hosted services in which multiple customers consolidate onto the same operation system, so it will effectively drive down the cost of computing infrastructure [84]. Database-as-a-Service (DaaS) is a new paradigm for data

management, in which a service provider hosts a database as a service, which provides data definition, storage, backup, and retrieval. This will save the companies from buying expensive software and hardware [84]. Today, there are many companies that present a DaaS, like Google Datastore, Amazon, and Microsoft SQL Azure. On the other hand, the new technique should be developed to protect data from any attack. The security of the data on a Cloud or outsourced data is the responsibility of both the customer and the provider [84].

1.7 Encryption Technique

Encryption is the process and the operation of encoding or changing a message or any information so that its meaning is not understandable; the way to bring the information back into its original form is called decryption [82]. A cryptosystem is a system for encryption and decryption, which involves a set of rules for how to encrypt the plaintext and how to decrypt the cipher text [82]. The encryption is used to protect the data and make it particularly hard for an intruder to find the data useful. Encryption clearly deals with the need for Confidentiality of data and can be used to ensure Integrity. In general, data that cannot be read, cannot easily be modified in a meaningful manner [82].

1.8 Cloud Computing Performance

In designing a relational Cloud, resource allocation has become a great challenge. Challenges include monitoring the resource requirements of each workload, predicting the load multiple workloads that will produce when run together on a server, passing workloads on to physical servers, and migrating them between physical nodes [32]. Because of a running database, the resource monitor captures a number of DBMS and OS statistics. For example, it can define the memory space required by a workload, so that DBMS can fill the entire buffer pool with pages.

A model of CPU, RAM, and disk has been developed to predict the combined resource requirements when several workloads have been consolidated into a single physical server.

Cloud computing has the following essential characteristics that will affect Cloud performance [84]:

1. On-demand self-service: Without requiring human interaction with each service provider, users can order and manage services on the Cloud. This means the availability of a database at any time and the associated resources occur automatically at the provider.
2. Broad network access: Standard mechanisms and protocols used to access Cloud services, like computer applications and databases.
3. Resource pooling: Computing resources like storage, CPU processing, memory, network bandwidth, and virtual machines are provided to serve many users on the Cloud.
4. Rapid elasticity: Resources can be provisioned swiftly and elastically.
5. Measured Service: Resources and services are monitored and controlled automatically by Cloud systems, and they support optimization of resource usage and pay-as-you-go business models.

Depending on these characteristics, any database application can run on Cloud computing, and Cloud computing can run the database application with high efficiency if the Cloud uses high specification resources like high speed internet and extra memory. For example, the Internet provides access to a Cloud database and if the speed of the Internet is very high, the response time to any query will be received rapidly. On the other hand, it is practically impossible to measure accurately the database processing time on a Cloud database [84].

In [69], researchers say the key to the successful management of resources is local analysis and global analysis. Local analysis is the identification of the correct configuration of system

resources for a client, like CPU and memory, to meet the service level agreements while optimizing the revenue. A service provider provides global analysis. The service provider has to classify the decision for how to allocate resources among clients based on the current system status, such as defining the CPU, memory, and a new database replica.

1.9 Problem Definition

The fears of the users and companies using Cloud computing are summarized in this list:

- Leakage of sensitive data or any other important data
- Attacking the data on the Cloud
- Loss of privacy
- Unauthorized use of data
- Theft of data from storage devices
- Decrypting data on the Cloud
- Excessive data transmission from the Cloud
- Increased cost of use
- Decrypting huge data on the Client site

1.10 The Contribution of the Dissertation

The focus of this dissertation is to investigate how to protect the outsourced data or data on a Cloud, which allows users and companies to give their data to external servers that then become responsible for their storage, management, and dissemination. The suggested study will protect any database table on a Cloud or any external server from any leakage or attack. The research also focuses on minimizing data transmission from the external server, reducing data processing time on a client site, and avoiding decryption of unwanted data by decrypting only required data. The dissertation solution should protect any sensitive information on Cloud computing while

maintaining a competitive processing overhead and low price compared to other solutions. One of the disadvantages of the encryption method is increasing the time of processing when data are requested by any inquiry from a database table. For example, when we request "select * from employee where Salary > 10,000", the Salary field must be decrypted even on unmatched records, and it will take a long time to decrypt the salary attribute for the whole employee table [82]. Therefore, the proposed model has been designed to avoid this problem. The model encrypts the data over a Cloud or data center in order to protect it from any leakage. A technique has been presented to encrypt the data before it is sent to the Cloud and to execute and run SQL queries on a Cloud over encrypted data. The strategy is to process the query at the service provider's site without having to decrypt the data. Decrypting the data is accomplished only at the client site to enforce security. In addition, four different techniques have been developed to index and partition the data. The dissertation will performed simulation analyses of the approaches to test their effectiveness. The results are provided in the form of graphs and tables to show and compare the efficiency of the proposed methods. The indexes and partitions of the data are used to select part of the data from the outsourced data depending on the required data. Figure 1.2 shows the general idea of the dissertation plan.

1.11 Research Advantages & Goals

The advantages and the goals of the dissertation can be summarized in this list:

- Developing an algorithm to protect data from attack
- Encrypting data on a Cloud by using a standard technique
- Running SQL over encrypted data on a Cloud
- Decrypting data only at the client site
- Minimizing the amount of data transmission from a Cloud

- Filtering and giving the right result to the client at the client site
- Reducing data processing at the client site
- Executing SQL queries on a Cloud over encrypted data
- Developing four techniques to partition the data
- Evaluating and comparing the four techniques

The Client

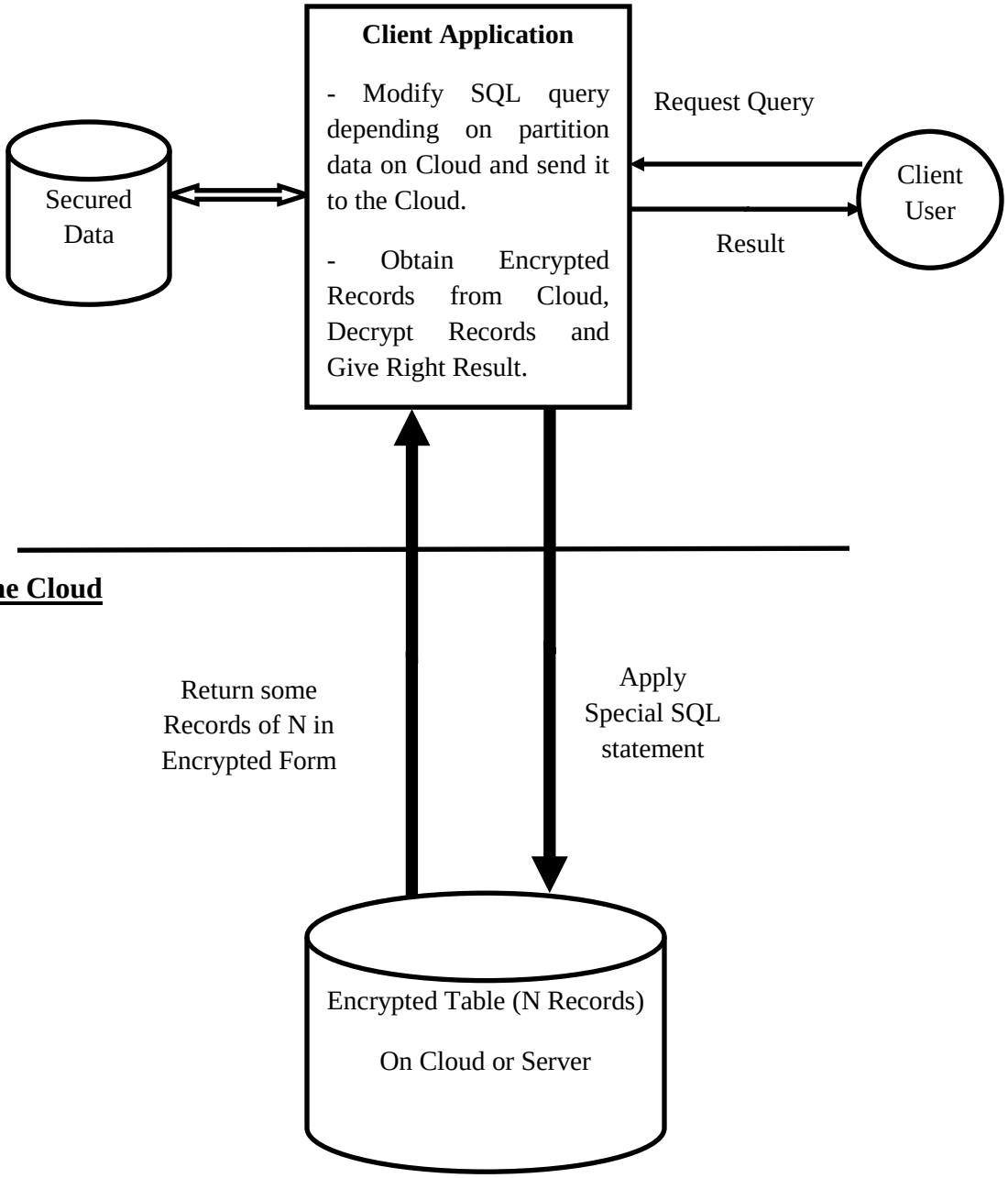


Figure 1.2 The Dissertation plan

2. BACKGROUND AND RELATED WORK

2.1 Introduction

Protecting a database table on outsourced databases or on Cloud has proven to be a very challenging task. It is a very important and difficult research field. Some research has been conducted over outsourced and Cloud database security as well as how to protect sensitive and non-sensitive information on outsourced data. While these papers give perfect solutions to the problem of data outsourcing, many of them depend on increasing the cost of the processing on the server and the clients' sites. Little research has been performed on how to protect outsourced data on Cloud with low processing overhead. Thus, developing mechanisms that protect sensitive or non-sensitive data in an outsourced database at the lowest cost is a key demand due to the amount of harm that can be caused by attackers. This research plans to investigate how to protect outsourced data or data on Cloud with the least processing overhead and at a low cost.

2.2 Cloud Computing Vulnerability

These days' blog entries, news, and many other publications warn about cloud computing security risks and threats. Security is cited as the most substantial roadblock for users moving to cloud computing [65]. Vulnerability is a major factor and is a weakness in the security system. Cloud computing technologies and applications have vulnerabilities that are prevalent in the development of the implementations, or are fundamental to the technology like insecure cryptography [65]. The following are some examples of Cloud vulnerabilities: unauthorized access to management applications, internet protocol vulnerabilities, data recovery vulnerability, and metering and billing avoidance. In addition, there are injection vulnerabilities, which process applications input to execute against the programmer's objectives. Injection vulnerabilities

include SQL injection, command injection, and cross-site scripting. Moreover, an attacker might take a virtual machine image from an untrustworthy source and analyze configuration and code in detail using administrative rights. This vulnerability can cause OS or application vulnerabilities to spread over many systems, and can lead to data leakage problems affecting communication, the physical disk, and Web applications and services.

2.3 Data Outsourcing and Security Issues

The suggested study is to protect any database table on a Cloud from leakage or attack and maintain efficient processing on the protected data. Therefore, to gain knowledge about this study and technique, searches, analysis, and reading on data security papers has been conducted. Many data security papers were found and this study focuses on Cloud computing and securing data outsourcing. These papers use different techniques to secure and protect data in these systems. Consequently, the study and search is categorized depending on this kind of security. A matrix has been developed and presented in Table 2.1. The research papers have been divided into five topics: Cloud Computing & Performance, Securing Data, Securing Outsourced Data, Securing Outsourced Data by Categorization, and Security Testing.

Table 2.1 Research Papers

Topic	REFERENCES
Cloud Computing & Performance	[32][63][65] [67] [69][70][71][72][73][74][75][76][77][78][79][81][99][100]
Securing Data	[5][10][15][21][23][26][29][31][34][35] [36][40][49][55][59][68][88][96][98]
Securing Outsourced Data	[1][2][3][4][6][7][8][9][11][12][13][16][17] [20][22][24][25][30][33][37][38][39][41][42]

	[43][44][45][46][47][48][50][61] [62][64][66][85][89][90][91][92][93][95][97]
Securing Outsourced Data by Categorization	[14][18][19][28][80][87][101]
Security Testing	[27][51][52][53][54][56][57][58][60]

2.4 Protecting and Executing Outsourced Data

Different researchers have introduced different techniques and methods to protect outsourced data. In [19], the authors designed a technique that protects data against attacks on the service provider. The paper discusses how that technique executes and runs SQL statements over encrypted data, and it explains how the technique is designed to run as much of the query as possible at the service providers' site on encrypted data. The client site is responsible for decrypting the data and managing the rest of the query processing to give the user the correct result. To decrease the computation at the client site, an algebraic framework that divides the query was designed. The proposed technique processes the query over encrypted data. Each relation in the database is stored on the server as an encrypted relation, and the technique stores an index corresponding to the attributes in the relation. The index is used to process any query at the server. In this paper, we find an example of how to partition an attribute that lies in range [0, 1000] into five partitions by using Equi-width partitioning. Different partition functions may be used to partition any attribute in a table. The paper suggests using AES, RSA, Blowfish, or DES methods to encrypt the tuples.

In [14] the authors discuss new privacy and security concerns when users and companies store their data in external servers, which then become responsible for their storage, management, and

distribution. They explain privacy threats in an outsourced database where trust in the service provider is very low. The paper suggests two methods to protect the data by encrypting the information outsourced to the server or on splitting information across several servers or tables. The paper begins by explaining the different issues that need to be investigated in the relationship between providing privacy and security of data outsourced to external parties. The paper suggests encrypting the data using either symmetric or asymmetric encryption schemes before outsourcing them to any server. Encryption can be applied on tables, attributes, or tuples. In the encrypted representation of the table, it is not possible to extract any subset of the tuples; so the method demands communication from the requesting client of the whole table involved in a query, and required an excessive workload for data owners and clients in encrypting and decrypting data. Additional indexing information is stored together with the encrypted table to query directly the encrypted data. The indexing information will be used to select specific data to return from the server in response to a query. This means the server executes a query and returns a set of encrypted tuples to the client, and the client decrypts them and discards spurious tuples by executing the remaining query on the client and giving the result to the user. A secure hash function satisfies important properties that turn out to be fundamental for the definition of an index. It has these properties: (1) the application of a secure hash function to a given attribute value always produces the same index value, thus making easy the translation of a query into an equivalent query on the server on the encrypted data. (2) A secure hash function produces collisions, meaning that different plaintext values map onto the same index value. This property guarantees that even if an adversary knows the distribution of plaintext values in the original database, it is not possible to infer the corresponding plaintext values from the index values, (3) a secure hash function does not preserve the domain order of the attribute on which it is applied. In

addition, the paper suggests two points for defining the indexing method for an attribute. First, the indexing information must relate to the data to provide an effective query execution mechanism. Second, designers should not open the door to inference and connecting attacks because of the relationship between indexes and data.

In [85] the authors developed a framework called ZeroVis to protect outsourced data in the Cloud and provide confidentiality for information stored in the Cloud. ZeroVis merges the ability to search over encrypted data with fine-grained access control on outsourced data. Their framework uses layered encryption in combination with Ciphertext Policy Attribute Based Encryption (CP-ABE) to support efficient query processing on encrypted data, and they utilized CP-ABE to control access to data based on the data client's attributes. Researchers created the ZeroVis framework on two primary building blocks, CP-ABE and CryptDB. CP-ABE generates the user's secret key by combining a user with a set of descriptive attributes. Users whose attributes match the access policy can decrypt the data, which is encrypted by an access policy. CryptDB preserves confidentiality for data stored on an untrusted database server, and it incorporates an encryption strategy that can adjust the encryption level of each column based on user queries and query requirements. The framework uses the ZeroVis proxy, which is responsible for encrypting data and queries, storing the encrypted data, and decrypting query results. The framework uses an attribute authority to provide authenticated attributes for each authenticated user and to prevent unauthorized users from sending queries through the framework.

Authors in [38] describe a model based on a client-based privacy manager in order to decrease users’ fears of data leakage and loss of privacy. It uses the idea of employing obfuscation and de-obfuscation of data to reduce the quantity of sensitive information held on the Cloud. The

obfuscation and de-obfuscation of the information are managed by a key, which is defined by the user, and it is not shown to Cloud service providers. Authors in [11] have described an insider threat in Cloud Relational Database Systems. The paper explains how to develop and make a knowledgebase in a Cloud Relational Database System to monitor user activities and mitigate insider threats. The paper suggests three models, which are the Peer-to-Peer model, the Centralized model and the Mobile-Knowledgebase model, and addresses the conditions under which they work well to prevent insider threats. Authors in [28] show how to offer means of protecting information, while guaranteeing its availability to legitimate clients. They offer a solution for remote querying of encrypted databases on untrusted servers. The solution bases on the use of indexing information attached to the encrypted database. The indexing is used by the server to select the data to return in response to a query, without the need for revealing the database content. The authors provide a method to indexing encrypted data constructed with efficiency and confidentiality in mind, providing a balance between the two. They also provide a measure of inference exposure of the encrypted and indexed data that nicely models the problem in terms of graph automorphisms. They enhance the indexing information to provide for efficient execution of interval-based queries by adapting the B+-tree structures to the database-service-provider model usually used inside DBMSs. The proposed index bases on a one-way secure hash function that takes in input of the plaintext values of an attribute and returns the corresponding index values, and the hash-based indexing offers more protection because the same index can be mapped by different plaintext values. Authors in [50] treat some security issues in a cloud database system. These issues are database and chosen plaintext attack knowledge threat and database and query result knowledge threat, which happens when an attacker hacks into the Cloud database. The authors suggest that a data owner must encrypt his data before he sends it to

the Cloud. They involve a secret sharing scheme between the client and the Cloud where a sensitive data item splits into two shares, one kept at the Cloud, which is the encrypted value, and the second at the client, which is the item key. They suggest a secure query processing technique and a set of elementary operators to work on relational tables, which contain encrypted data on the Cloud. The operators are data interoperable and can be used to formulate complex queries. At the end, only the client can decrypt the requested data.

Authors in [87] suggest an approach to execute and run SQL queries on the Cloud over encrypted data. The strategy is to process the query at the service providers' site without having to decrypt the data and achieve efficiency by returning no more than the exact set of requested data to the client. Decrypting the Data is accomplished at the client site to prevent any leakage at the Cloud or during transmission. They suggest storing the data on the Cloud in a different format by using a coding table, and all the attributes of a table are stored as one attribute on the cloud. In addition, each attribute uses different size code and different coding tables. Authors in [95] said the key area of concern in the acceptance of the Cloud is the security of the data and a very high degree of privacy and authentication. Cryptography is one of the important methods to protect the data in a Cloud database server. The paper suggests using cryptography to provide various symmetric and asymmetric algorithms to secure the data. The paper presents the symmetric cryptographic algorithm, which is the Advanced Encryption Standard (AES), and AES depends on several substitutions, permutation and transformation. It states that the AES algorithm is a highly secure encryption method.

2.5 Cloud Computing and Performance

Managing and allocating resources among different clients and users intelligently is very important for system providers, who manage the infrastructure resources in a cost-effective

behavior while satisfying the client service level agreements [69]. Authors in [69] concentrate on this fact to manage the resources in a shared cloud database system and present smart service level agreements. They suggest using machine-learning techniques to learn a model, which explains the potential profit margins for each client under several resource allocations. Using this learned model, the resource allocations are dynamically being regulated by the resource allocation decision module in order to achieve the optimum profits. The technique proves efficient for computing predictive models under different hardware resource allocations, like CPU, memory, and number of replicas in the database systems.

In [73], the researchers say that performance unpredictability becomes the most important problem in Cloud computing because database investigators implement wall clock experiments, and database applications offers service level agreements. They suggest a study of the infrastructure performance variance of Amazon EC2 from different perspectives. They utilize well-known micro benchmarks to measure performance variance in communication, CPU, and I/O, and to quantify the impact on real data-intensive applications, they utilized a multi-node MapReduce application. The authors collected and compared data for an entire month with the results obtained on a local cluster. The results show that EC2 performance varies a lot and often falls into two bands having a large performance, and these two bands correspond to the different virtual system types provided by Amazon.

3. DATA PROTECTION AND STORAGE

3.1 Introduction

Companies and users use computers to process and store their data for many reasons. They can send their data through the internet or share their data through computer networks with different users. In addition, they can use data outsourcing or Cloud computing to store their data in external servers. The companies and users are concerned about their stored data, and they want and desire to protect their data. They only want authorized people to use their data. Some solutions and techniques can be used to protect the data in the Cloud or on servers. One of the important solutions to protect information is to use encryption techniques. Data encryption prevents unauthorized people or intruders from seeing others' data and protects data in storage devices. By using encryption techniques, no one can read the encrypted data, and only the users with the decryption key can decrypt it. On the other hand, an efficient technique has to be used to manage data outsourced or in a Cloud, or companies and users will not use them if responding to queries takes long time.

3.2 Data Outsourcing and Cloud Computing

A new technique data outsourcing has been developed, because of the rapid evolution of storage, processing, and communication technologies [14]. Providers and external servers become responsible for storage, management, and dissemination for any companies or users who give their data to them. Using data outsourcing decreases the cost of the software and the hardware and offers high availability. Nevertheless, data are not under control of the companies or users, and attackers can violate their information. This will affect their confidentiality and integrity. A

perfect solution to this issue will allow companies and users to use a dissemination service and give their data to the providers and servers.

3.3 Data Encryption

Users and companies are concerned about their data when data are stored and managed in a Cloud or outsourcing servers. The main solution is to encrypt data before they send data to a Cloud or outsourcing servers to prevent the server or provider from accessing the data that are stored on its own machines. Data encryption can be performed by using symmetric or asymmetric encryption schemes. Many suggestions are based on symmetric encryption because symmetric encryption is cheaper than asymmetric encryption. When a client requests any information from an encryption table, the client has to request the whole table because it is not possible to filter the tuples in the encrypted representation of the table. This will require an excessive workload, demanding encryption and decryption of the data. To decrease this workload, additional indexing information can be added to the original database table, and these indexes are stored together with the encrypted tables on the Cloud or the server. The index data can be used by the database management system to select required data from a server or a Cloud to be returned in response to a query.

3.4 Processing Query on Encrypted Database Table

The index data can be stored on the Cloud or server with an encrypted database table. This helps in reducing the entire processing time, which includes data transfer time from the Cloud to the client and also data decryption and processing time at the client. In the meantime, the data are protected on the server. This method also avoids decryption of unwanted data by decrypting only the data requested by a user. This technique can help us to process a query on a Cloud or server without having to decrypt the data, and data decryption is performed only at the client site to

protect the data from any leakage. When the data are being indexed and partitioned by any technique, the same technique must be used to index and partition the data in a query before the query is sent to the Cloud or server. Generally, when a user submits a query, he has to map the query into a server query working on the encrypted table at the server site. Once the server executes the query, the server will return a set of encrypted records to the client that decrypts them and chooses only the right records, giving them to the user. Some research has been discussed in chapter two which use different methods and techniques to partition and index the data. These papers have also illustrated some important points when defining the indexing.

Definition 1 (Set Partition) [86]. Given a set A and an index set I, then $\{A_i\}_{i\in I}$ is a partition of A if

1. $A_i \neq 0$ and $A_i \subseteq A$ for each i∈I
2. $A=\bigcup_{i\in I} A_i$
3. $A_i \cap A_j = 0,\ for\ all\ i,j \in I\ where\ i \neq j$

3.5 Data Partitioning Methods

In this research, various ways of storing a database table on a Cloud, where the records are encrypted and divided into multiple partitions for secure and effective management, have been studied. The strategy is to process as many queries as possible at a Cloud or server site without having to decrypt the data. This processing will minimize computation at a client site. Therefore, minimum processing will be needed at the client site to get the right result. The technique starts by defining the attributes that will be used in all queries, and these attributes will be processed by some operations to get the partition categories. By getting the partition categories, each value in any attribute can be mapped to a specific range. This means to map the domain values of the attributes into the partition categories, such that these partitions cover the whole domain, and no two partitions overlap. For example, consider the employee table as shown in Table 3.1, as well

as Figure 3.1, which shows the identifiers assigned to the five partitions of the Sal and NetSal attributes. By using one of the encrypted methods and Figure 3.1, Table 3.1 can be sent as an encrypted table with extra category information to the Cloud as shown in Table 3.2. The extra category information is the Sal and NetSal attributes mapping corresponding to the index partition. For example, if Sal is equal to 450, its corresponding partition is [400, 600]. Since this partition is mapped to 3, it stores the value "3" as the identifier of the Sal for this tuple. Similarly, it stores the NetSal attribute with the identifier "2" for the NetSal value 300. For simplicity, the range of the partition is used as 200 for the two attributes.

Table 3.1 Employee Table

Record No.	ID	Name	Sal	NetSal
R1	14	AC	450	300
R2	15	DD	500	375
R3	16	BC	950	550

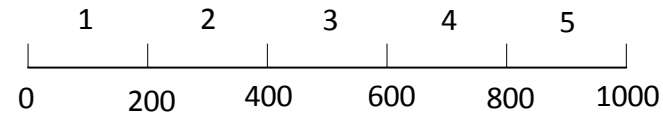


Figure 3.1 Partition and identification of the Sal and NetSal attributes.

Table 3.2 Encrypted Employee Table on the Cloud

Record No.	Whole Record Encrypted	Sal	NetSal
R1	71740103000000007179787871707...	3	2
R2	71750404000000007179777071707...	3	2
R3	71760203000000007179767671717...	5	3
...	.....	...	...

3.5.1 Type of Partitioning

To improve security, each attribute can use a different range of partitions for its values. By using the category data in the WHERE clause conditions, any query can be run to bring the data from the encrypted table on the server or the Cloud to the client site. To encrypt the data, one of the

standard algorithms like the Data Encryption Standard (DES), Advanced Encryption Standard (AES), or Rivest-Shamir-Adleman (RSA) can be used. For the partitioned data, four different approaches have been proposed to get the partitioning categories as explained below:

1. Frequency of use Based Partition

This kind of partitioning method begins by mining the log file of the database system in a period of time to be put on a server or Cloud, and by using this process a statistical matrix for all WHERE clause conditions of queries that will be executed on those tables is generated. This statistical matrix will be created for each attribute of the data that needs to be partitioned into categories. Figure 3.2 shows an example of a log file. Figure 3.3 shows the Statistical Matrix for a salary attribute in a table. It shows how many times a partitioning category has been used in WHERE conditions. After the statistical matrix is created, the technique eliminates any partitioning category that is less than a predetermined threshold. Figure 3.4 shows an example of how the technique eliminates some partitioning categories because they are under the threshold. Table 3.3 explains the number of records in each partitioning category, and by using this table, it can demonstrate how getting the smallest partitioning category can improve performance. For example, partitioning category [800, 1200] has 200 transactions and 30 records from the table; so in total it has to bring 200 *30 =6000 records from the server table in that given period of time. However, if we choose partitioning category [400, 1200], it has 20 +30 =50 records and 200 *50=10000 records in that given period of time when a user requests the query looking for a value between 800 to 1200. Therefore, breaking this category [400, 1200] into two different categories [400, 800] and [800, 1200] will improve the performance. The net improvement is 10000-6000=4000 records and it will correctly skip retrieval of 4000 records during this period

because the technique requests the data only form [800, 1200]. Table 3.4 shows the resulting partitioning categories.

List all where clause conditions:
Where salary>=400 and salary<=750
Where salary>=400 and salary<=800
Where salary>=400 and salary<=750
Where salary>=700 and salary<=1200
Where salary>=5000 and salary<=7000
Where salary between 1000 and 3000
Where salary>=1200 and salary<=2500
Where salary>=400 and salary<=4000
Where salary>=400 and salary<=800
Where salary between 400 and 2500
Where salary between 2500 and 5000
Where salary>=2500 and salary<=7000
.....

Figure 3.2 The Log file

Partition to / Partition From	750	800	1200	2500	3000	4000	5000	7000
400	5	100	50	200	7	100	100	50
700			15	3	1	0	0	5
800			200	100	4	200	100	50
1000				13	10	5	2	5
1200				300	5	200	200	100
2500					5	300	200	100
4000							600	100
5000								200
5500								3

Figure 3.3 Statistical Matrix

Partition to / Partition From	800	1200	2500	4000	5000	7000
400	100 (20)	50	200	100	100	50
800		200 (30)	100	200	100	50
1200			300 (30)	200	200	100
2500				300 (25)	200	100
4000					600 (10)	100
5000						200 (5)

Figure 3.4. The result Statistical Matrix (Threshold=25)

Table 3.3 Number of Record

From	To	No. of Records
400	800	20
800	1200	30
1200	2500	30
2500	4000	25
4000	5000	10
5000	7000	5

Table 3.4. The Result Partitioning Category

Partitioning Category	Category
[400 - 800]	1
[800 - 1200]	2
[1200 - 2500]	3
[2500 - 4000]	4
[4000 - 5000]	5
[5000 - 7000]	6

In addition, attributes may have discrete data, which can take only particular values. Discrete data can be numeric, like numbers of students, and can be categorical, like male or female and manager or computer programmer. The next example explains how to index and partition discrete data. For example, consider the attribute Job Title, which has discrete values, and possible values for this attribute are Manager, Computer Programmer, Database Designer, or Accountant. After the log file has been studied for this attribute, the Statistical Matrix will be created, which is shown in Figure 3.5. The method next removes any partition category less than the predetermined threshold, which is 90. Figure 3.6 confirms this step by eliminating some partitioning categories that are under 90. After that, it collects all the eliminated categories and puts them together in one category, or each group in different category depending on the total number of the frequency for each category. Figure 3.7 shows the result of Partitioning Categories.

Job Title	Manager	Computer Programmer	Database Designer	Accountant
Manager	250	50	40	200
Computer Programmer		30	20	20
Database Designer			20	30
Accountant				300

Figure 3.5 Statistical Matrix for Job Title Attribute

Job Title	Manager	Accountant
Manager	250	200
Accountant		300

Figure 3.6. The result of Statistical Matrix (Threshold=90)

1	2	3
Manager	Computer Programmer & Database Designer	Accountant

Figure 3.7. The result of Partitioning Category

2. Space Based Partition

This kind of partitioning category starts by counting the frequency of each value in an attribute of a table that needs to be partitioned and put on the server or the Cloud. The attributes can be defined by studying the log file. After the values and their frequency have been defined, they will be stored in a statistical table. This statistical table must be in ascending order before the next operations start. Table 3.5 shows an example of values and their frequency. Next, the capacity of a bucket used for each category is defined. This bucket will be used as the size of each partition category. The technique starts by reading the statistical table from the beginning of the list and checking the frequency of each value to see if the bucket can store the frequency of each value. If the size of the bucket is bigger than the size of the frequency, the value of this frequency can belong to the current bucket. Then the process checks the second value. If the remaining size of

the bucket is bigger than the size of the frequency, the second value will be placed in the current running cloud category. If the space left in that bucket is not large enough to store the entire category, it checks the next value, and so on. If the size of any value is greater than the size of the bucket, it divides the value over many buckets. Table 3.6 depicts an example of this process. The reason for having the original and Cloud category attributes is that the table works as a dynamic table where, if the size of any frequency is more than the size of the bucket, it can update the category data on the Cloud without bringing the data to the client and decrypting the data to update the table. This idea also works for discrete data because the technique works on separate values.

Table 3.5 Statistical Table (Values and their frequency)

Value	Frequency (Number of Record)
450	100
500	1000
1000	300
2000	200
3000	150

Table 3.6 Dividing Values of Frequency on Buckets (Bucket Size=500)

Value	Frequency	Original Category	Cloud Category
450	100	1	1
500	1000	2.2	2 (2.1, 2.2)
1000	300	3	1
2000	200	4	4
3000	150	5	4

3. Mondrian or Bisection Tree Based Partition

This method starts by ordering the data of any attribute that needs to be partitioned and calculating the median, then dividing the data into two partitions, right and left. It then takes the median for each partition and divides them into two partitions, right and left. This operation

repeats for each partition until the partition satisfies certain termination conditions. To explain this idea, let us use Table 3.5, which shows the values and their frequency, and the termination conditions as the total values of the frequency in each group are less than or equal to 500, or there is only one value in each partition. Figure 3.8 shows how the Bisection-Tree works. For example, in the second step in Figure 3.8 the group "1000, 2000, and 3000" is divided once more, because the total value of the frequency is greater than 500, which is 300+200+150=650 records. When it is divided into two groups, the total in the first group, which is 1000, is equal to 300 records, and the second group, which is 2000 and 3000, has 200+150=350 records which is less than 500. Table 3.7 shows the resulting category. This design will work for discrete data as well.

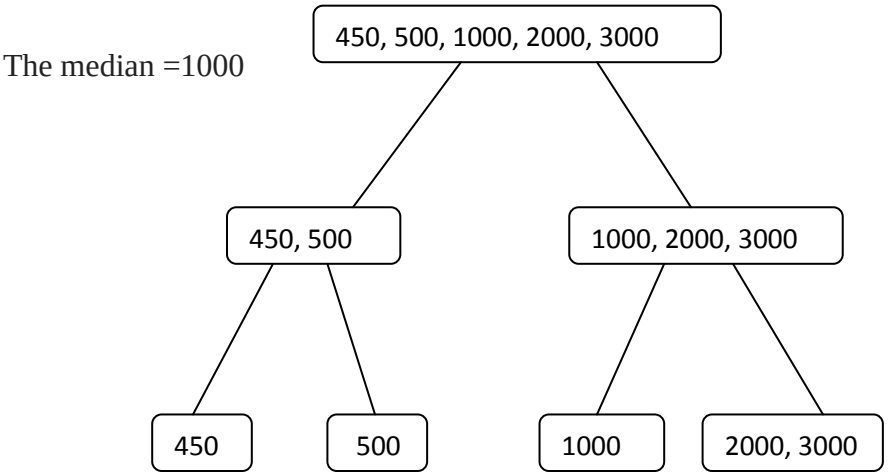


Figure 3.8. The Bisection-Tree.

Table 3.7 The resulting Category

Value	Frequency	Category
[450]	100	1
[500]	1000	2
[1000]	300	3
[2000 - 3000]	350	4

4. Histogram Based Partition

Histogram is a method of displaying statistical information, and Equi-width is one of its kinds. Equi-width technique, which divides the values into buckets of equal width, can be used to partition the data. This method subtracts minimum value from maximum value for the attribute that will be partitioned and divides the results by the number of buckets. Authors in [19] used this technique to partition the data. Figure 3.9 shows the identifiers assigned to the five partitions of an attribute. In this research, this method will be compared against the previous three methods to discover the efficiency. Table 3.8 shows the result of the partitioning category.

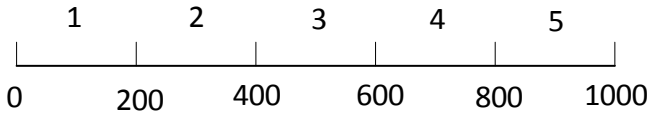


Figure 3.9 Partition an attribute into 5 buckets.

Table 3.8 The result of Partitioning Category

Partitioning Category	Category
[0 - 200]	1
[200 - 400]	2
[400 - 600]	3
[600 - 800]	4
[800 - 1000]	5

3.5.2 Partitioning Algorithm

In this part, the four kinds of the partition process are presented as steps of algorithms to show how to implement each method.

1. Frequency of use Based Partition Algorithm

This algorithm shows how to apply frequency of use based methods. The algorithm starts by reading the log file at a given time, and at the end, the algorithm gives the partitioning categories.

Input: The log file

Output: the resulting is the partitioning category.

1. Initialize Partition set={}
2. Read the log file and collect all WHERE commands for the attribute that needs to be partitioned and put them in an array called Statistical_Matrix array, which explains partition from and partition to by counting the frequency of each range.
3. Sort the Statistical_Matrix array by sorting the first column, which describes Partition From, and first row, which describes Partition To, in ascending order.
4. Define a threshold for minimum value of frequency to eliminate any partition category less than the threshold.
5. Get new Statistical_Matrix after eliminating any partition category less than the threshold.
6. List=0 (before the first row in Statistical_Matrix)
7. List=List+Next_Row
    a. Take the value from first column as Partition_From with a value of the first column in the same row that has a number as Partition_To.
    b. Range values Partition_From and Partition_To ∉ Partition set
        i. Partition set= Partition set U (Partition_From and Partition_To)

        (Check Partition set if the range values Partition_From and Partition_To are not found before; put the range values in Partition set)
8. Go to step 7 if the process has not processed the last row in the Statistical_Matrix array.
9. Collect any domain, which is not found in the Partition set and put them together in one category or each group in different category.
10. Put random numbers for each partition to get the result partitioning category

11. Return (Partition set)
12. End

2. Space Based Partition Algorithm

This algorithm shows how to apply space based methods. The algorithm starts by reading a database table that needs to be partitioned and put on the Cloud or server, and at the end, the algorithm gives the partitioning categories.

Input: Database table that contains the attributes to be partitioned.

Output: The partitioning category.

1. Initialize Partition set={}
2. Define the attribute that needs to be partitioned in the database table
3. Put each value and its frequency in the Partition set
4. Sort the Partition set in ascending order
5. Starting from number=1, Give each value in the Partition set an increment number by 1, and it is called original category
6. List=0 (beginning of the Partition set)
7. List=Main_row+1, bucket_size=Maximum size, Main_row=List
8. If (Cloud_category != space) goto 11.\\ space means has not partitioned it before.
9. Else If (frequency>bucket_size)
    a. Number of bucket= ⌈ (frequency/bucket_size) ⌉ \\Divide the frequency on number of buctets
    b. Cloud_category=List number of category buckets
    c. Put Cloud_category in the Partition set
    d. Goto step 11

10. If (frequency<=bucket_size)
    a. If (Cloud_category != space)

        List=List+1 \\Next row

        goto step 10

    b. Cloud_category=original category of the Main_row
    c. Put Cloud_category in the Partition set
    d. Bucket_size= Bucket_size-frequency \\To get the reminder of the bucket_size
    e. List=List+1 \\Next row
    f. If (bucket_size>0) go to step 10, else goto step 11
11. If (Main_row != end of the array_freq) goto step 7
12. Return (Partition set)
13. End

3. Mondrian or Bisection Tree Based Partition Algorithm

This algorithm shows how to apply Mondrian or Bisection Tree based methods. The algorithm starts by reading a database table that needs to be partitioned and stored on the Cloud or server, and at the end, the algorithm gives the partitioning categories.

Input: Database table that contains the attributes to be partitioned.

Output: The partitioning category.

1. Initialize Partition set={}
2. Define the attribute that need to be partitioned in the database table
3. Put each value and its frequency in an array
4. Sort the array in ascending order

5. Define the termination condition (Maximum frequency number of each group after dividing)
6. Take the median, so there are two groups, left and right partitions
7. If ((frequency of all values in right group <= termination condition) Or (only one value in the group))
    a. Stop dividing
    b. If (group of values), Put the smallest number (From value), largest number (To value), and the category maybe Random number or sequential number in the Partition set
    c. If (there is one value in the group), put the value and the category in the Partition set
8. If ((frequency of all values in left group <= termination condition) Or (only one value in the group))
    a. Stop dividing
    b. If (group of values), put the smallest number (From value), largest number (To value), and the category (maybe Random number or sequential number) in the Partition set
    c. If (there is one value in the group), put the value and the category in the Partition set
9. Else Goto step 6
10. Return (Partition set)
11. End

4. Histogram Based Partition Algorithm

This algorithm shows how to apply histogram based methods. The algorithm starts by reading a database table that needs to be partitioned and stored on the Cloud or server, and at the end, the algorithm gives the partitioning categories.

Input: Database table that has the attribute to be partitioned.

Output: the result is the partitioning category.

1. Initialize Partition set={}
2. Define number of bucket_size
3. Define the maximum and minimum values for the attribute that will be partitioned
4. Range= ⌐ ((maximum-minimum)/bucket_size) ¬
5. List=minimum value, bucket_no=0
6. List_To=List+Range
7. Put List, List_to, and category number can be Random number or sequential number in Partition set
8. List=List_To, bucket_no=bucket_no+1
9. If (bucket_no<bucket_size) Goto 6
10. Return (Partition set)
11. End

3.5.3 Applying Data Encryption

Each type of the data partitioning method described in the previous section has been applied on a relational table in order to evaluate and compare the time of encryption. These methods use an

index data, and these index data are stored with the encrypted table on a MySQL server. When data are sent to the MySQL server, the data are sent in encrypted form using a standard algorithm. Figure 3.10 shows the consumption time in milliseconds, which is used to encrypt an employee table that has 3000 records with a record size of 68 bytes, for each type. As can be seen from the graph, there are different values in encryption time for the same number of records for each kind. It can be clearly seen that encrypting the table by Frequency-of-Use-Based partition took the shortest time, and encrypting the table by Space-Based partition took the longest time.

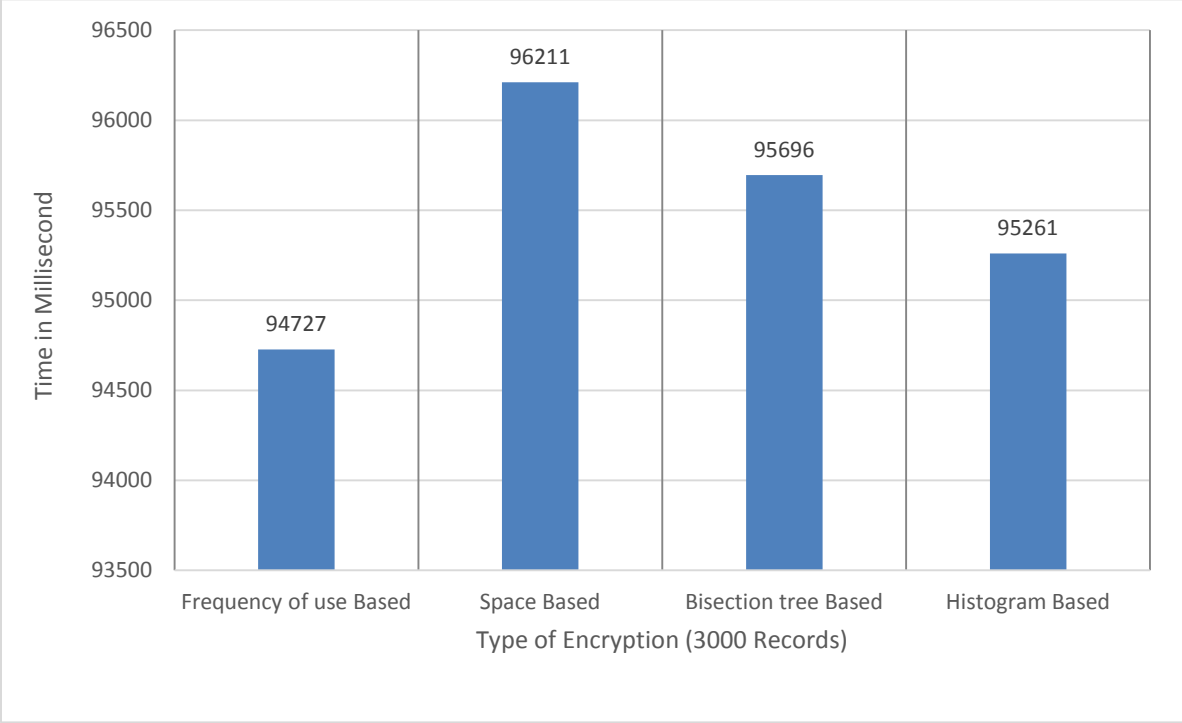


Figure 3.10. Running time to encrypt 3000 records for all partition types.

4. RESULTS AND ANALYSIS TYPES OF DATA PARTITIONING

4.1 Introduction

In chapter three, four types of data partitioning methods have been described with examples and implementation. These methods can be used for indexing data, and the indexed data will be stored with the encrypted table on a Cloud or server. The indexed data can be used to increase the performance when the data are requested from the encrypted table, which is stored on a Cloud or server. In this research, three new methods have been designed to partition and index the data. The new techniques are Frequency-of-Use-Based partition, Space-Based partition, and Mondrian or Bisection-Tree-Based partition. The fourth technique has been used by another paper [19] and will be used to compare the efficiency with the other three. In this chapter, the performance results, and the comparison of, the four techniques will be explained. In addition, these techniques will be compared against both unencrypted and encrypted tables that do not have category partition, to show the effectiveness.

4.2 Encrypt Data by applying one Type of Partitioning

By studying the log file database queries, the index attributes will be defined for each table. When data are sent to a Cloud, the data are sent in encrypted form using one of the standard algorithms. In addition to the encrypted data, additional index attributes are sent to the Cloud with each record. The index attributes can be found from a category list, which is created when a whole table is sent to a Cloud or server. The category list is created by applying one of the partition methods, which are explained in chapter three. The category list is used when data are encrypted or decrypted, and is used during fetching of information from a Cloud. The category

list should be stored on a client site to keep it secret from attackers. Also, in this category list any numeric variable is transformed into categorical counterparts.

4.3 Getting and Decrypting Data from the Cloud

When a query is defined at a client site to get data from a Cloud, some processing will be done on the client to modify the query. First, the index attributes and their values will be defined. Second, the name of index attributes will be changed to the equivalent name on the Cloud. Third, from the category list for each attribute, each value will be changed to the equivalent value on the Cloud. The process transforms numeric variables into categorical counterparts. The update query is sent to the Cloud or server and run on encrypted data. The strategy is to process as much of the query as possible at the Cloud site without having to decrypt the data. In addition, by applying this strategy, a set of records will be returned from the Cloud, and this processing will reduce the number of retrieved records and minimize the computation at the client site. Therefore, the remainder of the query processing is performed at the client site, by only decrypting the incoming data and giving the right result to the client, which will improve performance.

4.4 Managing the Query by Mapping Conditions

The translation of specific conditions in operations must be considered, which means translating the WHERE command condition to corresponding conditions over a Cloud site. As described previously, all tables are stored in an encrypted format on a MySQL server, and the numeric attributes which have to be used in the query conditions are stored depending on range partition. Therefore, when a user requests information from a Cloud by sending SQL statements from a client site, the given query has to be modified to be sent to the Cloud to work with the index partition. All comparison conditions {<, >, =, >=, <=, !=} must be considered, and the condition

has to be modified depending on the type of partitioning and the category list. For example, if there is a condition like (term1 > term2), this condition is managed by finding the maximum possible value that can be found for the term1 depending on the category list. The maximum possible value is found by taking the total maximum value from all positive attributes and the minimum value from all negative attributes depending on the partitioning scale. Next, the minimum possible value for term2 can be found by taking the total of minimum value from all positive attributes and the maximum value from all negative attributes depending on the partitioning scale. Therefore, depending whether the data partition stored on a Cloud is in encrypted form or not, the designers can generate some different processing to mange any condition.

4.5 Applying Data Partitioning

Each type of the data partitioning method described in the previous chapter has been applied on a relational table in order to evaluate and compare their effectiveness. In addition, the four kinds of data partitioning approaches are compared against the original unencrypted table and also the encrypted table that has no category data. Thus, six employee tables have been created with different structures each one satisfying the description in chapter three. All of these tables are stored on a MySQL server and processed in a simulated cloud environment. As explained in the following sections, different numbers of records have been created with different record sizes to achieve the evaluation.

4.5.1 Data Retrieved from A Table Having 100,000 Records with Record Size 68 bytes

In this section, the process to evaluate and compare the different partitioning techniques will be described using the employee table that has 100,000 records, with record size 68 bytes. These records have been added randomly to perform the evaluation. Table 4.1 shows the structure of

the employee table. In addition to this structure, two indexed attributes are added to test the four types of data partitioning. Table 4.2 shows different percentage numbers used to define the number of records retrieved from the employee table on the server, and it shows the WHERE clauses which were used. To do the evaluation, in addition to the original table, five different encrypted formats from the employee table have been created to test each type of data partitioning.

Table 4.1 Structure of the original Employee table (Record size=68 Bytes)

Attribute Name	Attribute Type	Attribute Size (Byte)
Id	int	4
Name	Varchar	25
Sal	Int	4
NetSal	Int	4
Address	Varchar	30
Sex	char	1
Total=		= 68 Byte

Table 4.2 Percentage of data requested by WHERE clauses

	Data requesting %	Where Clauses
1	10%	Where (Sal>=200 and Sal<300)
2	20%	Where (Sal>=800 and Sal<1000)
3	30%	Where ((Sal>=1 and Sal<200) or (Sal>=200 and Sal<300))
4	40%	Where ((Sal<1) or (Sal>=450 and Sal<650) or (Sal>=800 and Sal<1000))
5	50%	Where ((Sal>=300 and Sal<650) or (Sal>=650 and Sal<800))
6	60%	Where ((Sal>=0 and Sal<200) or (Sal>=450 and Sal<650) or (Sal>=800 and Sal<1000))
7	70%	Where ((Sal>=0 and Sal<300) or (Sal>=450 and Sal<650) or (Sal>=800 and Sal<1000))

4.5.1.1 Running Query on Table Having 100,000 Records with Record Size 68 bytes

The WHERE clauses as explained in Table 4.2 have been applied on the employee table with different structures depending on the type of the partition used. Table 4.3 shows the running time for each type of partitioning in milliseconds. Figure 4.1 shows the percentage of the retrieved

records from the employee table and the consumption time in milliseconds, which is used to retrieve the corresponding records for each type. As can be seen from the graph, there are different values of data retrieval time for the same percentage of records. The encrypted table without data category took the most time, but it took the longest when 60% of data were retrieved from the encrypted table with Space-Based category, because the WHERE clause has many different ranges. The graph also shows that the amount of retrieval time increased steadily for all types of partitions. In addition, the graph shows the encrypted table with the Bisection-Tree-Based partition and the encrypted table with Frequency-of-Use-Based partition are more efficient than the others and are close to the retrieval time from unencrypted table.

Table 4.3 Running time for each type of partitioning in milliseconds (size=100,000 records with record size 68 bytes).

Data requesting %	Unencrypted Table *	Encrypted Table without data Category *	Frequency use Based *	Space Based *	Mondrian or Bisection tree Based *	Histogram Based *
10%	971	1151	967	968	960	984
20%	1021	1199	1001	1027	1016	1029
30%	1068	1189	1065	1055	1050	1068
40%	1113	1235	1115	1151	1104	1153
50%	1140	1247	1166	1171	1154	1160
60%	1192	1246	1186	1256	1200	1233
70%	1211	1302	1237	1286	1247	1278

(*=Running time in Millisecond)

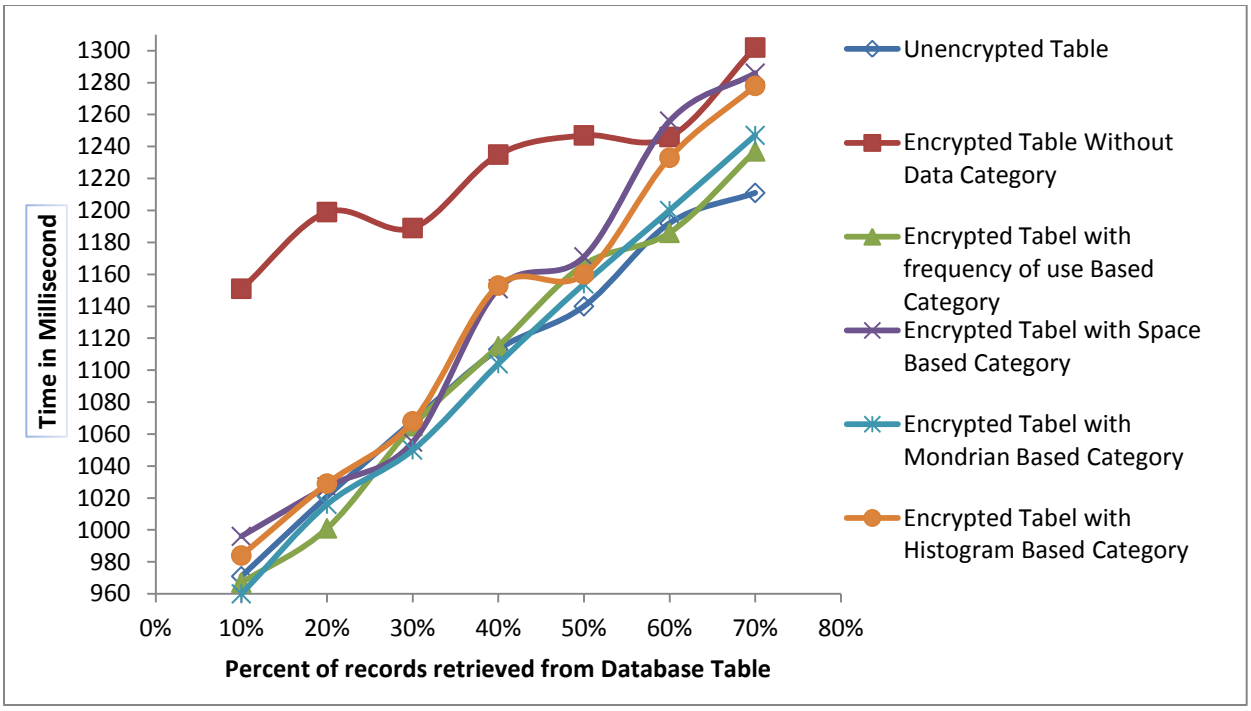


Figure 4.1 Data retrieval time from database table having 100,000 Records with record size 68 Bytes.

4.5.1.2 Records Retrieved from Table Having 100,000 Records with Record Size 68 bytes

By applying the WHERE clauses as explained in Table 4.2, the methods are tested using different percentages of retrieved records. Table 4.4 shows the number of records retrieved in various cases. Figure 4.2 shows the results in graph form. The graph shows that the encrypted table without data category retrieves all records, which is 100,000, because the WHERE clause cannot be applied on the encrypted records. They all are sent to the client site, decrypted, and then the condition is applied. The graph also shows that Frequency-of-Use-Based partition and Bisection-Tree-Based partition are more efficient than the others, and the Histogram-Based partition is the least efficient.

Table 4.4 Retrieved records in each kind of the partitioning (size=100,000 Records).

Data requesting %	Unencrypted Table	Encrypted Table without data Category	Frequency of use Based	Space Based	Mondrian or Bisection tree Based	Histogram Based
10%	10131	100000	10131	13857	11145	20194
20%	19954	100000	19954	21690	20161	26278
30%	29997	100000	29997	30538	30696	33616
40%	39907	100000	39907	46013	41080	59637
50%	49940	100000	49940	58613	50733	53592
60%	59773	100000	59773	65701	61448	72956
70%	69904	100000	69904	74568	71776	86493

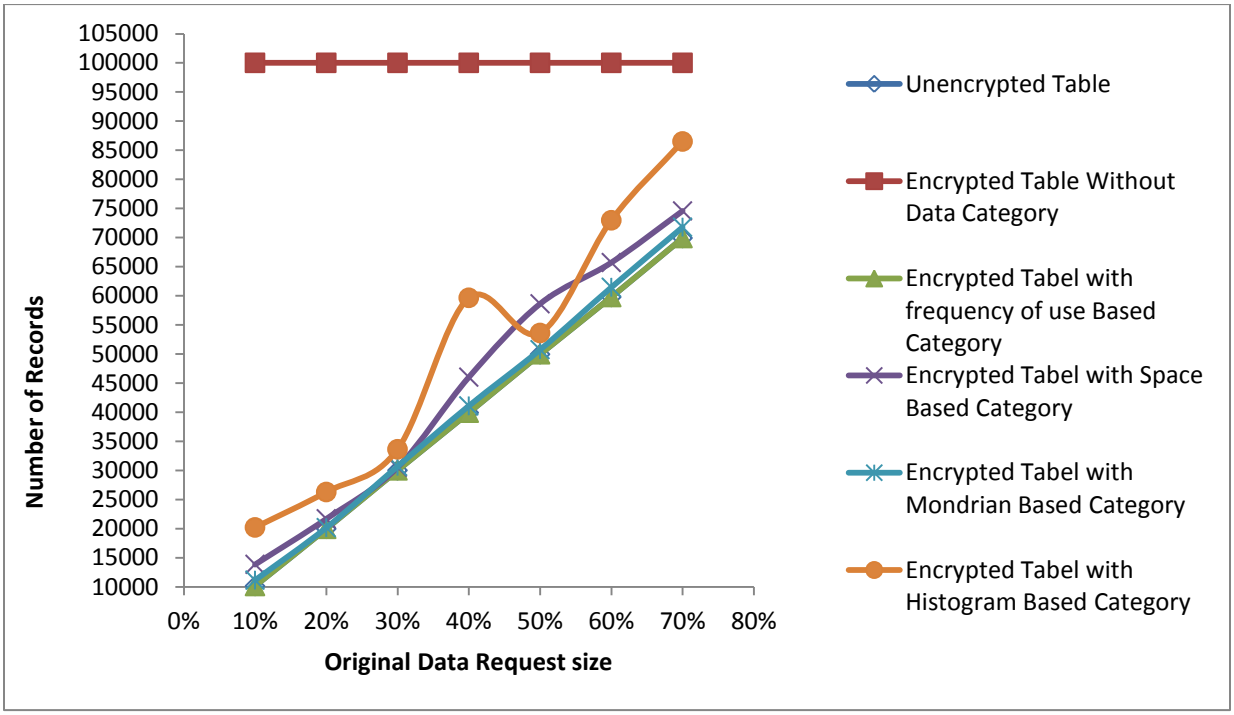


Figure 4.2 Records Retrieved from Database Table having 100,000 Records with record size 68 Bytes.

4.5.2 Data Retrieved from Table Having 100,000 Records with Record Size 168 bytes

In this section, the process to evaluate and compare the different partitioning techniques will be explained using the employee table having 100,000 records with record size 168 bytes. Table 4.5 shows the structure of the employee table. Table 4.6 shows different percentage numbers that

were used to define the number of records retrieved from the employee tables on the server, and also the WHERE clauses which were used. To do the evaluation, in addition to the original table, five different encrypted versions of the employee table have been created to evaluate each type of data partitioning. In addition to this structure, two indexed attributes are added to test the four types of data partitioning.

Table 4.5 Structure of the original Employee table (Record size=168 Bytes)

Attribute Name	Attribute Type	Attribute Size (Byte)
id	int	4
Name	Varchar	25
Sal	Int	4
NetSal	Int	4
Address	Varchar	30
Sex	char	1
Note	Varchar	100
Total=		= 168 Byte

Table 4.6 Percentage of data requested by WHERE clauses

	Data requesting %	Where Clause
1	10%	Where (Sal>=2000 and Sal<3000)
2	20%	Where (Sal>=0 and Sal<2000)
3	30%	Where ((Sal>=0 and Sal<2000) or (Sal>=2000 and Sal<3000))
4	40%	Where ((Sal>=4500 and Sal<6500) or (Sal>=8000 and Sal<10000))
5	50%	Where ((Sal>=3000 and Sal<6500) or (Sal>=6500 and Sal<8000))
6	55%	Where ((Sal>=1000 and Sal<3000) or (Sal>=3000 and Sal<6500))
7	70%	Where ((Sal>=3000 and Sal<6500) or (Sal>=6500 and Sal<10000))

4.5.2.1 Running Query on Table Having 100,000 Records with Record Size 168 bytes

The WHERE clauses as shown in Table 4.6 have been applied on the employee table with different structures depending on the type of partition to get the evaluation. Table 4.7 shows the running time for each type of partitioning in milliseconds. Figure 4.3 shows the percentage of the retrieved records from the employee tables and the consumption time in milliseconds, which is

used to retrieve the corresponding records from each type. As can be seen from the graph, there are different retrieval time values for the same percentage of records. The encrypted table without data category took the maximum time. The graph also shows that the amount of retrieval time increased steadily for all types of partitions. As the graph depicts, the retrieval time is augmented in the encrypted table with Space-Based partition from 30% to 50%. In addition, the graph shows the encrypted table with Frequency-of-Use-Based partition and the encrypted table with Bisection-Tree-Based partition are more efficient than the others. When the amount of records retrieved are more than 55%, the consumption time for all kinds are almost the same.

Table 4.7 Running time for each type of partitioning in milliseconds (size=100,000 records with record size 168 bytes)

Data requesting %	Unencrypted Table *	Encrypted Table without data Category *	Frequency of use Based *	Space Based *	Mondrian or Bisection tree Based *	Histogram Based *
10%	981	1180	985	1093	1000	1021
20%	1058	1256	1070	1094	1069	1078
30%	1116	1312	1151	1188	1145	1153
40%	1153	1370	1225	1354	1226	1258
50%	1214	1554	1291	1384	1286	1298
55%	1243	1573	1335	1389	1338	1365
70%	1391	1633	1544	1571	1565	1570

(*=Running time in Millisecond)

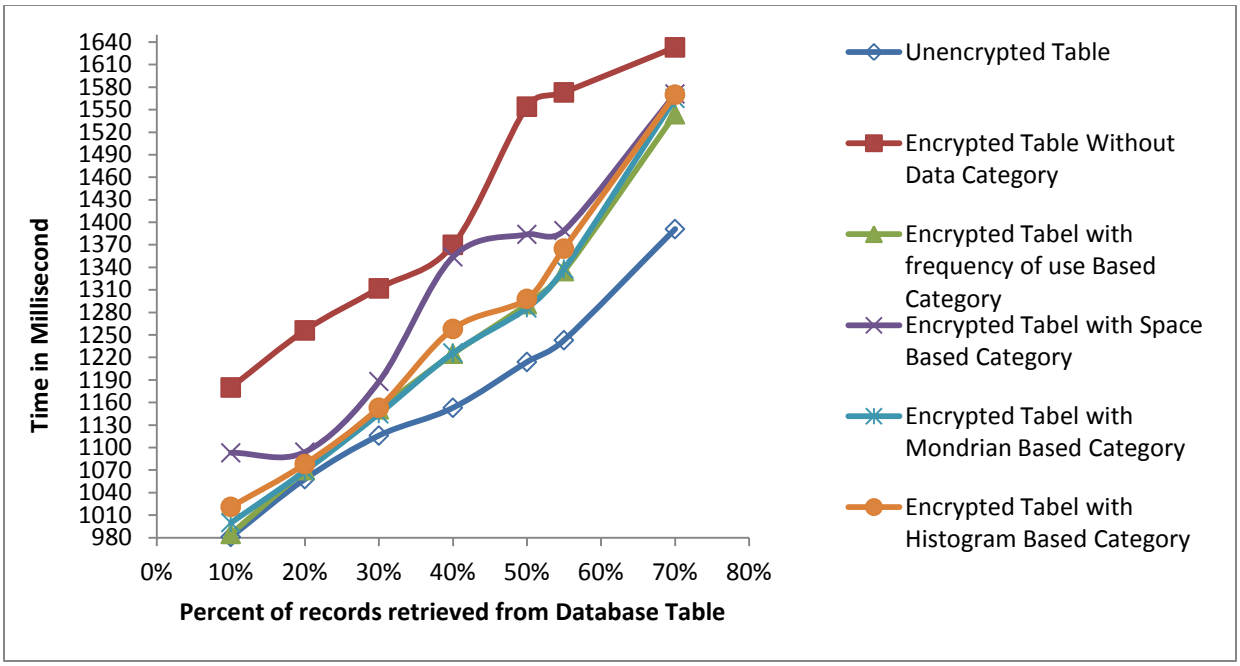


Figure 4.3 Records retrieval time from database table having 100,000 records with record size 168 Bytes

4.5.2.2 Records Retrieved from Table Having 100,000 Records with Record size 168 bytes

By applying the WHERE clause as shown in Table 4.6, the methods are tested using different percentages of retrieved records. Table 4.8 shows the number of records retrieved under various cases. Figure 4.4 shows the results in graph form. The graph shows that the encrypted table without data category retrieves all records, which is 100,000, because the WHERE clause cannot be applied on the encrypted records. They all are sent to the client site, decrypted, and then the condition is applied. The graph also shows the encrypted table with Frequency-of-Use-Based partition and the encrypted table with Bisection-Tree-Based partition are more efficient than the others and are equally efficient as the unencrypted table. The encrypted table with Space-Based partition and the Histogram-Based partition are the least efficient.

Table 4.8 Retrieved records in each kind of partitioning (size=100,000 Records).

Data requesting %	Unencrypted Table	Encrypted Table without data Category	Frequency of use Based	Space Based	Mondrian or Bisection tree Based	Histogram Based
10%	10003	100000	10003	11999	10925	20064
20%	19954	100000	19954	20000	20264	19960
30%	29957	100000	29957	29999	30423	33317
40%	40149	100000	40149	51016	41571	53493
50%	50038	100000	50038	56977	50834	53338
55%	54916	100000	54916	59991	56222	60019
70%	70043	100000	70043	75001	70359	73308

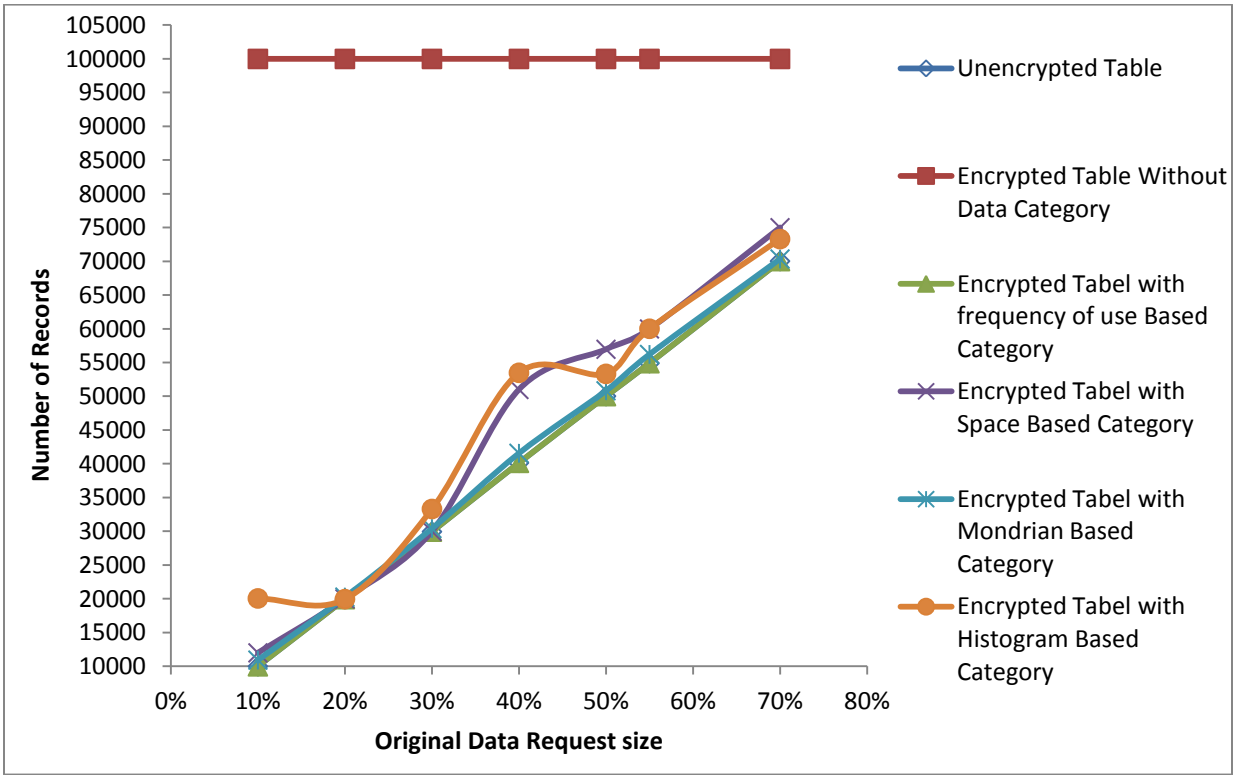


Figure 4.4 Records Retrieved from Database Table having 100,000 Records with record size 168 Bytes

4.5.3 Data Retrieved from Table Having 100,000 Records with Record Size 468 bytes

In this part, the procedure to evaluate and compare the different partitioning techniques will be described using the employee table having 100,000 records with record size 468 bytes. Table 4.9

shows the structure of the employee table. Table 4.10 shows different percentage numbers that were used to define the number of records retrieved from the employee table on the server, and it shows the WHERE commands which were used. To do the evaluation, in addition to the original table, five different encrypted formats from the employee table have been created to evaluate each type of data partitioning. Also, two indexed attributes are added to this table to evaluate all data partitioning types.

Table 4.9 Structure of the original Employee table (Record size=468 Bytes)

Attribute Name	Attribute Type	Attribute Size (Byte)
id	int	4
Name	Varchar	25
Sal	Int	4
NetSal	Int	4
Address	Varchar	30
Sex	char	1
Note	Varchar	100
Description	Varchar	300
Total=		= 468 Byte

Table 4.10 Percentage of data requested by WHERE clauses

	Percentage of data requesting	Where Clause
1	10%	Where (Sal>=2000 and Sal<3000)
2	20%	Where (Sal>=4500 and Sal<6500)
3	30%	Where ((Sal>=0 and Sal<2000) or (Sal>=2000 and Sal<3000))
4	40%	Where ((Sal>=4500 and Sal<6500) or (Sal>=8000 and Sal<10000))
5	50%	Where ((Sal>=3000 and Sal<6500) or (Sal>=6500 and Sal<8000))
6	55%	Where ((Sal>=1000 and Sal<3000) or (Sal>=3000 and Sal<6500))
7	70%	Where ((Sal>=3000 and Sal<6500) or (Sal>=6500 and Sal<10000))

4.5.3.1 Running Query on Table Having 100,000 Records with Record Size 468 bytes

To get the evaluation the WHERE clauses as explained in Table 4.10 have been applied on the employee tables with different structures, depending on the type of partition. Table 4.11 shows the running time for each type of partitioning in milliseconds. Figure 4.5 shows the percentage of the retrieved records from the employee tables and the consumption time in milliseconds, which is used to retrieve the requested records for each type. It can be clearly seen that the unencrypted table took the shortest times, and the encrypted table without data category took the longest times. The graph demonstrates how retrieval times increased regularly for all partition types. In addition, the graph illustrates that the encrypted table with Frequency-of-Use-Based category and the encrypted table with Bisection-Tree-Based category are more efficient than the others.

Table 4.11 Running time for each type partitioning in milliseconds (size=100,000 records with record size 468 bytes)

Data requesting %	Unencrypted Table *	Encrypted Table without data Category *	Frequency of use Based *	Space Based *	Mondrian or Bisection tree Based *	Histogram Based *
10%	1065	1332	1077	1129	1089	1124
20%	1169	1655	1186	1309	1189	1220
30%	1314	1704	1373	1397	1367	1403
40%	1413	1792	1477	1645	1474	1540
50%	1496	1912	1584	1681	1593	1613
55%	1618	1957	1716	1778	1723	1750
70%	1743	2083	1940	1967	1927	1923

(*=Running time in Millisecond)

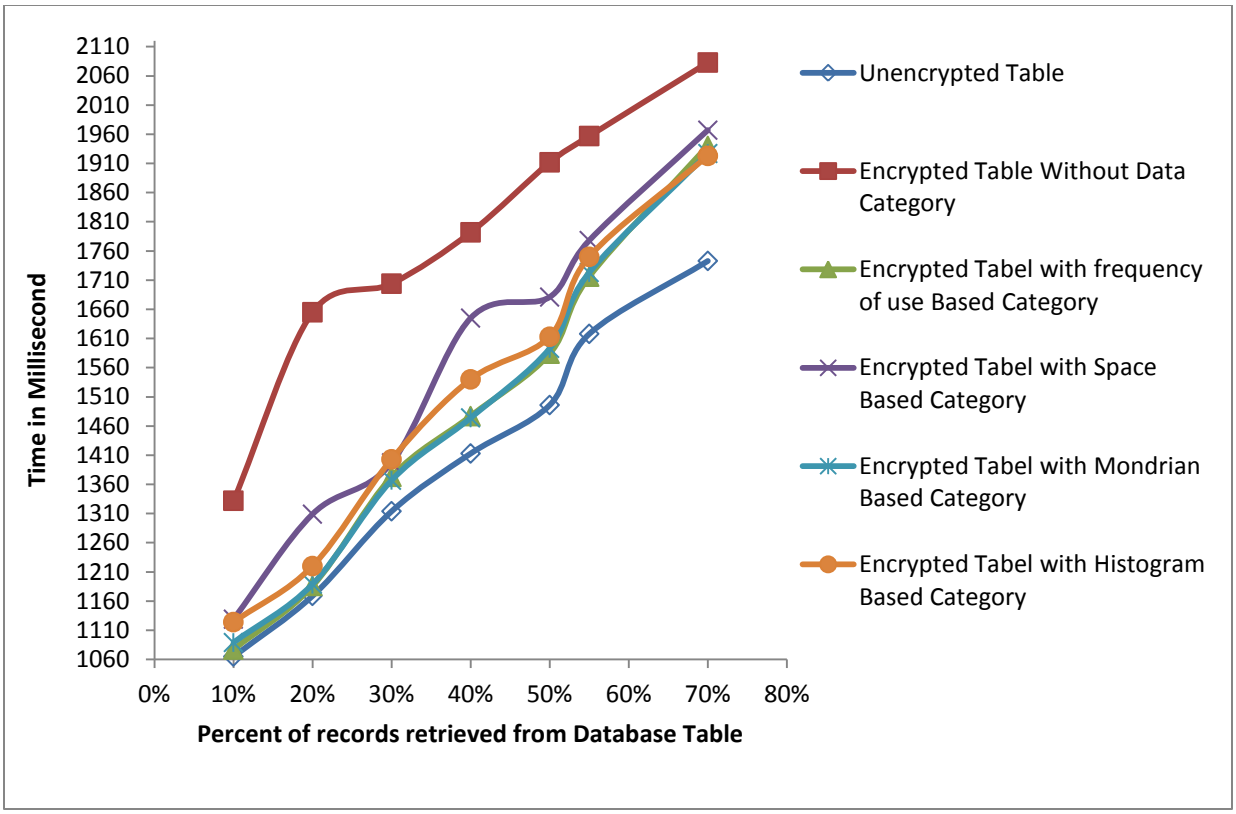


Figure 4.5 Records Retrieval Time from Database Table having 100,000 Records with record size 468 Bytes

4.5.3.2 Records Retrieved from Table Having 100,000 Records with Record Size 468 bytes

By applying the WHERE clauses as shown in Table 4.10, the methods are tested using different percentages of retrieved records. Table 4.12 shows the number of records retrieved under various cases. Figure 4.6 shows the results in graph form. The graph shows that the encrypted table without data category retrieves all records, which is 100,000, because these records must be fetched each time the WHERE command is applied to decrypt all the records on a client site. The graph also shows the encrypted table with Frequency-of-Use- Based category and the encrypted table with Bisection-Tree-Based category are more efficient than the others, and are equal to the unencrypted table. The graph explains how retrieval times increased regularly for all types of partitions.

Table 4.12 Retrieved records in each kind of partitioning (size=100,000 Records)

Data requesting %	Unencrypted Table	Encrypted Table without data Category	Frequency of use Based	Space Based	Mondrian or Bisection tree Based	Histogram Based
10%	10003	100000	10003	11999	10925	20064
20%	20144	100000	20144	29994	21240	26713
30%	29957	100000	29957	29999	30423	33317
40%	40149	100000	40149	52016	41571	53493
50%	50038	100000	50038	56977	50834	53338
55%	54916	100000	54916	59991	56222	60019
70%	70043	100000	70043	75001	70359	73308

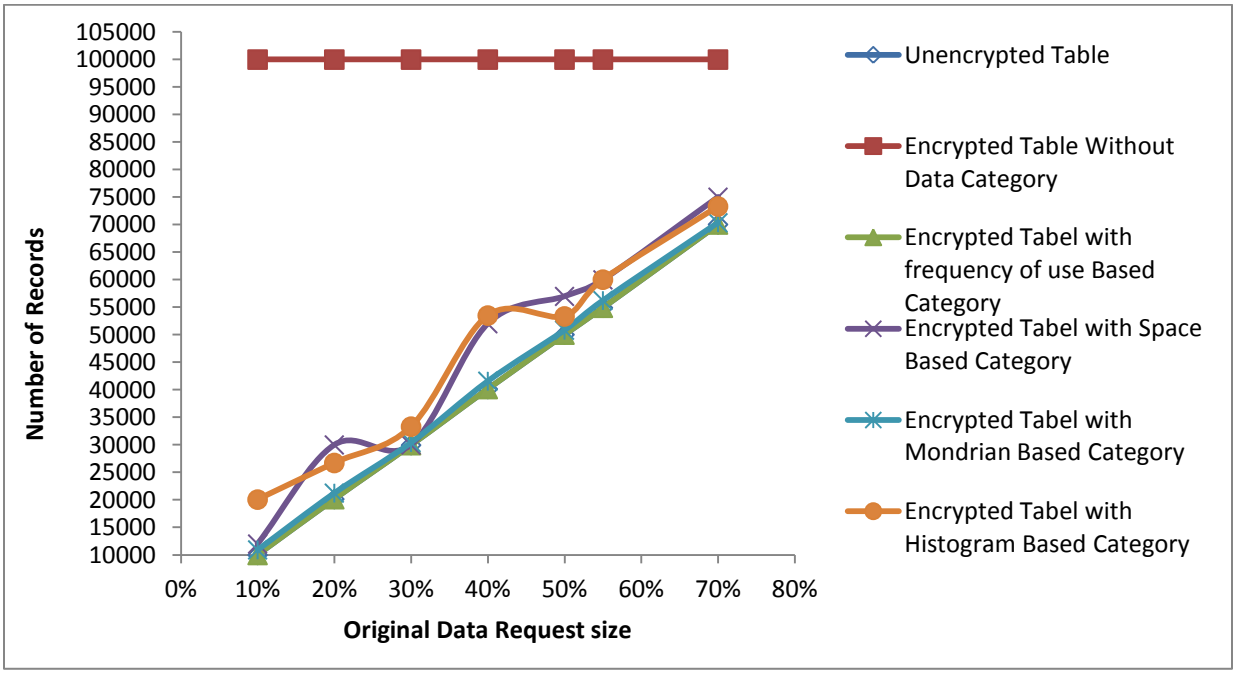


Figure 4.6 Records Retrieved from Database Table having 100,000 Records with record size 468 Bytes

4.5.4 Data Retrieved from Table Having 500,000 Records with Record size 68 bytes

In this section, the process to evaluate and compare the different partitioning techniques has been changed using new parameters. The employee table size is increased to have 500,000 records with record size 68 bytes. Table 4.1 shows the structure of the employee table. Table 4.2 shows different percentages have been used to define the number of records retrieved from the tables,

and it shows the WHERE commands which were used. Four different encrypted formats with indexing attributes from the employee table were used to evaluate each type of data partitioning.

4.5.4.1 Running Query on Table Having 500,000 Records with Record Size 68 bytes

The WHERE clauses as shown in Table 4.2 have been applied on the employee tables with different structures depending on the type of partition used to get the evaluation. Table 4.13 shows the retrieval time for each type of partitioning in milliseconds. Figure 4.7 shows the percentage of the retrieved records and the consumption time in milliseconds. As can be seen from the graph, there are different values in retrieval time for the same percentage. The unencrypted table has the minimum retrieval time, and for the four kinds of partitions the encrypted table with Frequency-of-Use-Based category and the encrypted table with Bisection-Tree-Based category are more efficient. The graph also shows the time increased steadily for all types of partitions. The encrypted table with Space-Based category has the smallest amount of data retrieval time when the retrieved data is less than 55%, but it takes a large amount when the retrieved data is more than 55%. In addition, the graph displays the encrypted table without data category has the greatest value when the retrieved data less than 55%, but it has a value equal to the encrypted table with Frequency-of-Use-Based category and the encrypted table with Bisection-Tree-Based category, when the retrieved data is greater than 55%.

Table 4.13 Running time for each type of partitioning in milliseconds (size=500,000 records with record size 68 bytes)

Data requesting %	Unencrypted Table *	Encrypted Table without data Category *	Frequency use Based *	Space Based *	Mondrian or Bisection tree Based *	Histogram Based *
10%	1457	2524	1469	1726	1525	1612
20%	1606	2645	1675	1853	1709	1768
30%	1921	2736	2045	2158	2063	2113
40%	2126	2848	2336	2644	2374	2647
50%	2328	3003	2639	2827	2669	2661
60%	2625	3021	2969	3265	2991	3103

70%	2796	3161	3118	3457	3198	3320

(*=Running time in Millisecond)

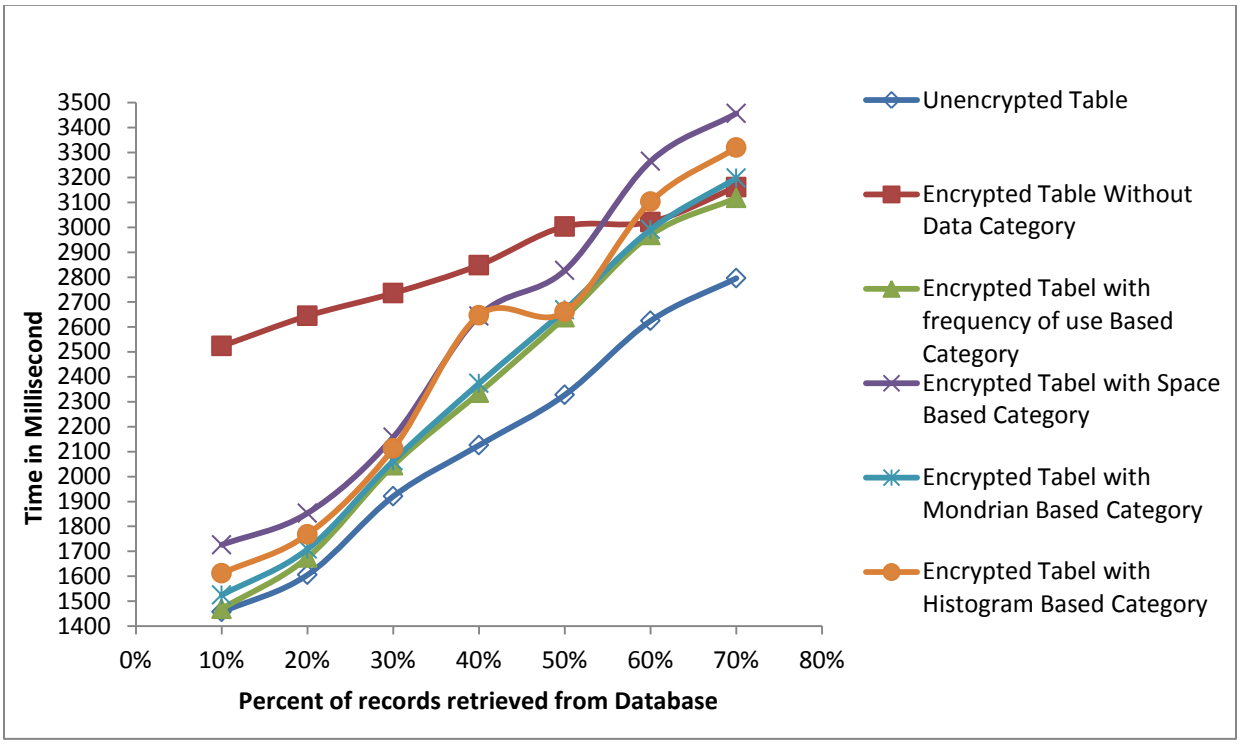


Figure 4.7 Records Retrieval Time from Database Table having 500,000 Records with record size 68 Bytes

4.5.4.2 Records Retrieved from Table Having 500,000 Records with Record Size 68 bytes

The methods were tested using different percentages of retrieved records by applying the WHERE clauses as explained in Table 4.2. Table 4.14 shows the number of records retrieved for each type of partitioning. Figure 4.8 shows the number of records retrieved from the various tables. As the graph depicts, from the encrypted table without any category all 500,000 records were retrieved for the same reason as explained before. The graph also shows that the encrypted table with Histogram-Based category was less efficient than the other types of the partitions.

Table 4.14 Retrieved records in each kind of partitioning (size=500,000 Records)

Data requesting %	Unencrypted Table	Encrypted Table without data Category	Frequency of use Based	Space Based	Mondrian or Bisection tree Based	Histogram Based
10%	50655	500000	50655	69285	55725	100970
20%	99770	500000	99770	108450	100805	131390
30%	149985	500000	149985	152690	153480	168080
40%	199535	500000	199535	230065	205400	298185
50%	249700	500000	249700	263785	253665	267960
60%	298865	500000	298865	328505	307240	364780
70%	349520	500000	349520	372840	358880	432465

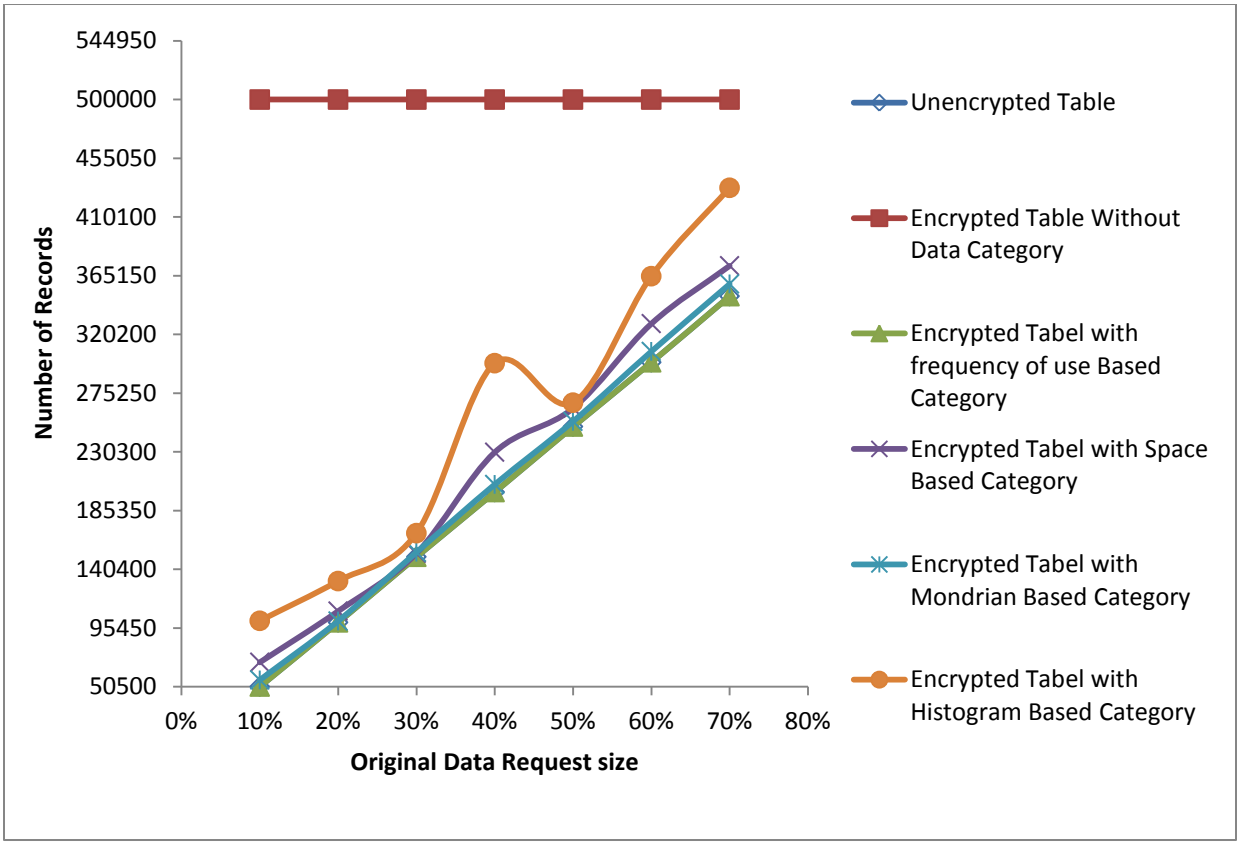


Figure 4.8 Records Retrieved from Database Table having 500,000 Records with record size 68 Bytes

4.5.5 Data Retrieved from Table Having 500,000 Records with Record Size 168 bytes

In this section, the size of the record is changed to 168 bytes to evaluate and compare the different partitioning techniques using the employee table having 500,000 records. Table 4.5 shows the structure of the employee table. Table 4.6 shows the different percentages have been used to define the number of records retrieved from the employee tables, and shows the WHERE commands which were used. Four different encrypted formats with indexing attributes from the employee table were used to evaluate each type of data partitioning.

4.5.5.1 Running Query on Table Having 500,000 Records with Record Size 168 bytes

The WHERE clauses which were explained in Table 4.6 have been applied on the employee tables with different structures depending on the type of partition used to get the evaluation. Table 4.15 shows the retrieval time for each type of partitioning in milliseconds. Figure 4.9 shows the percentage of the retrieved records and the consumption time in milliseconds. The graph shows that the unencrypted table has the smallest amount of data retrieval time, and the encrypted table without data category has the largest amount. The graph also shows the time increased steadily for all types of partitions. For the four kinds of partitions, the graph shows the encrypted table with Frequency-of-Use-Based category and the encrypted table with Bisection-Tree-Based category are more efficient than the others. In addition, the encrypted table with Space-Based category is less efficient, especially when the retrieved data is more than 30%.

Table 4.15 Running time for each type of partitioning in milliseconds (size=500,000 records with record size 168 bytes)

Data requesting %	Unencrypted Table *	Encrypted Table without data Category *	Frequency of use Based *	Space Based *	Mondrian or Bisection tree Based *	Histogram Based *
10%	1602	2932	1665	1887	1743	1987
20%	1939	3222	2170	2288	2202	2218
30%	2240	3446	2689	2817	2412	2748
40%	2646	3921	3073	3781	3161	3373
50%	2941	4127	3597	3845	3617	3681
55%	3100	4365	3770	4008	3752	3851
70%	3445	4533	4345	4543	4332	4446

(*=Running time in Millisecond)

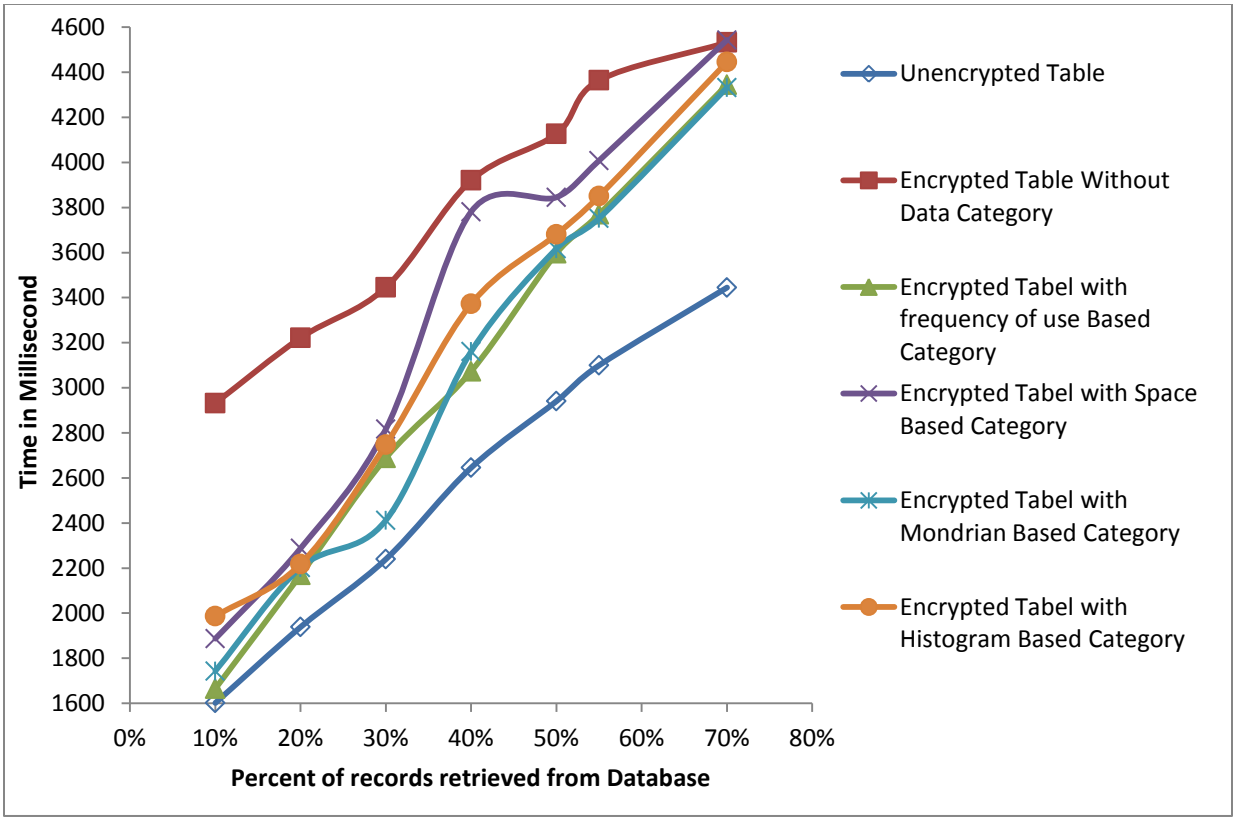


Figure 4.9 Records Retrieval Time from Database Table having 500,000 Records with record size 168 Bytes

4.5.5.2 Records Retrieved from Table Having 500,000 Records with Record Size 168 bytes

Table 4.16 shows the number of records retrieved for each type of partitioning. Figure 4.10 shows the number of records retrieved from the various tables. As the graph depicts, from the encrypted table without any category all 500,000 records were retrieved for the same reason as explained before and that because this number of records must be fetched each time the WHERE clause is applied to decrypt all the records at the client site. The graph also displays that the encrypted table with Histogram-Based category and encrypted table with Space-Based category were less efficient than the other types of partitions. On the other hand, the encrypted table with Frequency-of-Use-Based category and encrypted table with Bisection-Tree-Based category are more efficient than the rest.

Table 4.16 Retrieved records in each kind of partitioning (size=500,000 Records)

Data requesting %	Unencrypted Table	Encrypted Table without data Category	Frequency of use Based	Space Based	Mondrian or Bisection tree Based	Histogram Based
10%	50015	500000	50015	59995	54625	100320
20%	99770	500000	99770	100000	101320	99800
30%	149785	500000	149785	149995	152115	166585
40%	200745	500000	200745	255080	207855	267465
50%	250190	500000	250190	284885	254170	266690
55%	274580	500000	274580	299955	281110	300095
70%	350215	500000	350215	375005	351795	366540

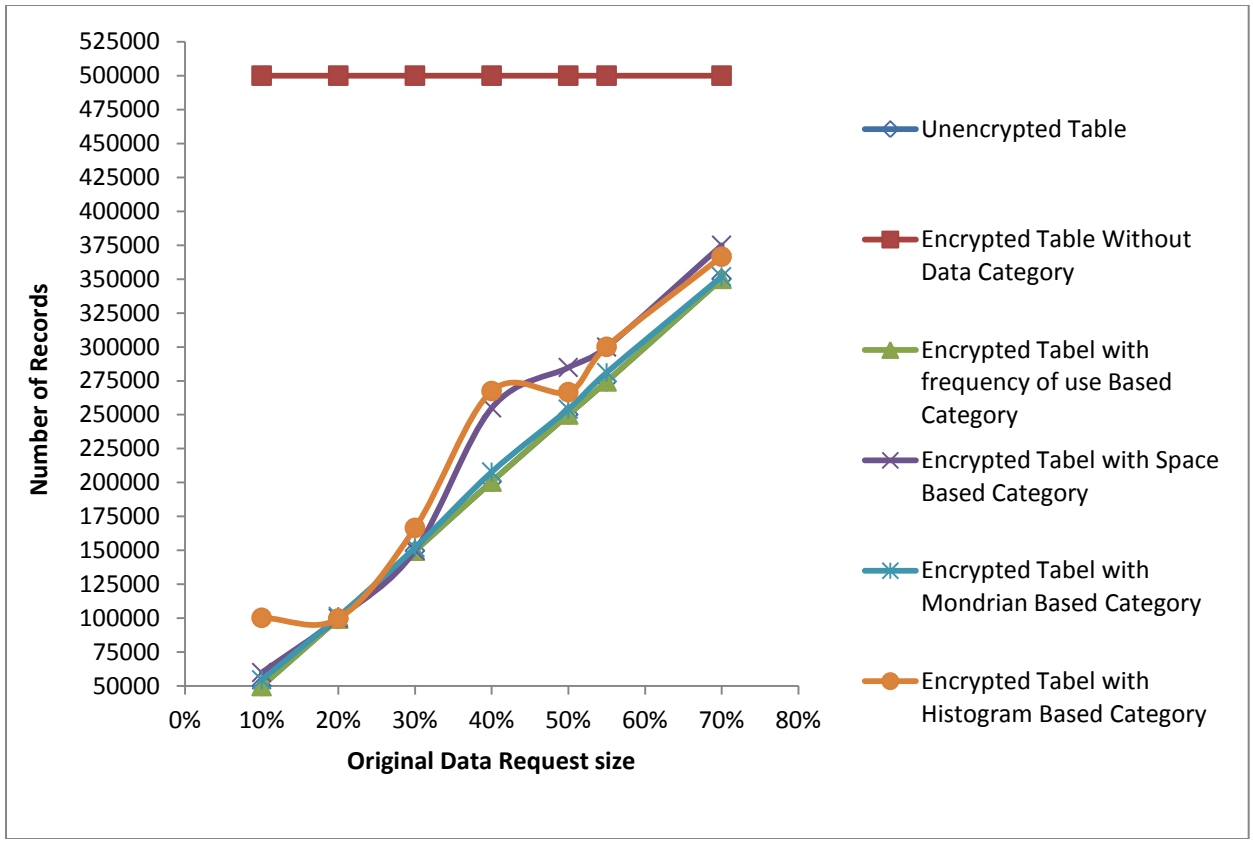


Figure 4.10 Records Retrieved from Database Table having 500,000 Records with record size 168 Bytes

4.5.6 Data Retrieved from Table Having 500,000 Records with Record Size 468 bytes

Next, the size of the record is increased to 468 bytes to evaluate and compare the different partitioning techniques using the employee table having 500,000 records. Table 4.9 shows the structure of the employee table. Table 4.10 shows different percentages have been used to define the number of records retrieved from the tables. Four different encrypted formats with indexing attributes from the employee table were used to evaluate each type of data partitioning.

4.5.6.1 Running Query on Table Having 500,000 Records with Record Size 468 bytes

The WHERE clauses as explained in table 4.10 have been applied on the employee tables with different structures depending on the type of partition used to calculate the evaluation. Table 4.17

shows the retrieval time for each type of partitioning in milliseconds. Figure 4.11 shows the percentage of the retrieved records and the consumption time in milliseconds. The graph shows that the unencrypted table has the shortest amount of data retrieval time, especially when the retrieved data grows above 30%, and the encrypted table without data category has the maximum value and close to the four types of partitions on 70% data retrieved. The graph also shows the retrieval times increased uniformly for all partition types. For the four kinds of partitions, the graph shows the encrypted table with Frequency-of-Use-Based category and the encrypted table with Bisection-Tree-Based category as the most efficient kinds of partitions, and as time increases they swap efficiency. In addition, the encrypted table with Space-Based category is less efficient, especially when the retrieved data is more than 20%.

Table 4.17 Running time for each type of partitioning in milliseconds (size=500,000 records with record size 468 bytes)

Data requesting %	Unencrypted Table *	Encrypted Table without data Category *	Frequency of use Based *	Space Based *	Mondrian or Bisection tree Based *	Histogram Based *
10%	2431	5693	2552	2830	2572	2919
20%	2989	6182	3173	3900	3291	3772
30%	3783	6421	4054	4415	4000	3824
40%	4031	7146	4341	5279	4602	5074
50%	4683	7393	5228	5825	5351	5311
55%	5147	7791	6067	7102	5842	6642
70%	6492	8047	7530	7831	7609	7605

(*=Running time in Millisecond)

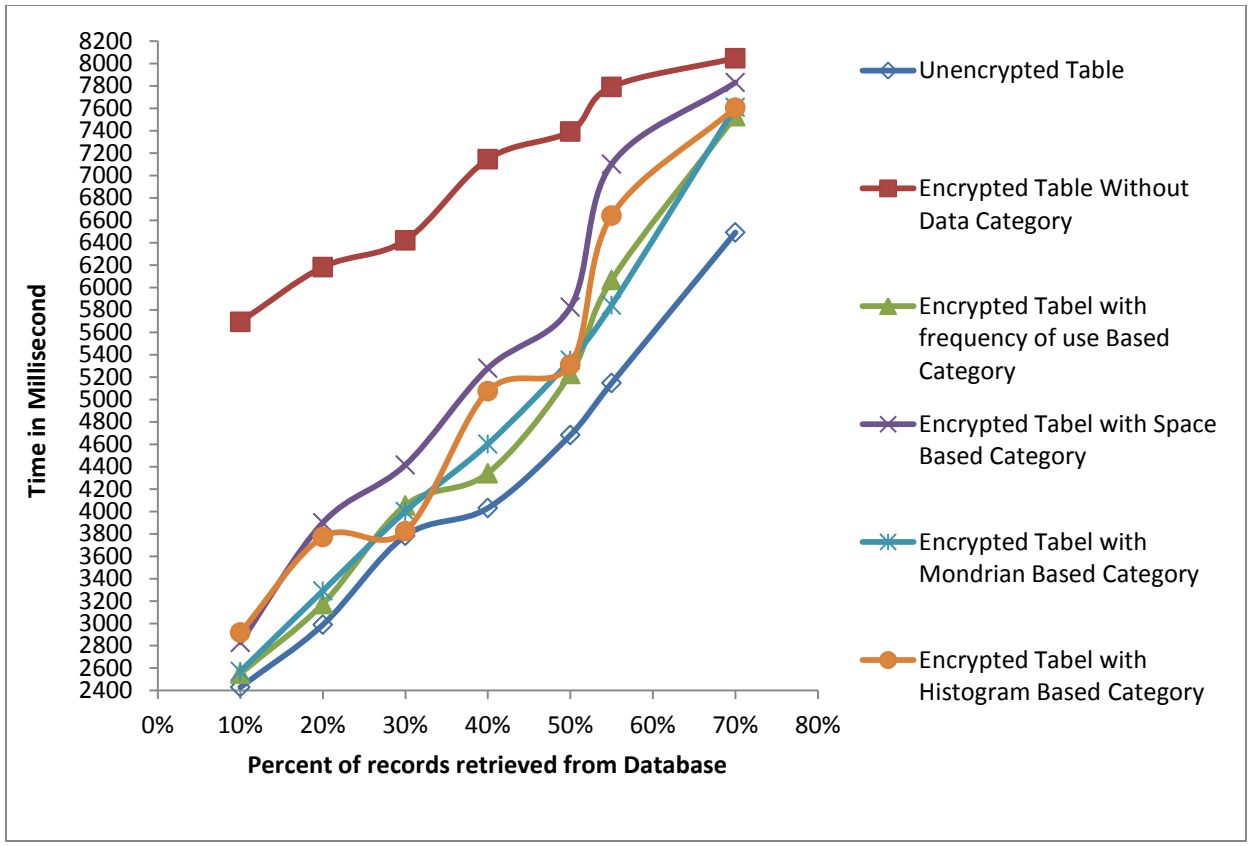


Figure 4.11 Records Retrieval Time from Database Table having 500,000 Records with record size 468 Bytes

4.5.6.2 Records Retrieved from Table Having 500,000 Records with Record Size 468 bytes

Table 4.18 shows the number of records retrieved for each type of partitioning. Figure 4.12 shows the number of records retrieved from the various tables. The graph shows that the encrypted table without any category retrieved all 500,000 records for the same reason as explained before. The graph also displays that the encrypted table with Histogram-Based category and encrypted table with Space-Based category were less efficient than the other types of partitions, and they switch inefficiency between them. On the other hand, the encrypted table with Frequency-of-Use-Based category and encrypted table with Bisection-Tree-Based category are more efficient than the rest.

Table 4.18 Retrieved records in each kind of partitioning (size=500,000 Records)

Data requesting %	Unencrypted Table	Encrypted Table without data Category	Frequency of use Based	Space Based	Mondrian or Bisection tree Based	Histogram Based
10%	50015	500000	50015	59995	54625	100320
20%	100720	500000	100720	144970	106200	133565
30%	149785	500000	149785	149995	152115	166585
40%	200745	500000	200745	255080	207855	267465
50%	250190	500000	250190	284885	254170	266690
55%	274580	500000	274580	299955	281110	300095
70%	350215	500000	350215	375005	351795	366540

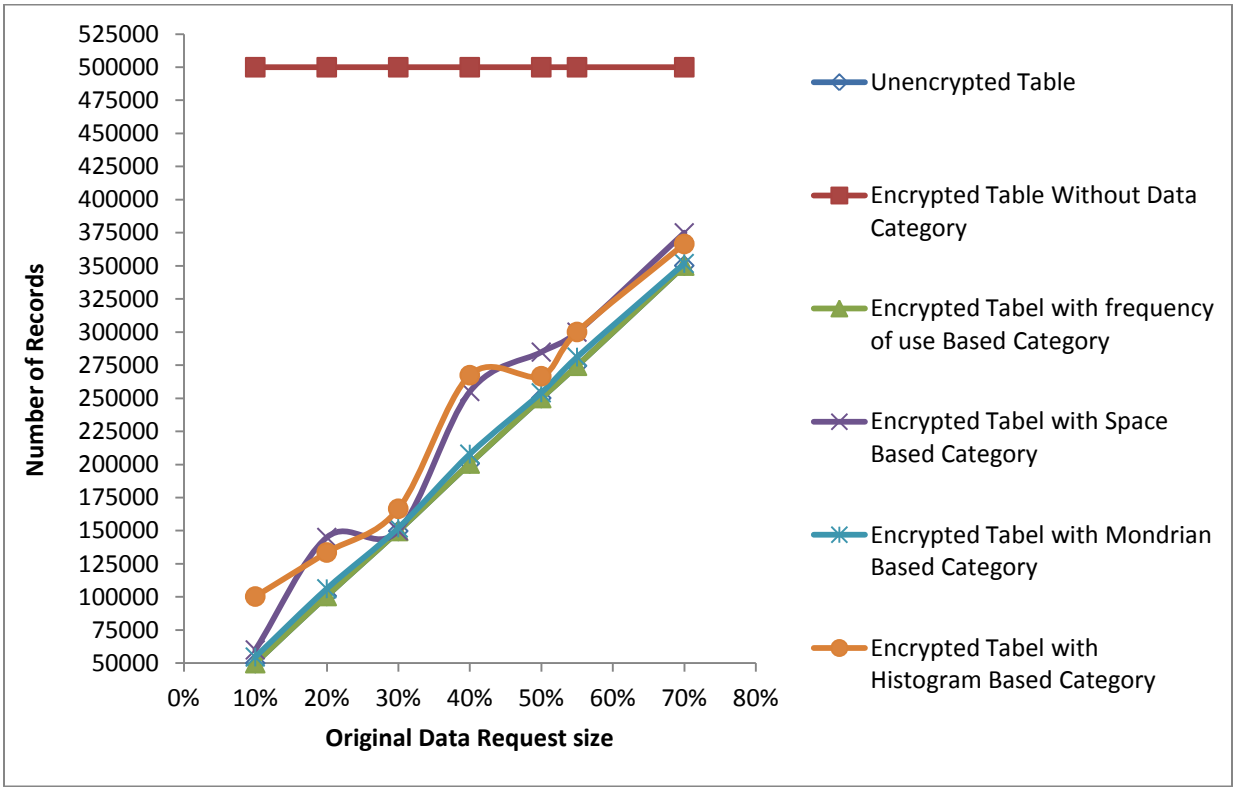


Figure 4.12 Records Retrieved from Database Table having 500,000 Records with record size 468 Bytes

4.6 Factors affecting Cloud Performance

The database system performance is measured under different parameter settings, like range of the CPU shares, memory size, number of replicas, and arrival rate. The following list will affect the performance of any application.

- CPU: In [69], researchers explain that if the CPU share increases, the performance will be improved in a close to linear fashion. Therefore, the four techniques, which are suggested in this research, can improve the running time. The four techniques can use more than one processor to work on each partition category and collect the final result for a client. The results' running time, which are explained before for each type of partitioning, are run on one processor, so adding more than one processor will improve the performance.
- Memory: Memory is a crucial component for any application. Researchers in [69] show that increasing memory size will help the performance, and adding more memory does not help when memory is already larger than a certain threshold.
- Replica: Researchers in [69] demonstrate that increasing the number of replicas will improve the performance and narrow the chance of queuing delay.
- Network bandwidth: Authors in [100] illustrate that when the bandwidth increases, this increase will improve the performance.
- Disk I/O (sequential and random): Researchers in [73] say disk I/O will affect the performance, because many Cloud applications need instances to store intermediate or transitional results and do some operations on local disks if some operations cannot be processed in main memory.

In addition, authors in [100] illustrate some other factors that can affect the performance like: security, recovery, service level agreements, buffer capacity, disk capacity, fault tolerance, availability, number of users, location of data center, usability, scalability, and workload.

4.7 Combining Frequency-of-Use-Based and Bisection-Tree-Based

From previous results it was clear that the two kinds Frequency-of-Use-Based and Bisection-Tree-Based partitions are the most efficient kinds of partitions. The Frequency-of-Use-Based

method depends on the frequency of the queries as found in the log file. This means all queries are determined previously, and any change to any query may affect the performance. For example, Table 4.19, Figure 4.13, Table 4.20, and Figure 4.14 show the running time and number of retrived records for the Frequency-of-Use-Based and Bisection-Tree-Based methods while considering normal queries, which is dependent on the main techniques that are clarified in chapter 3. They also show what will happen if a different range of Frequency-of-Use-Based partition is used, and how it will affect the performance when the queries are modified. To solve this issue and improve performance, a combination of the Frequency-of-Use Based and Bisection-Tree-Based methods can be used to partition the data. Figure 4.13 and Figure 4.14 show a slight difference between the Bisection-Tree-Based method and the combination of the Frequency-of-Use-Based and the Bisection-Tree-Based approache. This minor difference is because the percentage of data requested is low; i.e., only 10%. If that number increases to more than 50%, the difference will be significant.

Table 4.19 Running time of combining Frequency-of-Use-Based and Bisection-Tree-Based & comparing them without Combining.

Data requesting %	Frequency of use Based without changing query *	Different Range of Frequency use Based *	Mondrian or Bisection tree Based *	Combination of Frequency of use Based and Bisection tree Based *
10%	985	1065	1017	1005

(*=Running time in Millisecond)

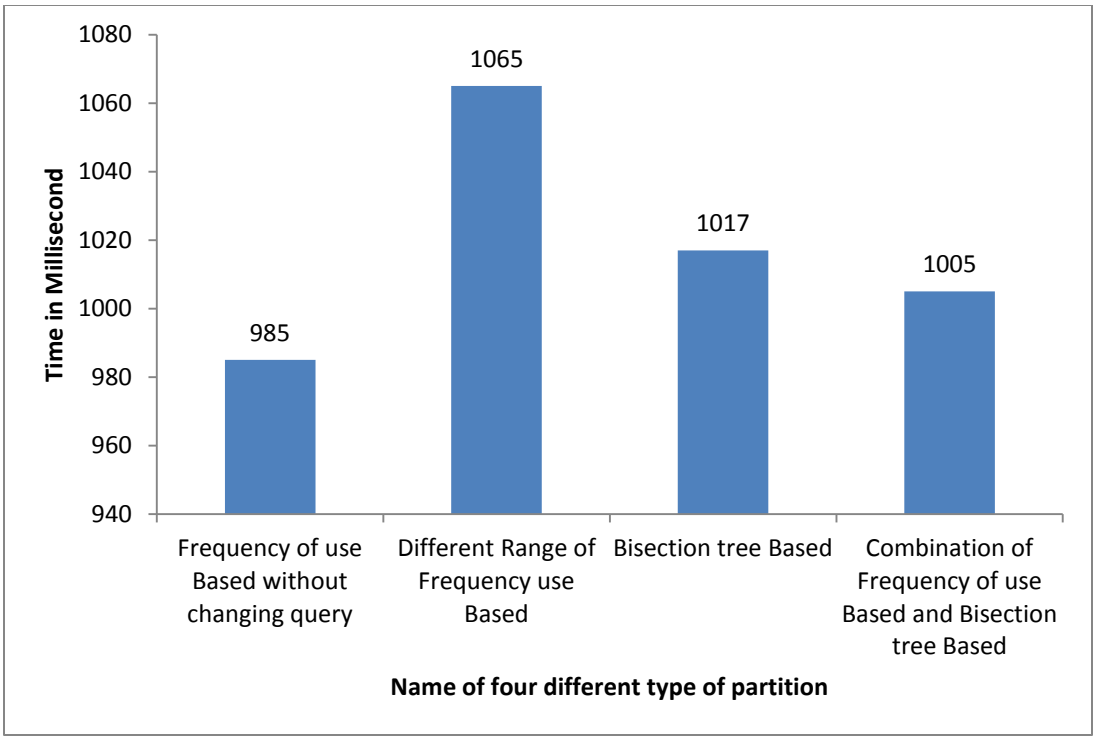


Figure 4.13 Running time for records retrieved from database table having 100,000 total Records in DB to show the running time of combining Frequency-of-Use-Based and Bisection-Tree-Based & comparing them without combining.

Table 4.20 Retrieved records of combine Frequency-of-Use-Based and Bisection-Tree-Based & comparing them without combining.

Data requesting %	Frequency of use Based without changing query	Different Range of Frequency use Based	Mondrian or Bisection tree Based	Combination of Frequency of use Based and Bisection tree Based
10%	10003	35182	10886	10605

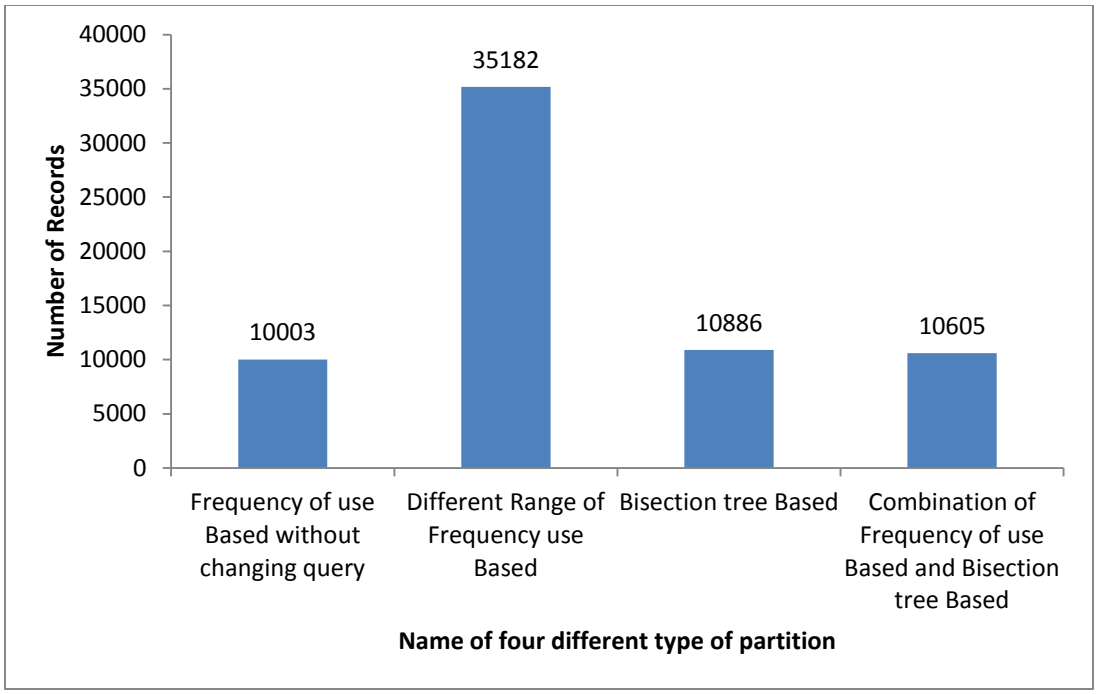


Figure 4.14 Retrieved records of combining Frequency-of-Use-Based and Bisection-Tree-Based & comparing them without combining from database table having 100,000 Records.

4.7.1 Example of Combined Frequency-of-Use-Based and Bisection-Tree-Based Method

This example clarifies how to use Frequency-of-Use-Based and Bisection-Tree-Based methods together. The technique begins by mining the log file and completing the same steps used in Frequency-of-Use-Based partition. For example, Table 4.21 is created after applying the Frequency-of-Use-Based method, and the number of records in each range are calculated. Before the category numbers are given to table 4.21, a bucket size is determind to apply Bisection-Tree-Based method. In this example Bucket size is equal to 3000. The following steps will explian how to apply Bisection-Tree-Based method in Table 4.21, particularly the partitions [4500-6500] and [6500-8000]. The first step takes the partitions [4500-6500], and if the number of records in it is more than the bucket size, it calculates the median, which is equal to (4500+6500)/2=5500. Thus the partition category [4500-6500] is split into [4500-5500] and [5500-6500]. The process then checks the partition category [4500-5500], and if the number of records in that range is less

than the bucket size, the process takes the next partition category and completes the same processing. If the number of records in that range is more than the bucket size, the process calculates the median and divides the range into two different categories, and repeats the process. Figure 4.15 illustrates this idea. Figure 4.15 also shows how 10%, which is equal to the range from 6000 to 7000, have been calculated for the query in the Table 4.20. Figure 4.15 shows how to get the values from 6000 to 7000, and it is explined the partitions [6000-6250], [6250-6500], [6500-6688], [6688-6875], [6875-7063] which equal to 2508+2475+1899+1866+1857=10605 must be fetched. Therefore, this technique will improve the performance as shown in Table 4.19 and Table 4.20.

Table 4.21 Partition category and their frequency

Partitioning Category	Frequency (number of records)
[0-1000]	10041
[1000-2000]	9913
[2000-3000]	10003
[3000-4500]	14856
[4500-6500]	20144
[6500-8000]	15038
[8000-10000]	20005
Total	100,000

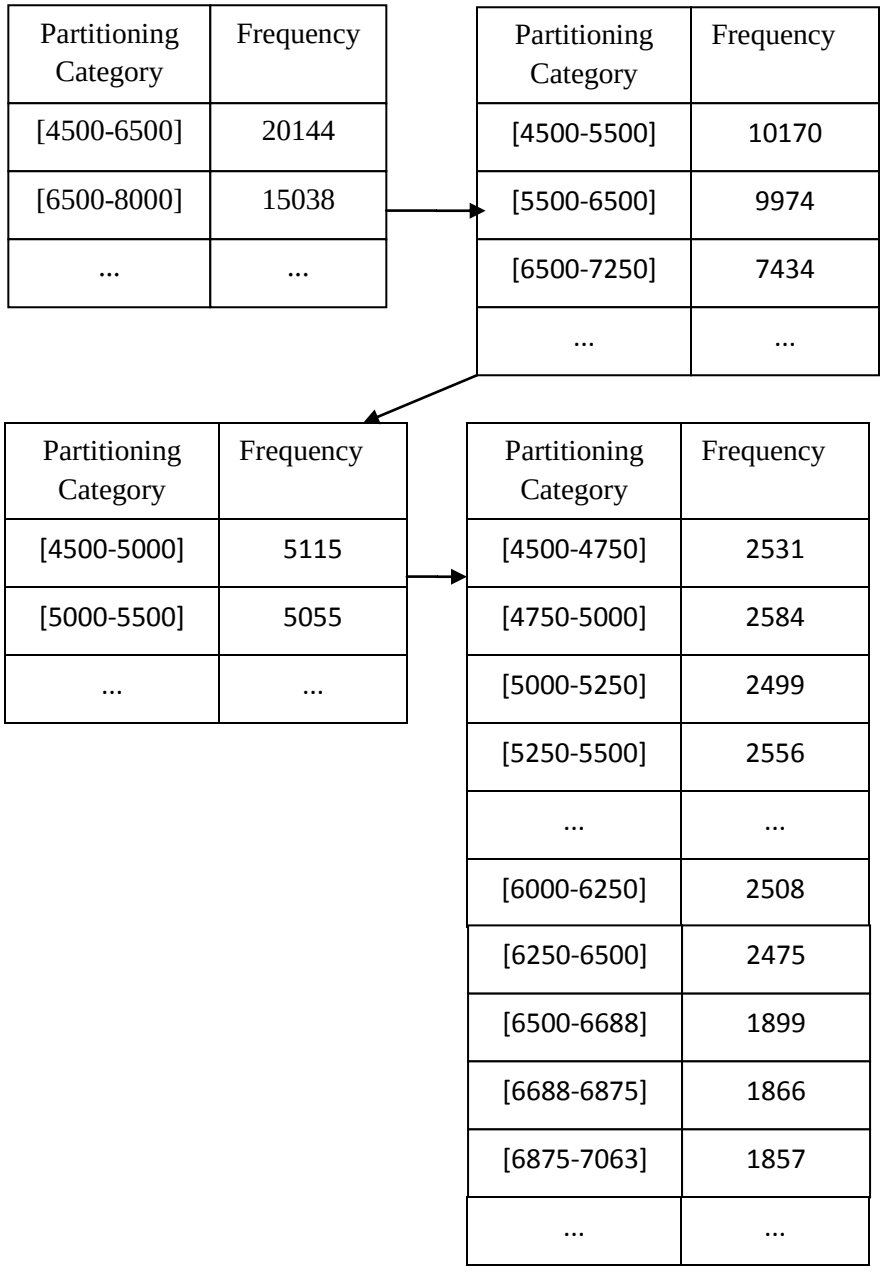


Figure 4.15 Frequency-of-Use-Based and Bisection-Tree-Based method.

4.8 Table Fragmentation

Table fragmentation can be used to divide the original table into many tables. The table can be divided using the idea of the horizontal fragmentation, by which a table can be divided into fragments of tuples. Each fragment has unique rows, and each fragment must have the same attributes [94]. The reasons for fragmenting a relation or table are to improve the efficiency, increase the parallelism or concurrency, and increase the security [94]. When table fragmentation is used, the divided tables are stored in encrypted form without the indexed data. This means, the size of the data that is stored on a Cloud is reduced because the extra indexed data is not stored with tables on the Cloud.

4.8.1 Method of the Table Fragmentation

In this research, the horizontal fragmentation can be used to collect each partition category in one table. Therefore, if the result of the partition category is as explained in Table 4.22, the main table can be divided into 6 tables depending on the number of the categories. The table can be divided into more than one attribute category, and the table must be divided depending on the number of possibilities of all attributes. For example, assume a table is to be divided depending on two attributes. The first attribute, like the one shown in the Table 4.22, has six possibilities and another attribute having three possibilities. So, the total number of divisions of the table will be, 6 possibilities * 3 possibilities= 18 possibilities. Therefore, the table has to be divided into eighteen tables depending on the two attributes where each partition has one possibility from the first attribute and one possibility from the second attribute.

Table 4.22. Partitioning category

Partitioning Category	Category
[400 - 800]	1
[800 - 1200]	2
[1200 - 2500]	3
[2500 - 4000]	4
[4000 - 5000]	5
[5000 - 7000]	6

4.8.2 Comparing Table Fragmentation against One Table

This section will explain the difference between using table fragmentation and one table, which is the whole table. A table having 500,000 records has been used as a whole table to show the differences, and only some records are returned to the client depending on the category. The fragmented table, which is one table that has only one category size, is brought from the Cloud to the client. Figure 4.16 shows the deference between using the table fragmentation and using the original table. It shows in “one category” only one partitioning table, and 14.86% from the whole table are brought from the server. In “3 categories,” three tables are brought from partitioning tables and 50% from the original unfragmented table are brought from the server. In “6 categories,” which mean all data or all categories, which are six tables, and 100% from the whole table are brought from the server. In addition, “6 categories” explains the time consumed when any query requests are based on other attributes, not the partition categories. The graph shows the requested data from the encrypted partitioning table is more efficient than requested data from the original unfragmented table.

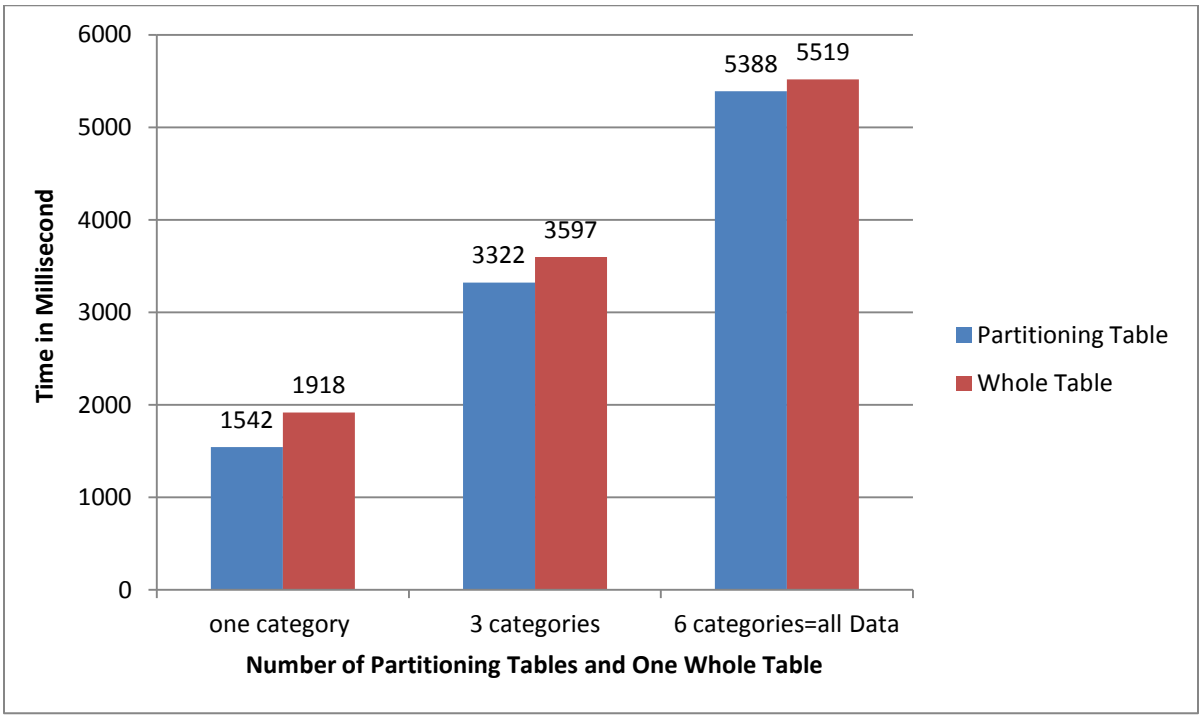


Figure 4.16 Deference between using the table fragmentation and using one table.

5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

This dissertation has studied how to manage and protect any database table on an external server or on a Cloud. The dissertation has discussed how to improve query processing performance while protecting database tables on a Cloud by encrypting those so that they remain secure. It shows how to process SQL queries on encrypted databases designed to protect data from any leakage or attack, even from service providers. The strategy is to process the query on a server or a Cloud without having to decrypt the data, and data decryption is performed only at the client site to enforce security. Additionally, in order to run any SQL query on a Cloud or server in an efficient way, no more than the required data is returned to the client.

An indexed method is used to increase performance. The index data is stored on the Cloud or the server with the encrypted database table. This helps in reducing the entire processing time, which includes data transfer time from the Cloud to the client and also data decryption and processing time at the client. The technique begins by defining the attributes that will be used in all queries, and these attributes will be processed by some operations to get the partition categories. The partition categories are utilized to map any value to a specific range, provided that these categories cover the whole domain, and no two partitions overlap. Four different techniques have been developed to index and partition the data. These techniques are Frequency-of-Use-Based partition, Space-Based partition, Bisection-Tree-Based partition, and Histogram-Based partition. These methods were illustrated with examples in chapter three. Three techniques, Frequency-of-Use-Based partition, Space-Based partition, and Bisection-Tree-Based partition have been compared with the Histogram-Based partition because this technique has been used in paper [19]

to compare the efficiency of each technique. In addition, these techniques have been compared against an unencrypted table and an encrypted table that has no category data. Chapter 4 has explained some graphs and tables with different percentage numbers of the retrieved records from a database table to show and compare the efficiency of the proposed methods. The comparison includes the running time and retrieved records with different number of records in table and different record sizes. The graphs show that the encrypted table with Frequency-of-Use-Based partition and the encrypted table with Bisection-Tree-Based partition are the most efficient kinds of partitions. In addition, the particular issue in chapter four has elucidated how to improve performance by combining Frequency-of-Use-Based and Bisection-Tree-Based methods to partition the data.

5.2 Related Ideas

In this section, some ideas will be introduced to show how the partition will be worked and be used. First, working with more than one related table, consider the relational database schema in Figure 5.1. The Invoice_Detail table has four attributes and one extra attribute, which is Index_Partition, used as a partition category on a Cloud for the Invoice_Number attribute. Therefore, if any table related to Invoice_Detail table is stored on the Cloud, like Items_Invoice_Detail table, the Index_Partition has to be added to the related table. If a query requests data from the two tables, such as bringing all the invoice detail items with it, the client has to generate two queries, one for the Invoice_Detail table and another query for the Items_Invoice_Detail table. At the client, the two queries have to be joined to give the right result to the client. For example, assume that the user requests all details about Invoice_Number=20201. First, the Invoice_Number 20201 is mapped to index partition 2 based on the stored data. Second, the user sends the query: Select * From Invoice_Detail where

Index_Partition=2 and the query: Select * From Items_Invoice_Detail where Index_Partition=2. Third, running code at the client, like Java, links the two queries and gives the right result to the user.

Invoice_Detail Table

Invoice_Number	Inv_Customer	Inv_Date	Inv_Total	Index_Partition
20201	XXX	12-12-2015	$200.000	2
20202	YYY	12-12-2015	$90.50	2
20205	XXX	12-15-2015	$500.00	5

Items_Invoice_Detail Table

Invoice_Number	Item_Number	Item_Qty	Item_Unit_Price	Index_Partition
20201	111222	100	$1.00	2
20201	113456	50	$2.00	2
20202	123099	1	$90.50	2
20205	123444	2	$250.00	5

Figure 5.1 Part of a Database

Second, the technique can use the idea of controlling tables, which is introduced in paper [87]. The idea of controlling tables can be used to protect and check the partitioning data on a Cloud or server. Therefore, the technique can be applied to use one table to sum all partitioning attributes in all records, and the second table to sum all columns of partitioning attributes. Therefore, these tables can be used to define any attempted attacks or find any errors and correct them.

Third, one of the advantages of Cloud computing is supporting parallel computing, which means many calculations and works are carried out simultaneously. Therefore, the partition category technique can benefit from this advantage, and necessary algorithms can be developed to work on database tables on the Cloud. For example, more than one process can be sent to the Cloud. Each process will work on one category of each attribute, and all processes will be collected at the client site to give the result.

5.3 Future Work

Authors in [53] present a mutation-based testing approach for SQL injection vulnerabilities, which is one of the most famous vulnerabilities for web-based applications. They suggest some mutation operators that used to test an application source code to ensure its quality. The paper can be used as a future work to test the partition category for any attributes and the degree of the security of the database tables on the Cloud. Therefore, testing cases and mutation operators can be designed to test all partition categories for all attributes to be sure the processing of all partitions is accurate. The test includes the domain values of all the attributes mapped into the partition categories, whether all partitions can cover the whole domain, and no two partitions overlap. In addition, the work should include test cases and important steps to ensure the quality of the partition category.

REFERENCES

[1] K. Venkataramana, and M. Padmavathamma, "Multi-Tenant Data Storage Security In Cloud Using Data Partition Encryption Technique," International Journal of Scientific & Engineering Research, 2013.

[2] Kleber Vieira, Alexandre Schulter, Carlos Westphall, Carla Westphall, "Intrusion Detection for Grid and Cloud Computing," IT Professional, vol.12, no. 4, July/August 2010, pp. 38-43.

[3] S. Pearson, Y. Shen, M. Mowbray, "A privacy manager for cloud computing," in: Proceedings of the 1st International Conference on Cloud Computing, CloudCom'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 90–106.

[4] S.A. Almulla and C. Y. Yeun, "Cloud computing security management," ICESMA, 2010, pp. 1-7.

[5] Y. Hu, B. Panda, "A data mining approach for database intrusion detection," in: Proceedings of 2004 ACM Symposium on Applied Computing, New York, NY, 2004, pp. 711–716.

[6] Jianneng Cao, Fang-Yu Rao, Mehmet Kuzu, Elisa Bertino, and Murat Kantarcioglu, "Efficient tree pattern queries on encrypted xml documents," In Proceedings of the Joint EDBT/ICDT 2013 Workshops, ACM, 2013, pp. 111-120.

[7] Siani Pearson and Azzedine Benameur, "Privacy, security and trust issues arising from cloud computing," 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010, pp. 693-702.

[8] Steven Y. Ko, Kyungho Jeon, and Ramsés Morales," The HybrEx model for confidentiality and privacy in cloud computing," Proc. of HotCloud, 2011.

[9] Qussai Yaseen, Qutaibah Althebyan, and Yaser Jararweh, "PEP-side caching: An insider threat port," Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference. IEEE, 2013.

[10] Weihan Li, Brajendra Panda, and Qussai Yaseen, "Malicious Users' Transactions: Tackling Insider Threat," Information Security and Privacy Research. Springer Berlin Heidelberg, 2012, pp. 211-222.

[11] Qussai Yaseen and Brajendra Panda, "Tackling Insider Threat in Cloud Relational Databases," IEEE/ACM Fifth International Conference on Utility and Cloud Computing, 2012, pp. 215-218.

[12] Ajeet Ram Pathak, B. Padmavathi "Analysis of Security Techniques Applied in Database Outsourcing," *(IJCSIT) International Journal of Computer Science and Information Technologies*, Vol. 5 (1), 2014, pp. 665-670.

[13] Sunil Sanka, Chittaranjan Hota, and Muttukrishnan Rajarajan, "Secure Data Access in Cloud Computing," *In Internet Multimedia Services Architecture and Application (IMSAA), 2010 IEEE 4th International Conference*, 2010, pp. 1-6.

[14] Pierangela Samarati and Sabrina De Capitani di Vimercati, "Data protection in outsourcing scenarios: Issues and directions," *In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, 2010, pp. 1-14.

[15] Bo Zhao, Benjamin I. P. Rubinstein, Jim gemmell, and Jiaw Han, "A Bayesian approach to discovering truth from conflicting sources for data integration," August 2012, pp. 550-561.

[16] Somchart Fugkeaw, "Achieving Privacy and Security in MultiOwner Data Outsourcing," *In Proc. of IEEE Transactions*, 2012, pp. 239-244.

[17] L. Ferretti, F. Pierazzi, M. Colajannni, and M. Marchetti, "Security and confidentiality solutions for public cloud database services," *in Proc. of the 7th International Conference on Emerging Security Information, Systems and Technologies*, August 2013.

[18] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," *in International Conference on Very Large Data Bases, VLDB,* 2004, pp. 720–731.

[19] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service-provider model," *In Proc. of SIGMOD*, 2002, pp. 216–227.

[20] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained access control in cloud computing," *in Proc. of IEEE INFOCOM'10*, San Diego, CA, USA, March 2010.

[21] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," *In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, Paris, France, June 2004.

[22] Raluca Ada Popa, Catherine Redfield, and Nickolai Zeldovich, "Cryptdb: protecting confidentiality with encrypted query processing," *In Proceedings of SOSP*, 2011.

[23] David F. C. Brewer and Michael J. Nash, "The Chinese Wall Security Policy," *in Proceedings of the 1989 IEEE Symposium on Security and Privacy, IEEE Computer Society Press*, Los Alamitos, CA, 1989, pp. 206–218.

[24] Hassan Takabi, James B.D. Joshi, and Gail-Joon Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Computer and Reliability Societies*, November/December 2010, pp. 24-31.

[25] Hassan Takabi, James B. D. Joshi, "Policy Management as a Service: An Approach to Manage Policy Heterogeneity in Cloud Computing Environment," *45th Hawaii International Conference on System Science*, 2012.

[26] Harini Ragavan, and Brajendra Panda, "Mitigation of malicious modifications by insiders in databases," *In Information Systems Security*, Springer Berlin Heidelberg, 2011, pp. 337-351.

[27] Yang Haixia, and Nan Zhihong, "A database security testing scheme of web application," *In Computer Science & Education, 2009. ICCSE'09. 4th International* Conference *on Computer Science & Education IEEE*, 2009, pp. 953-955.

[28] Ernesto Damiani, S. D. C. D. Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati, "Balancing confidentiality and efficiency in untrusted relational DBMSs," *In Proceedings of the 10th ACM conference on Computer and communications security*, ACM, 2003, pp. 93-102.

[29] Yaseen Q., Panda B., "Knowledge Acquisition and Insider Threat Prediction in Relational Database Systems," *in Proc. of the International Workshop on Software Security Processes*, Canada, 2009, pp. 450-455.

[30] P. Saripalli and B. Walters, "QUIRC: A Quantitative Impact and Risk Assessment Framework for Cloud Security," *IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 280-288.

[31] Shamir A., "How to share a secret," *Communications of the ACM*, vol. 22, no.11, 1979, pp. 612–613.

[32] C. Curino, E. Jones, R. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, "Relational cloud: A database-as-a-service for the cloud," *5th Biennial Conference on Innovative Data Systems Research, CIDR 2011*, January 9-12 2011, pp. 235-240.

[33] C. C. Lo, C. C. Huang, and J. Ku, "Cooperative Intrusion Detection System Framework for Cloud Computing Networks," *First IEEE International Conference on Ubi-Media Computing*, 2008, pp. 280-284.

[34] Q. Yaseen, B. Panda, "Mitigating Insider Threat without Limiting the Availability in Concurrent Undeclared Tasks," *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference*, 20-22 June 2012, pp.235-244.

[35] Qussai Yaseen and Brajendra Panda, "Malicious Modification Attacks by Insiders in Relational Databases: Prediction and Prevention," *IEEE Second International Conference on Privacy, Security, Risk and Trust*, August 2010, pp. 849-856.

[36] Chen, Gang, Ke Chen, and Jinxiang Dong, "A database encryption scheme for enhanced security and easy sharing," *In Computer Supported Cooperative Work in Design, 2006. CSCWD'06. 10th International Conference*, IEEE, 2006, pp. 1-6.

[37] Wassim Itani, Ayman Kayssi, and Ali Chehab, "Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures." *In Dependable, Autonomic and Secure Computing, 2009, DASC'09, Eighth IEEE International Conference*, IEEE, 2009, pp. 711-716.

[38] Mowbray Miranda and Siani Pearson, "A client-based privacy manager for cloud computing," *Proceedings of the fourth international ICST conference on COMmunication system softWAre and middlewaRE.ACM*, 2009, pp. 1-8.

[39] Jeong-Min Do, You-Jin Song, and Namje Park, "Attribute based Proxy Re-Encryption for Data Confidentiality in Cloud Computing Environments," *In Computers, Networks, Systems and Industrial Engineering (CNSI), 2011 First ACIS/JNU International Conference*, 2011, pp. 248-251.

[40] Younis A.Younis, Madjid Merabti and Kashif Kifayat, "Secure cloud computing for critical infrastructure: A survey," *Liverpool John Moores University, United Kingdom, Tech. Rep*, 2013.

[41] Parsi Kalpana, and Sudha Singaraju, "Data security in cloud computing using RSA algorithm," *IJRCCT* 1, no. 4, 2012, pp. 143-146.

[42] Subedari Mithila, and P. Pradeep Kumar, "Data Security through Confidentiality in Cloud Computing Environment," *Subedari Mithila et al, / (IJCSIT) International Journal of Computer Science and Information Technologies* 2, 2011, pp. 1836-1840.

[43] Juan Du, Wei Wei, Xiaohui Gu, and Ting Yu, "RunTest: assuring integrity of dataflow processing in cloud computing infrastructures," *In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ACM, 2010, pp. 293-304.

[44] Juan Du, Nidhi Shah, and Xiaohui Gu, "Adaptive data-driven service integrity attestation for multi-tenant cloud systems," *In Proceedings of the Nineteenth International Workshop on Quality of Service*, IEEE Press, 2011, pp. 1–9.

[45] Kai Hwang, Sameer Kulkareni, and Yue Hu, "Cloud Security with Virtualized Defense and Reputation-based Trust Management," *In Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*, IEEE, 2009, pp. 717-722.

[46] Sebastian Roschke, Feng Cheng, and Christoph Meinel, "Intrusion Detection in the Cloud," *In Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference*, IEEE, 2009, pp. 729-734.

[47] Weichao Wang, Zhiwei Li, Rodney Owens, and Bharat Bhargava, "Secure and Efficient Access to Outsourced Data," *In Proceedings of the 2009 ACM workshop on Cloud computing security*, ACM, 2009, pp. 55-66.

[48] Puya Ghazizadeh, Ravi Mukkamala, and Stephan Olariu, "Data Integrity Evaluation in Cloud Database-as-a-Service," *In Services (SERVICES), 2013 IEEE Ninth World Congress*, IEEE, 2013, pp. 280-285.

[49] Nuno Antunes, and Marco Vieira, "Detecting SQL injection vulnerabilities in web services," *In Dependable Computing, 2009. LADC'09. Fourth Latin-American Symposium*, IEEE, 2009, pp. 17-24.

[50] Wai Kit Wong, Ben Kao, David Wai Lok Cheung, Rongbin Li, and Siu Ming Yiu, "Secure Query Processing with Data Interoperability in a Cloud Database Environment," *In Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, ACM, 2014, pp. 1395-1406.

[51] Percy Antonio Pari Salas, Padmanabhan Krishnan, and Kelvin J. Ross, "Model-based security vulnerability testing," *In Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, IEEE, 2007, pp. 284-296.

[52] Marco Vieira, Nuno Antunes, and Henrique Madeira, "Using web security scanners to detect vulnerabilities in web services," In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference*, IEEE, 2009, pp. 566-571.

[53] Hossain Shahriar, and Mohammad Zulkernine, "MUSIC: Mutation-based SQL injection vulnerability checking," *In Quality Software, 2008. QSIC'08. The Eighth International Conference on Quality Software*, IEEE, 2008, pp. 77-86.

[54] Hossain Shahriar, and Mohammad Zulkernine, "Mutation-based testing of buffer overflow vulnerabilities," *In Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, IEEE, 2008, pp. 979-984.

[55] Nuno Laranjeiro, Marco Vieira, and Henrique Madeira, "Protecting Database Centric Web Services against SQL/XPath Injection Attacks," *In Database and Expert Systems Applications*, Springer Berlin Heidelberg, 2009, pp. 271-278.

[56] Ke He, Zhiyong Feng, and Xiaohong Li, "An attack scenario based approach for software security testing at design stage," *In Computer Science and Computational Technology, 2008. ISCSCT'08*, vol. 1, IEEE, 2008, pp. 782-787.

[57] Aaron Marback, Hyunsook Do, Ke He, Samuel Kondamarri, and Dianxiang Xu, "Security test generation using threat trees," *In Automation of Software Test, 2009. AST'09*, IEEE, 2009, pp. 62-69.

[58] Dazhi Zhang, Donggang Liu, Yu Lei, David Kung, Christoph Csallner, and Wenhua Wang, "Detecting vulnerabilities in c programs using trace-based testing," In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference*, IEEE, 2010, pp. 241-250.

[59] Rafael Bosse Brinhosa, Carlos Becker Westphall, and Carla Merkle Westphall, "A security framework for input validation," *In Emerging Security Information, Systems and Technologies, 2008. SECURWARE'08. Second International Conference*, IEEE, 2008, pp. 88-92.

[60] Sarah Al-Azzani, and Rami Bahsoon, "Using implied scenarios in security testing," *In Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, ACM, 2010, pp. 15-21.

[61] Minqi Zhou, Rong Zhang, Wei Xei, Weining Qian, and Aoying Zhou, "Security and Privacy in Cloud Computing: A Survey," *Sixth International Conference on Semantics, Knowledge and Grids*, November 2010, pp. 105-112.

[62] Hyun-Suk Yu, Yvette E. Gelogo, and Kyung Jung Kim, "Securing Data Storage in Cloud Computing," *Journal of Security Engineering*, June 2012, pp. 251-259.

[63] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good, "The Cost of Doing Science on the Cloud: The Montage Example," Proceedings of the 2008 ACM/IEEE conference on Supercomputing, 2008, pp. 50.

[64] Shilpashree Srinivasamurthy, and David Liu, "Survey on cloud computing security," *In Proc. Conf. on Cloud Computing, CloudCom*, vol. 10, 2010.

[65] B. Grobauer, T. Walloschek and E. Stöcker, "Understanding Cloud Computing Vulnerabilities," IEEE Security and Privacy, vol. 99, 2010.

[66] Y. Vijaya Ratna Kumari, T. Bindu Madhavi, and L. Ravi Kumar, "Efficient and Secure Scheme for Distributed Data Storage Systems," *International Journal of Computer Science and Information Technologies (IJCSIT)*, Vol. 6 (1), 2015, pp. 839-843.

[67] Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu. "RCFile: A Fast and Space-efficient Data Placement Structure in MapReduce-based Warehouse Systems." In *Data Engineering (ICDE), 2011 IEEE 27th International Conference*, IEEE, 2011, pp. 1199-1208.

[68] Bala Iyer, Sharad Mehrotra, Einar Mykletun, Gene Tsudik, and Yonghua Wu. "A framework for efficient storage security in RDBMS." In *Advances in Database Technology-EDBT 2004*, Springer Berlin Heidelberg, 2004, pp. 147-164.

[69] Pengcheng Xiong, Yun Chi, Shenghuo Zhu, Hyun Jin Moon, Calton Pu, and Hakan Hacigümüş. "Intelligent management of virtualized resources for database systems in cloud environment." In *Data Engineering (ICDE), 2011 IEEE 27th International Conference*, IEEE, 2011. pp. 87-98.

[70] Liang Zhao, Anna Liu, and Jacky Keung. "Evaluating cloud platform architecture with the care framework." In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, IEEE, 2010. pp. 60-69.

[71] Markus Klems, David Bermbach, and Rene Weinert. "A runtime quality measurement framework for cloud database service systems." In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference*, IEEE, 2012. pp. 38-46.

[72] Indu Arora, and Anu Gupta. "Cloud databases: a paradigm shift in databases." *International J. of Computer Science Issues* 9, no. 4, 2012. pp. 77-83.

[73] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. "Runtime measurements in the cloud: observing, analyzing, and reducing variance."*Proceedings of the VLDB Endowment* 3, no. 1-2, 2010. pp. 460-471.

[74] Liang Zhao, Sherif Sakr, Alan Fekete, Hiroshi Wada, and Anna Liu. "Application-managed database replication on virtualized cloud environments." In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference*, IEEE, 2012, pp. 127-134.

[75] Thibault Dory, Boris Mejías, P. V. Roy, and Nam-Luc Tran. "Measuring elasticity for cloud databases." In *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*. 2011.

[76] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. "CloudCmp: comparing public cloud providers." In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, ACM, 2010. pp. 1-14.

[77] Dawei Jiang, Beng Chin Ooi, Lei Shi, and Sai Wu. "The performance of mapreduce: An in-depth study." *Proceedings of the VLDB Endowment* 3, no. 1-2, 2010, pp. 472-483.

[78] Byung Tak Chul, Bhuvan Urgaonkar, and Anand Sivasubramaniam. "To move or not to move: The economics of cloud computing." In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, USENIX Association, 2011, pp. 5-5.

[79] Liangzhe Li, and Le Gruenwald. "Autonomous database partitioning using data mining on single computers and cluster computers." In *Proceedings of the 16th International Database Engineering & Applications Sysmposium*, ACM, 2012, pp. 32-41.

[80] Rohit Jain, and Sunil Prabhakar. "Access control and query verification for untrusted databases." In *Data and Applications Security and Privacy XXVII*, Springer Berlin Heidelberg, 2013, pp. 211-225.

[81] Shuai Zhang, Shufen Zhang, Xuebin Chen, and Xiuzhen Huo. "Cloud computing research and development trend." In *Future Networks, 2010. ICFN'10. Second International Conference*, IEEE, 2010, pp. 93-97.

[82] C. P. Pfleeger and S. L. Pfleeger. Security in Computing (4th Edition). Prentice Hall, 2006.

[83] Hassan A. Afyouni. Database Security and Auditing: Protecting data integrity and accessibility. Cengage Learning, 2006.

[84] Liang Zhao, Sherif Sakr, Anna Liu, and Athman Bouguettaya. Cloud Data Management. Springer, 2014.

[85] Michael G. Solomon, Vaidy Sunderam, and Li Xiong, "Towards Secure Cloud Database with Fine-Grained Access Control." In *Data and Applications Security and Privacy XXVIII*, Springer Berlin Heidelberg, 2014, pp. 324-338.

[86] Ralph P. Grimaldi. Discrete and Combinatorial Mathematics: An Applied Introduction, 3th Edition. Addison-Wesley, 1994.

[87] Osama M. Ben Omran, Brajendra Panda. "Efficiently Managing Encrypted Data in Cloud Databases." *In 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, IEEE, 2015, pp. 266-271.

[88] Wei Xu, V. N. Venkatakrishnan, R. Sekar, and I. V. Ramakrishnan. "A framework for building privacy-conscious composite web services." In *Web Services, 2006. ICWS'06. International Conference*, IEEE, 2006, pp. 655-662.

[89] Elisa Bertino, Federica Paci, Rodolfo Ferrini, and Ning Shang. "Privacy-preserving Digital Identity Management for Cloud Computing." *IEEE Data Eng. Bull.* 32, no. 1, 2009, pp. 21-27.

[90] Osama M Ben Omran, and Brajendra Panda. "A Data Partition Based Model to Enforce Security in Cloud Database." In *Journal of Internet Technology and Secured Transactions (JITST), Volume 3, Issues 3/4, September/December 2014*, 2014.

[91] Osama M. Ben Omran, and Brajendra Panda. "A new technique to partition and manage data security in cloud databases." In *Internet Technology and Secured Transactions (ICITST), 2014 9th International Conference for*, IEEE, 2014, pp. 191-196.

[92] Youssef Gahi, Mouhcine Guennoun, and Khalil El-Khatib. "A secure database system using homomorphic encryption schemes." *arXiv preprint arXiv:1512.03498*, 2015.

[93] Swarnalata Bollavarapu, and Kamal Mistry. "Secure Database as a Service-a Review." In *International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 3, March 2015*, 2015, pp. 425-429.

[94] Thomas M. Connolly, and Carolyn E. Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.

[95] V.R. Pancholi and B.P. Patel 2016. Enhancement of Cloud Computing Security with Secure Data Storage using AES. *International Journal for Innovative Research in Science and Technology*, *2*(9), pp. 18-21.

[96] Gurpreet Singh and Supriya "A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security." *International Journal of Computer Applications* Vol. 67, No. 19, 2013, pp. 33-38.

[97] A. Azarudeen, N. Ganesh, R. Dinesh, and R. Ramakrishnan. "Secure Storage using TPC-C in Cloud." In *2015 JSRSET*, Vol. 1, Issue 2, 2015, pp. 58-61.

[98] Kevin D. Bowers, Catherine Hart, Ari Juels, and Nikos Triandopoulos. "Securing the Data in Big Data Security Analytics." *IACR Cryptology ePrint Archive* 2013, 2013, pp. 1-14.

[99] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. "Big data and cloud computing: current state and future opportunities." In *Proceedings of the 14th International Conference on Extending Database Technology*, ACM, 2011, pp. 530-533.

[100] Niloofar Khanghahi and Reza Ravanmehr. "CLOUD COMPUTING PERFORMANCE EVALUATION: ISSUES AND CHALLENGES." *Comput* 5, no. 1, 2013, pp. 29-41.

[101] Einar Mykletun and Gene Tsudik. "Aggregation queries in the database-as-a-service model." In *Data and Applications Security XX*, Springer Berlin Heidelberg, 2006, pp. 89-103.