ENHANCING QUERY PERFORMANCE AND PRIVACY ON RELATIONAL CLOUD DATABASE

BY

ROSEMARY MRUMUN DZER

DEPARTMENT OF MATHEMATICS,
FACULTY OF SCIENCE,
AHMADU BELLO UNIVERSITY, ZARIA, NIGERIA

MARCH, 2016

ENHANCING QUERY PERFORMANCE AND PRIVACY ON RELATIONAL CLOUD DATABASE

BY

Rosemary Mrumun DZER
B.Sc Computer Science(UniJos), 2008
MSc/SCI/7648/2011-2012

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF A
MASTER OF SCIENCE DEGREE IN COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS,
FACULTY OF SCIENCE,
AHMADU BELLO UNIVERSITY, ZARIA, NIGERIA

MARCH, 2016

Declaration

I declare that the work in this dissertation entitled "*Enhancing Query Performance and Privacy on Relational Cloud Database*" has been carried out by me in the Department of Mathematics under the supervision of Dr. A.A. Obiniyi and Dr. S.E. Abdullahi.

The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this dissertation was previously presented for another degree or diploma at this or any other Institution.

DZER, Rosemary Mrumun	_____	_____
Name of Student	Signature	Date

Certification

This dissertation entitled "ENHANCING QUERY PERFORMANCE AND PRIVACY ON RELATIONAL CLOUD DATABASE" by Rosemary Mrumun DZER meets the regulations governing the award of the degree of Master of Science in Computer Science of Ahmadu Bello University, Zaria, and is approved for its contribution to knowledge and literary presentation.

_____	_____
Dr. A.A. Obiniyi Chairman, Supervisory Committee	Date
_____	_____
Dr. S.E. Abdullahi Member, Supervisory Committee	Date
_____	_____
Prof. S.E. Adewumi External Examiner	Date
_____	_____
Prof. A.A. Tijjani Head of Department	Date
_____	_____
Prof. K. Bala Dean, School of Postgraduate Studies	Date

Dedication

This dissertation is dedicated to the Almighty God and father of our Lord Jesus Christ, whose mighty hand guided me through the turbulent times and sustained me with grace, surrounded me with love and favour to undertake this research work to the very end.

Acknowledgement

My unalloyed gratitude goes to God Almighty, who in His infinite mercy bestowed me with grace and wisdom and the divine connections that enabled me to accomplish this research.

Secondly, I express my sincere thanks to my Supervisors – Dr. A.A. Obiniyi and Dr. S.E. Abdullahi. Your constant but gentle push, constructive criticisms, fatherly advice and forgiveness of my flaws inspired me greatly. Your weekly seminars kept me on track. Keep it up and may God reward your good deeds.

Furthermore, I wish to appreciate the Head, Department of Mathematics – Prof. B. Sani, the Seminar Coordinator – Dr. A.M. Ibrahim, the Post Graduate Coordinator – Dr. Abubakar Yahaya, all my lecturers (Prof. S.B. Junaidu, Dr. Hammawa, Prof. Adewale), the Computer Lab Technicians – Mal Abdullahi and Mal Jaafaru and all the staff of the Department. I would not have made it without your support.

My heartfelt gratitude to Mr. Chika Ugwu, whose technical ingenuity put the pieces of blocks in their places and together we raised the structure to a perfect finish. May God Almighty reward you.

Special thanks to Federal University Dutsinma, for the study fellowship granted to me which cushioned the financial burden of this study. Also, to Prof. J.O. Fatokun (DVC), Mrs M.O. Olanrewaju (Head, Department of Mathematics), Dr. A.O. Yusuf, Mrs Balogs, Mr. Chinedu Peter, Mr. R. Tyokyaa and all others not mentioned individually, I am grateful for your encouragement and support.

To my friends and course mates – Vivian, Sikirat (my encourager), Benny, ThankGod, Femi the endowed classrep, Solomon, Maryam etc, we started together, but some halted along the way; may our end be more beautiful than the beginning. One Love!

In addition, to my parents – Mr and Mrs Boniface H. Dzer, my siblings – Bro Joseph, Bro Godwin and family, Bro Charles and family, Eunice, Felix, Bridget, David and family, I say thank you for your love, support and prayers. May the bond of love and unity never be broken.

Finally, my warmest kiss of love to my treasure, my beloved, my husband – Mr John T. Dima, for your love, encouragement, prayers and the sacrifices you made to ensure I move forward. May God bless and keep you for me.

To all my friends and well-wishers too numerous to mention, God bless you richly.

Abstract

Database outsourcing has become a trend in the information technology industry because it offers scalability to the enormous amount of digital content stored and generated on a daily basis by individuals and corporations. In large outsourced databases the efficiency of data retrieval, especially as it relates to privacy, remains an open challenge, because traditional query languages cannot work with encrypted data. While several architectures, techniques and tools have been proffered to ensure that privacy and performance are balanced and optimized, each of these approaches has its limitations. This research proposed a framework which focused on enhancing query performance and privacy through the use of hash map and AES 128-bit encryption algorithm. The design and implementation of *secureSQL* is built on the client-side without any alteration to the DBMS structure. *SecureSQL* model guarantees efficiency and is able to execute 20 out of the 22 Transaction Processing Performance Council (TPC-H) benchmark queries while ensuring privacy. This is proof that it is not restricted to simple query constructs but is able to handle even complex queries involving nested sub queries and joins. The execution time of queries between the client and database on the cloud is reduced, as is evident in the comparative performance analysis between *secureSQL* and the traditional method of querying. Also, with increasing number of records, the proposed method maintains some degree of constancy in execution time thereby supporting the O(1) time complexity assertion for the use of the hash map data structure.

Table of Contents

Title Page ii

Declaration iii

Certification iv

Dedication v

Acknowledgement vi

Abstract vii

Table of Contents viii

List of Figures xiii

List of Tables xv

Abbreviations and Acronyms xvi

1.0 INTRODUCTION 1

1.1 Background of Study 1

1.2 Cloud Service Models 5

1.2.1 Software-as-a-Service (SaaS): 5

1.2.2 Platform-as-a-Service (PaaS): 6

1.2.3 Infrastructure-as-a-Service (IaaS): 6

1.2.4 Database-as-a-Service (DBaaS): 6

1.3 Cloud Deployment Models 7

1.3.1 Private Cloud: 7

1.3.2 Community Cloud: 7

1.3.3 Public Cloud: 8

1.3.4 Hybrid Cloud: 8

1.4 Research Questions 8

1.5 Problem Statement 9

1.6 Motivation 9
1.7 Aim and Objectives of the research 10
1.8 Research Method 10
1.9 Contribution to Knowledge 11
1.10 Organization of Dissertation 12

2.0 REVIEW OF LITERATURE 13
2.1 Introduction 13
2.2 Concept of Cloud Database 13
2.3 Types of Cloud Database 14
2.3.1 SQL Databases 14
2.3.2 NoSQL DATABASES 15
2.3.3 SQL vs NoSQL 16
2.4 Data Security 16
2.4.1 Confidentiality 17
2.4.2 Availability 17
2.4.3 Integrity 18
2.4.4 Access Control 18
2.5 Concept of Database Security 18
2.5.1 Levels of Encryption 20
2.6 Data Security Concerns 22
2.6.1 Encryption Overhead 22
2.6.2 Handling Encryption/Decryption Keys 22
2.6.3 Client Side Encryption 23
2.7 Terms used in Cryptography 23
2.8 Types of Ciphers 24

2.8.1 Block Ciphers 24
2.8.2 Stream Ciphers 29
2.9 Symmetric Encryption 30
2.9.1 Advantages of Symmetric Encryption 31
2.10 Advanced Encryption Standard 32
2.11 Choice of Encryption Granularity and Algorithm 35
2.12 Query Efficiency 36
2.13 Hash Map 37
2.14 Benchmarking 38
2.15 Review of Related Literature 38

3.0 ARCHITECTURE AND DESIGN OF SEQURE SQL 45
3.1 Introduction 45
3.2 System Model 45
3.3 Assumptions 47
3.4 System Architecture 47
3.4.1 Key Storage 48
3.4.2 Authentication 48
3.4.3 Encryption/Decryption Engine 48
3.4.4 Query Execution 49
3.5 System Design 53
3.6 Implementation Tools and Platform 54
3.6.1 MySQL Workbench 54
3.6.2 Google Cloud SQL 55
3.6.3 TPC-H Benchmark 55
3.6.4 DBGen (Database Generator) 56

4.0 IMPLEMENTATION, RESULTS AND DISCUSSION 57

4.1 Introduction 57

4.2 Generation of test data with dbgen for the tpc-h database using visual studio 57

4.3 Data loading into the database from the flat .tbl files 60

4.4 TPCH Schema Diagram 63

4.5 System Implementation 64

4.5.1 Code Implementation 64

4.5.2 Client Side Application 65

4.5.3 Interface Tabs 67

4.5.4 Encryption Process 67

4.5.5 Columns encrypted in the TPCH database 69

4.5.6 Decryption Process 69

4.5.7 Update Process 70

4.5.8 Delete Process 70

4.6 Google Cloud SQL instance connection 71

4.7 Export and Import of database to google cloud SQL 72

4.8 Query Testing, Results and Discussion 75

4.9 Observation and Discussion 81

4.10 Comparison of Methods 83

5.0 SUMMARY, CONCLUSION AND RECOMMENDATION 86

5.1 Summary 86

5.2 Conclusion 87

5.3 Recommendation for future work 88

REFERENCES 90

APPENDIX: SAMPLE PROGRAM CODE 98

List of Figures

Figure 1.1: Segments of Cloud Computing (Aderounmu, 2012) 3

Figure 1.2: Cloud Service Models (Aderounmu, 2012) 5

Figure 1.3: Database as a Service 7

Figure 2.1: Levels of Encryption (Bouganim and Guo, 2009) 22

Figure 2.2: Block Cipher Operation (Lafourcade, 2008) 25

Figure 2.3: ECB Mode Encryption (Lafourcade, 2008) 26

Figure 2.4: ECB Mode Decryption (Lafourcade, 2008) 26

Figure 2.5: CBC Mode Encryption (Lafourcade, 2008) 27

Figure 2.6: CBC Mode Decryption (Lafourcade, 2008) 27

Figure 2.7: CFB Mode Encryption (Lafourcade, 2008) 28

Figure 2.8: CFB Mode Decryption (Lafourcade, 2008) 28

Figure 2.9: OFB Mode Encryption (Lafourcade, 2008) 29

Figure 2.10: OFB Mode Decryption (Lafourcade, 2008) 29

Figure 2.11: Stream Cipher Operation (Tantawy, 2010) 30

Figure 2.12: Byte Substitution (Lafourcade, 2008) 33

Figure 2.13: Shift Rows (Lafourcade, 2008) 34

Figure 2.14: Mix Columns (Lafourcade, 2008) 34

Figure 2.15: Round Key Addition (Lafourcade, 2008) 35

Figure 3.1: Overview of the Proposed System Model 46

Figure 3.2: Proposed System Architecture 47

Figure 3.3: Encryption/Decryption Engine Architecture 48

Figure 3.4: Query Execution Process 51

Figure 4.1: Generating dbgen.exe using Microsoft Studio 58

Figure 4.2: Command Prompt showing dbgen.exe directory 58

Figure 4.3: dbgen.exe Help Options 59
Figure 4.4: Error in Data Generation 59
Figure 4.5: Generation of Eight Flat File(.tbl) Tables 59
Figure 4.6: TPCH Schema 64
Figure 4.7: *SecureSQL* Graphical User Interface (GUI) 65
Figure 4.8: *SecureSQL* GUI with Query result set 65
Figure 4.9: Batch Encrypt Processes 68
Figure 4.10: Single row encryption 68
Figure 4.11: A Unified View of Selected Encrypted Columns in the TPCH Database ..69
Figure 4.12: Decryption in Process 70
Figure 4.13: Authorize Access to Google Cloud Instance 72
Figure 4.14: Exporting Data in .sql Format 72
Figure 4.16: Dumping of the .sql files on Google Cloud Storage 73
Figure 4.17: Uploading .sql files to Cloud Storage Bucket 73
Figure 4.18: Importing Data from Cloud Bucket to Cloud SQL 74
Figure 4.19: Viewing the Cloud Tables from mysql Client 74
Figure 4.20: Connecting *SecureSQL* to Google Cloud SQL 75
Figure 4.21: Graph of Time vs Records for Query 1 77
Figure 4.22: Graph of Time vs Records for Query 2 78
Figure 4.23: Graph of time vs records for query 6 79
Figure 4.24: Graph of Time vs Records for Query 16 80
Figure 4.25: Graph of Time vs Queries using Log Scale 81
Figure 4.26: Chart of Query Type Support and Key Handling 84
Figure 4.27: Comparison of Queries Handled 85
Figure 4.28: Graph of Time vs Number of Records 85

List of Tables

Table 2.1: SQL vs NoSQL 16

Table 3.1: Query Execution Algorithm 52

Table 3.1: TPCH Tables and Number of Records 56

Table 4.1: Selective Columns Encrypted in the TPCH Schema 69

Table 4.2: Mean Execution Time for Query 1 76

Table 4.3: Mean Execution Time for Query 2 78

Table 4.4: Mean Execution Time for Query 6 79

Table 4.5: Mean Execution Time for Query 16 80

Table 4.6: Comparison of Methods 83

Table 4.7: Comparison of Methods Redefined 84

Abbreviations and Acronyms

AES:	Advanced Encryption Standard
Amazon EC2:	Amazon Elastic Cloud 2
Amazon S3:	Amazon Simple Storage Service
API:	Application Program Interface
BASE:	Basically Available, Soft state, Eventual consistency
CBC:	Cipher Block Chaining
CSP:	Cloud Service Provider
DaaS:	Database as a Service
DAS:	Database as a Service
DBA:	Database Administrator
DBaaS:	Database as a Service
DBMS:	Database Management System
DDL:	Data Definition Language
DDoS:	Distributed Denial of Service
DML:	Data Manipulation Language
IaaS:	Infrastructure as a Service
IBM:	International Business Machines
NIST:	Nigerian Institute of Standards and Technology
NoSQL:	Not only structured query language
PaaS:	Platform as a Service
PDA:	Personal Digital Assistant
SaaS:	Software as a Service
SLA:	Service Level Agreement
SQL:	Structured Query Language

SSL:	Secure Sockets Layer
TLS:	Transport Layer Security
TPC-H:	Transaction Processing Performance Council decision support
XML:	Extensible Markup Language

CHAPTER ONE

INTRODUCTION

1.1 Background of Study

Database outsourcing has become a current trend in the Information Technology Industry due to the enormous amount of digital content stored and generated on a daily basis by individuals and corporations. According to Liu and Ting (2010), enterprises are becoming data-centric and increasingly producing huge amounts of data daily. Most importantly, is the management of the data as it transits to and from the storage servers. Database as a service (DaaS) model provides users with Internet access, the power to create, store, modify and retrieve data from anywhere in the world. This technology primarily saves cost in terms of procuring hardware, software and manpower. It also ensures elastic scaling and availability. A key problem in outsourcing the storage and processing of data is that parts of the data may be sensitive, such as business secrets, credit card numbers, health records or other personal information (Kamara and Lauter, 2010).

Generally, this work is framed in the context of relational databases, since they are by far the most common and workable solution for the management of data in most environments. Relational databases are also characterized by a clear data model and a simple query language that facilitate the design of a solution.

A larger scalability problem comes from MySQL. Relational databases are limited in capacity due to the way they represent information. When that limit is reached, database management becomes more difficult. A procedure called data partitioning can be used to manage this. Using this method, the data is split into independent sets, and indefinite scaling can be done. But if the data cannot be split, then movement to a distributed

database is the next option which implies a cloud solution. This is beneficial because the cloud allows for scaling indefinitely; it just means that more servers need to be added.

Databases are traditionally protected by means of access control mechanism. This method guarantees security only when the data resides on a trusted server. A critical question is: "to what extent is the confidentiality of sensitive data guaranteed once it is outsourced to a third party?" Cloud database poses security risks to the sensitive contents due to possible malicious activities by internal and external parties. This challenge can be addressed by encrypting data before outsourcing in order to keep them hidden from the service provider (Davida *et. al*., 1981).

The storage and processing of encrypted data on storage servers is very desirable but implies a sacrifice of functionality for security (Song *et al*., 2000). Traditional encryption schemes prevent the execution of most SQL queries through the Database Management System (DBMS) engine.

Previous research by Hacigumus *et al.*, (2002a) and Popa *et al*., (2011), proposed encryption schemes that allow the execution of Structured Query Language (SQL) queries over encrypted data. These architectures are based on a trusted intermediate proxy, which accesses the database on behalf of the clients.

Cloud is a broad solution that delivers IT as a service. It is a pattern of parallel and distributed system composed of a collection of interconnected and virtualized computers that are dynamically stipulated and presented as one or more unite computing resources established on Service Level Agreements (SLA's) found amongst negotiation between the service supplier and consumer. It uses remote services through a network using various resources. It is basically meant to give maximum capability of

computing with the minimum hardware resources at the user end. This is possible only through this technology which requires and utilizes its resources in the best way.

The United States National Institute of Standards and Technology (NIST), defines Cloud Computing as “a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”(Badger *et al.*, 2011). Cloud computing realizes computing as a utility by providing a pool of resources which can be allocated to users dynamically according to their requirement. Such resources are accessible by devices like mobile phones, laptops, ipads, Personal Digital Assistant’s anywhere and anytime (Xiao and Xiao, 2014).

Cloud computing is basically broken down into three segments: "application"(client computer) "storage"(datacenter) and "connectivity"(distributed servers) (Aderounmu, 2012) as depicted in Figure 1.1. The datacenter comprises the Cloud Service Models – Software-as-a-Service, Platform-as-a-Service and Infrastructure-as-a-Service.

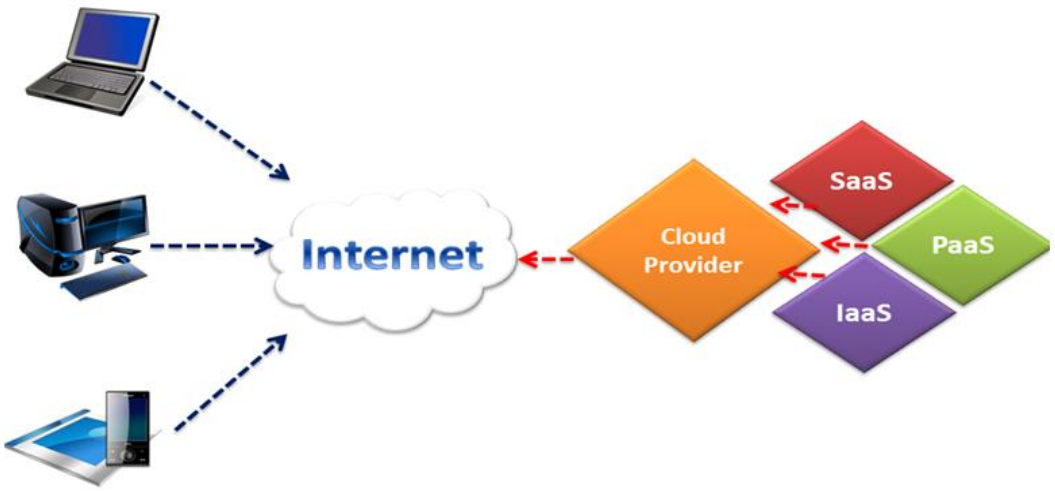


Figure 1.1: Segments of Cloud Computing (Aderounmu, 2012)

Cloud computing is an elevated form of grid computing, parallel or distributed computing which evolves to accelerate the idea of sharing resources expeditiously by offering access and the use of multiple server-based computational resources via digital networks where user may access the server resources using any kind of computing devices. (Badger *et al.*, 2011)

Cloud computing has five essential characteristics (National Institute of Standards and Technology, 2011):

a. On-demand self-service
b. Broad network access
c. Resource pooling
d. Rapid elasticity
e. Measured service

These characteristics are explained as follows:

a. On-demand self-service:

Clients can automatically request and obtain computing provisions such as "server time and network storage", without requiring human interaction with service provider.

b. Broad network access:

Access to services is available over the network through multiple platforms such as cellular phones, laptops, and Personal Digital Assistants.

c. Resource pooling:

The providers co-locate resources such as applications, memory, bandwidth, virtual machines, to service many users regardless of location using a multi-tenant model.

d. Rapid elasticity:

Resources can be elastically provisioned and released (often automatically) and in a scalable manner.

e. Measured service:

Cloud computing provider transparently measures, monitors, controls and documents service usage for billing.

1.2 Cloud Service Models

In a cloud environment, services provided to a user are categorized as Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) as shown in Figure 1.1. Database as a Service (DbaaS) is derived from SaaS, PaaS and IaaS. Figure 1.2 clearly shows the level of abstraction of each service model (Aderounmu, 2012).

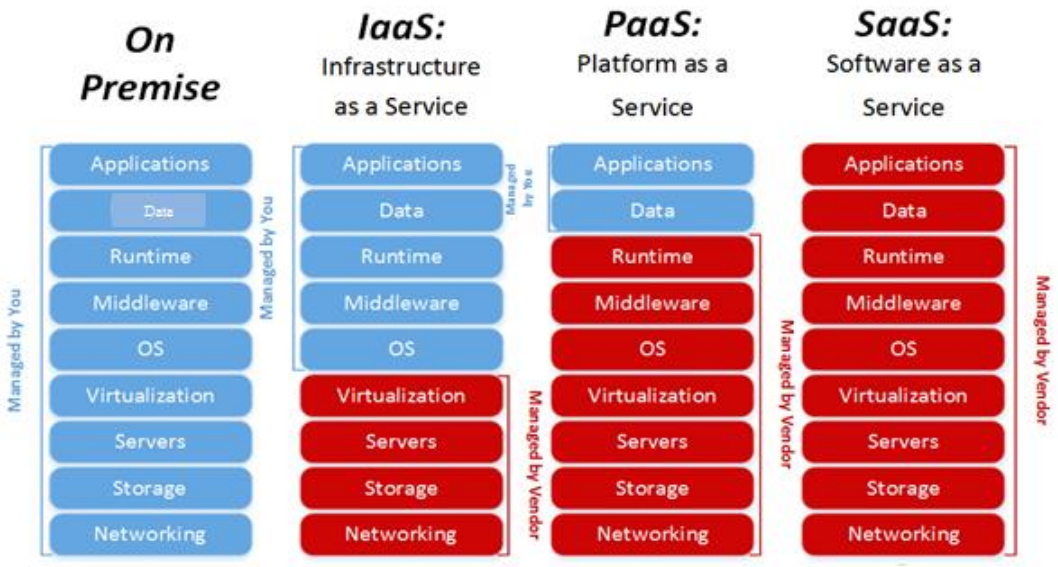


Figure 1.2: Cloud Service Models (Aderounmu, 2012)

1.2.1 Software-as-a-Service (SaaS):

The Cloud Service Provider (CSP) grants the client access to software and its functions remotely as a web-based browser such that the client only controls individual

configuration settings for user-specific applications. This eliminates the possibility of organization handling installation, set-up and maintenance. SaaS include: Gmail, Google Apps, MS Office 365, Onlive, GT Nexus (Badger *et al.*, 2011).

1.2.2 Platform-as-a-Service (PaaS):

The CSP provides an enabling platform such as operating system, programming language execution environment, database and web servers for the client to develop and deploy their applications on a cloud infrastructure with no cost and difficulty of acquiring and handling the main hardware and software films. The client controls deployed applications and the configuration settings but not the underlying cloud infrastructure. Platforms such as Google App Engine, Microsoft Windows Azure, Amazon Elastic Cloud 2 are PaaS (Badger *et al.*, 2011).

1.2.3 Infrastructure-as-a-Service (IaaS):

The CSP offers hardware infrastructure like servers, routers, storage and other networking components in the form of virtual machines as a service for the client to control. This offers flexibility and scaling (up and down) to the user which allow for the management of spikes no matter the load. IaaS include examples such as Amazon Elastic Cloud 2, Rackspace open cloud, HP Cloud (Badger *et al.*, 2011).

1.2.4 Database-as-a-Service (DBaaS):

The SaaS, PaaS and IaaS categories extend to give rise to another service known as Database-as-a-service (DBaaS) as shown in figure 1.3, in which the IaaS can be extended to provide multiple dedicate database storages, SaaS can widen to support specific database interfaces and PaaS can be expanded to allow database management service tools. This provides the client seamless mechanisms to create store and access their databases at the host site (Donkena and Gannamani, 2012).

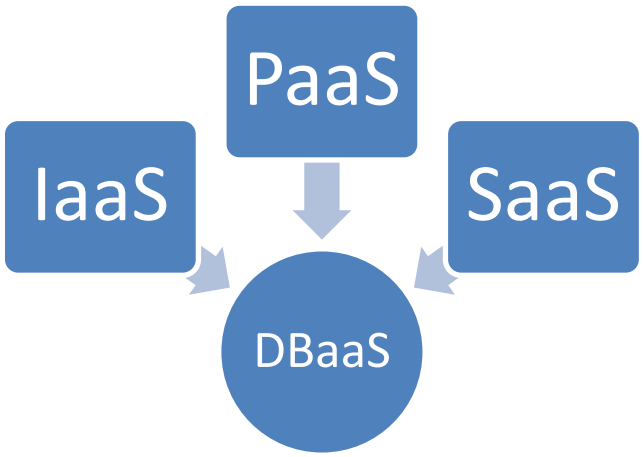


Figure 1.3: Database as a Service

Currently, relational and non-relational database services are supported which guarantee scalability, high availability and stability (Webopedia, 2012).

1.3 Cloud Deployment Models

There are four deployment models for cloud services:

1.3.1 Private Cloud:

Private cloud model provides infrastructure for exclusive use by a single organization within its firewall who can further grant resources, storage of data to a limited number of hosted services within its boundaries. Private cloud removes a number of security concerns inherent to the cloud computing model and is ideal for adaptation when it comes to protecting highly sensitive data (Sharma and Trivedi, 2014).

1.3.2 Community Cloud:

In community cloud model, the cloud infrastructure is provided for and used exclusively by a community of users that have the same requirements and concerns such as security needs, law compliance and policy considerations. A cloud service provider can be chosen for the management and operational tasks. This model is relatively secure and

more cost efficient than owning an on premise private cloud (Sharma and Trivedi, 2014).

1.3.3 Public Cloud:

In public cloud model, the cloud infrastructure is built and managed by the cloud service provider and available for open use by everyone with Internet access. It may be owned and managed by academia, government or corporate bodies. Public clouds are not very secure as compared to private and community cloud (Sharma and Trivedi, 2014).

1.3.4 Hybrid Cloud:

A hybrid cloud is formed when two or more distinct cloud infrastructures are used together. This is necessary when there are different security requirements for different sets of data. An organization provides and manages some resources in-house and has others provided externally. It offers the benefit of multiple deployment models to the users (Sharma and Trivedi, 2014).

1.4 Research Questions

a. What are the issues that affect performance of a cloud database with respect to client/database communication over the Internet?
b. What is the performance in terms of execution time of client/cloud database queries using the conventional method of querying?
c. What can be done to improve performance of queries without trading off the security of a database system in the cloud?

1.5 Problem Statement

In large outsourced databases, storing data is not problematic, retrieval is the challenge. Raybourn (2013) states that, the efficiency of data retrieval from untrusted servers especially as it relates to security, remains an open challenge.

Before you encrypted your data you were able to send a precise query to the server and to retrieve only the information you needed. But to preserve your privacy, you encrypt the information. Thus, you cannot make the selection on the server anymore because query languages cannot work with encrypted data so computations are limited. Therefore, for each query you have to download the whole database and do the decryption and querying on your own computer before re-encrypting and uploading. Network connectivity and bandwidth problems further make the process tiring and cumbersome. Of course, one can send his/her encryption key to the Cloud Service Provider (CSP) and ask it to do the decryption, but then the end result is the same situation of vulnerability which arises from snooping administrators, hackers, compromised servers.

Available query mechanisms reviewed that allow server-side computation on encrypted data are limited as to the number of queries they can execute. In essence, data retrieval is constrained.

1.6 Motivation

Analysis of cloud storage providers show that most but not all cloud storage providers offer built-in methods to encrypt the data to be stored in the cloud. However, the encryption schemes are mostly not sufficient, as some storage providers encrypt data by using an encryption key generated by and stored at the provider site. This implies that users cannot be sure of the confidentiality of their data (Borgmann *et al.*, 2012).

More so, security measures typically introduce significant computational overhead to the running time of general database operations due to cost of decryption which results in a potential disadvantage of performance degradation of queries. To reduce this overhead, it should be possible to encrypt only sensitive data while keeping insensitive data unencrypted. Thus, only data of interest should be decrypted during query execution.

1.7 Aim and Objectives of the research

This dissertation is aimed at enhancing the privacy and query technique on relational cloud database.

The objectives of this dissertation are to:

a. Design an efficient technique that will enhance the privacy and performance of SQL queries on cloud database.
b. Implement a model that minimizes the running time of client/cloud database queries.
c. Analyze query performance using the conventional querying method versus the proposed system with the Transaction Processing Performance Council (TPC-H) benchmark.

1.8 Research Method

a. Review of related works from various sources.
b. Design the framework of system model.
c. Build the system (implementation).
    i. Design the interface and application in Java

ii. Implement client-side encryption of selective columns of the database using Advanced Encryption Standard(AES) in Cipher Block Chaining(CBC) mode implemented in java.

iii. Use hash map to store mapping of plain text:cipher text as key:value pair.

iv. Upload the SQL database to Google SQL cloud service by using the *"gsutil"* tool to upload data from tpch to google cloud storage buckets with python, then import the tpch data from the buckets to google cloud SQL.

v. Connect the client application to google cloud SQL.

d. Perform the TPCH queries against the database in the cloud.

e. Analyze the traditional vs proposed system

f. Present and discuss result.

1.9 Contribution to Knowledge

The concept of E-Banking, E-Commerce, E-Government, E-Education, Telemedicine as well as the harmonization of databases in various sectors in Nigeria and beyond tends towards the use of such data in information extraction for analysis and decision making. This research harnesses the benefits of cloud technology in conjunction with relational database system to achieve the following:

a. The efficiency of queries on relational cloud database has been enhanced through the design and implementation of *secureSQL* which is able to execute 20 of the 22 TPC-H benchmark queries.

b. Query execution time of client/cloud database queries as compared to the traditional method of querying encrypted databases has been significantly reduced.
c. Privacy of data has been ensured through the use of AES-128 bit which is a cryptographically secure and fast algorithm.

1.10 Organization of Dissertation

Chapter one has introduced the concept of data outsourcing and cloud computing. The problem statements and motivation for the research were outlined as well as the aim, objectives and contribution to knowledge.

Chapter two elucidates on the theoretical concepts of cloud database, data security and reviews relevant literatures.

Chapter three expounds on the architectural framework and design of the secureSQL model.

Chapter four dwells primarily with the implementation of secureSQL, the experiments and discussion of results and observations.

Chapter five concludes with the dissertation summary, conclusion and recommendation of future work.

CHAPTER TWO

REVIEW OF LITERATURE

2.1 Introduction

This chapter addresses the framework of database outsourcing (database in the cloud), data security, query efficiency and benchmarking, which form the background of the research. It goes further to review literature relating to the aforementioned.

2.2 Concept of Cloud Database

A database is an organized collection of data which enables easy access, update and management. It is a structured repository for information with links within the information that help make the data searchable. A Database Management System (DBMS) is a software package with programs that control the creation, maintenance and use of a database. It allows organizations to conveniently develop databases for various applications (Elmasri, 2008).

A cloud database also referred to as Database-as-a-Service (DBaaS or DaaS) is a database that is accessible to clients from the cloud and delivered to users on demand via the Internet from a cloud database provider's servers. It typically runs on a shared cloud computing platform such as Windows Azure, Amazon EC2, GoGrid, Google Cloud SQL, all of which comprise hardware and software (Mateljan *et al.*, 2010). The cloud platform is structured to host multiple outsourced databases by providing database as a specialized service or providing virtual machines to deploy any databases on.

Database services on the cloud are provided with automated features which enable optimized scaling, high availability, multi-tenancy and effective resource allocation. The services have the advantages of increased accessibility, automatic failover and fast

automated recovery from failures, automated on-the-go scaling, load balancing, minimal investment and maintenance of in-house hardware, better performance over the traditional database. Some potential drawbacks include security and privacy issues, potential loss/no access to data in event of disaster, internet outage or bankruptcy of cloud database service provider (Phan, 2013).

The cloud database is constructed by collecting a number of sites also called nodes which are interlinked by a communication network. Every node is a database class. Each class has its own database, terminals, the central processor and their individual local database management system (Donkena and Gannamani, 2012).

2.3 Types of Cloud Database

Cloud databases are categorized based on the model. Some are SQL based and some use the NoSQL data model.

2.3.1 SQL Databases

Structured Query Language (SQL) was developed in the 1970's by IBM when Edgar Codd introduced the relational data model. It is a standard query language for relational database management systems (RDBMS) (Codd, 1970).

In the relational model, data are organized into relations, each represented by a table comprising rows and columns. It also has a key, which is used to map data to other relations. SQL is used to make queries to the database such as creating, reading, updating and deleting data. SQL supports indexing mechanism to speed up reading operations, or creating views which can join data from multiple tables and other features for database optimization and maintenance (Elmasri, 2008).

One important attribute of SQL databases is that they follow the Atomicity, Consistency, Isolation and Durability (ACID) rules (Codd, 1970).

a. Atomicity: A transaction is a logical unit of work which must be either completed with all of its data modifications, or none of them is performed that is, transactions are all-or-nothing
b. Consistency: Before and after a transaction, all data must be left in a stable state. Roll-back occurs in event of failure.
c. Isolation: Modifications of data performed by a transaction must be independent of another transaction (without interference). Unless this happens, the outcome of a transaction may be erroneous
d. Durability: When the transaction is completed, effects of the modifications performed by the transaction must be permanent in the system (committed). Committed transactions cannot be lost. Track of changes are kept in logs to ensure reliability of data at any point in time.

Although relational databases focus on integrity, the main challenge is scaling for constantly growing data. Examples of relational databases include MySQL, Oracle, SQL Server.

2.3.2 NoSQL DATABASES

Not only SQL (NoSQL) is a DBMS that is also used on the cloud. NoSQL does not divide data into relations (non-relational), nor does it use SQL to communicate with the database. It has fewer limitations on data structure thus supports more data types. NoSQL databases are schema free and designed to store data that is simple, schema-less, object-oriented or unstructured. According to the CAP (Consistency, Availability, Partition Tolerance) theorem, NoSQL databases cannot guarantee the three properties at

the same time. They also exhibit BASE (Basically Available, Soft state, Eventual consistency) characteristics (Donkena and Gannamani, 2012).

Types of NoSQL databases include: document stores, graph databases, key-value stores and column family stores.

2.3.3 SQL vs NoSQL

Table 2.1 shows the differences between SQL databases and NoSQL databases in other to shed more light on the choice of relational database in this research (Elmasri, 2008).

Table 2.1: SQL vs NoSQL

S/N	SQL	NoSQL
1	Relational Model (Structured schema)	Non-relational data (schema-less, unstructured, simpler)
2	Tables	Key-value, Document, Graph, Column family stores
3	ACID	BASE
4	Focus on Consistency	Focus on Availability, Performance and Scalability
5	Single Server	Cluster of Servers (Horizontal scalability)
6	SQL Query	Simpler and different interface (API)

2.4 Data Security

Migrating databases to the cloud environment involves a number of security challenges which encompass data confidentiality, integrity and availability. Security challenges also entail that data must be secured while at rest, on transit and in use. Also, access to data must be controlled. Access control (authentication and authorization) has proved to be useful as long as the data is accessed through the intended system interfaces.

However, access control is useless if attacker simply gains direct access to the raw data in the database bypassing traditional mechanisms (Sion, 2008).

2.4.1 Confidentiality

Confidentiality is a set of rules that bound access or location restriction on certain types of information. It must be ensured that user's private information is not accessed by anyone including application, platform, CPU and physical memory (Sharma and Trivedi, 2014). If database as a service is to be successful then data or information outsourced must be secured from both authorized and unauthorized parties. Potentially untrusted servers should be able to process queries on encrypted data on behalf of clients without compromising or breaching confidentiality (Sion, 2008).

2.4.2 Availability

Availability means the extent to which system resources are accessible and usable to individual users or organizations (Sakhi, 2012). Uninterrupted service to customers is a main concern in cloud computing. Information should be available to access/modify by authorized persons.

The denial-of-service attack (DoS) and distributed DoS (DDoS) attacks are popular online attacks which affect the availability of online servers and the data therein. Other threats include equipment failures and natural disasters. In such cases, most of the downtime is unplanned. Nevertheless, this could cause untold consequences on many applications and users (Sakhi, 2012).

Incidents of such attacks include:

Gmail(one-day outage in mid-October 2008); Amazon S3(over seven-hour downtime on July 20, 2008); Google's blogger outage(over 48 hours downtime on May 12, 2011);

Gmail's reset problem(accidentally resetting gmail accounts on February 27, 2011) (Kumar and Chelikani, 2011).

2.4.3 Integrity

Integrity means that the content of the communicated data is assured to be free from any type of modification between the end points (sender and receiver). It ensures "completeness" and "wholeness" by protecting data from un-authorized alteration (Sharma and Trivedi, 2014). Data integrity proves the validity, consistency and regularity of data by ensuring that it is not accidentally or maliciously deleted or changed. Data should be secured in a persistent way that guarantees retrieval in the same layout to escape all possibilities of data corruption and data crash.

2.4.4 Access Control

Loss of access control is another major security threat for outsourced data. This is because client loses physical, logical and personal control over the data. Recent research (Sharma and Trivedi, 2014) shows that access control threats emanate from internal client's employees and cloud service provider's employees. Therefore, proper access control and monitoring procedures for cloud database administrators are crucial to ensure security of sensitive data.

2.5 Concept of Database Security

Database encryption refers to the use of encryption techniques to transform a plain text database into an encrypted form called a cipher, thus making it unreadable to anyone except those who possess the knowledge of the decryption key(s) (Bouganim and Guo, 2009).

Encryption for data on transit as well as data at rest is a sure way of ensuring confidentiality of sensitive data. Encryption support for data on transit is offered by most Cloud Service Providers using Secure Socket Layer protocol (SSL) and Transport Layer Security (TSL) protocol. However, very few offer encryption for data at rest (Sakhi, 2012). Encryption options available could be client-side or server-side which employ techniques that have been tested and trusted or custom built encryption solutions.

Conventional database security solutions and mechanisms are divided into three layers: physical security, operating system security and database management system (DBMS) security (Fernandez *et al.*, 1980). These layers are not sufficient to guarantee the security of a database when database content is kept in a clear-text, readable form. Many enterprises are addressing this challenge of private data exposure, especially in the banking, financial, insurance, government, and healthcare industries, by adopting database encryption. Database-level encryption offers some measure of data protection by ensuring that only authorized users can see the data, and it protects database backups in case of loss, theft, or other compromise of backup media (Shmueli *et al.*, 2009).

Generally, database encryption should meet the requirements for data security (without the encryption key it is impossible to retrieve any information about the plain text from the cipher text), high performance (limit both encryption size and speed overhead) and detection of unauthorized modifications (Min-Shiang and Wei-Pang, 1997). The client has a combination of sensitive and non-sensitive data stored in a database at the server. The server's added responsibility is to protect the client's sensitive data ensuring its confidentiality, integrity and availability.

The granularity of encryption is another dimension that is considered. It can be a field, row or page. The field is considered the best choice because it minimizes the number of bytes encrypted (Mattsson, 2005).

A straight forward approach of wrapping a layer of encryption entirely round the data which is adopted in the Database Outsourcing scenario,(Damiani *et al.* 2003; Hacigumus *et al.* 2002a) is based on the assumption that all the data are equally sensitive. This is overkill because in most situations, the sensitivity is the association between data (Ciriani *et al.*, 2010). Hence, there is need for the proposed encryption of selective fields.

2.5.1 Levels of Encryption

The various levels of encryption explained, are depicted diagrammatically in Figure 2.1.

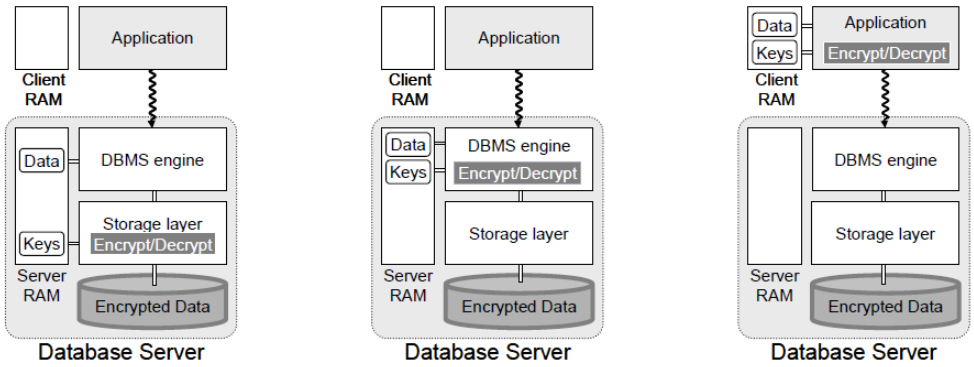
a. **Storage-level encryption** encrypts data in the storage subsystem and thus protects the data at rest (e.g., from storage media theft). It is well suited for encrypting files or entire directories in an operating system context. From a database perspective, storage-level encryption has the advantage to be transparent, thus avoiding any changes to existing applications. On the other side, since the storage subsystem has no knowledge of database objects and structure, the encryption strategy cannot be related with user privileges (e.g., using distinct encryption keys for distinct users), nor to data sensitivity. Thus, selective encryption – that is, encrypting only portions of the database in order to decrease the encryption overhead – is limited to the file granularity (Bouganim and Guo, 2009).

b. **Database-level encryption** allows securing the data as it is inserted to, or retrieved from the database. The encryption strategy can thus be part of the database design and can be related with data sensitivity and/or user privileges.

   Selective encryption is possible and can be done at various granularities, such as table (relation), column (field or attribute), row (tuple). It can even be related with some logical conditions (e.g., encrypt salaries greater than ₦50,000/month). Depending on the level of integration of the encryption feature and the DBMS, the encryption process may incur some change to applications (Bouganim and Guo, 2009).

c. **Application-level encryption** moves the encryption/decryption process to the applications that generate the data. Encryption is thus performed within the application that introduces the data into the system. The data is sent encrypted, thus naturally stored and retrieved encrypted, to be finally decrypted within the application. This approach has the benefit to separate encryption keys from the encrypted data stored in the database since the keys never have to leave the application side (Bouganim and Guo, 2009). However, applications need to be modified to adopt this solution.

   In terms of granularity and key management, application-level encryption offers the highest flexibility since the encryption granularity and the encryption keys can be chosen depending on application logic.



a) Storage-level encryption b) Database-level encryption c) Application-level encryption

Figure 2.1: Levels of Encryption (Bouganim and Guo, 2009)

2.6 Data Security Concerns

Security of data in a database has consequences which cannot be overlooked. This section looks at some data security concerns.

2.6.1 Encryption Overhead

The process of converting a plain text to a cipher text incurs a huge cost of system resources, depending on the encryption algorithm used and the method of implementation. It is desirable that this overhead be reduced to the minimum with the right choice of algorithm. Selective encryption, as in the case of mixed database (Mykletun and Tsudik, 2006) provides de facto access control while minimizing the encryption overhead.

2.6.2 Handling Encryption/Decryption Keys

According to Mattsson (2005), certain criteria must be considered in handling encryption/decryption keys. They are:

a. Cryptographic Access Control: Encrypt objects from different security groups with different keys to restrict user.

b. Secure Key Storage: Encryption keys should be kept securely.

c. Key Recovery: It should be possible to recover encryption keys whenever needed.

2.6.3 Client Side Encryption

In the Database-as-a-Service (DaaS) architecture, one fundamental problem (beside performance degradation due to remote access to data) is data privacy. If an encryption scheme is defined under the assumption that server is not trusted then a question of how a query is evaluated arises, if a server has no access to the encryption keys. (Song *et al.*, 2000). Client side encryption is introduced to counter this challenge.

2.7 Terms used in Cryptography

Some of the more commonly used terms in cryptography as defined by Cowie and Irwin (2009) are as follows:

a. Plain Text: A text or message that is in a clear readable form. For example, Alice as a person wishes to send "Hi, how are you?" message to a person Bob, "Hi, how are you?" is referred to as plain text.

b. Cipher Text: A text or message which has been converted into incomprehensible code using an algorithm. For example "ib%ipvbufzpv@" is a cipher text produced for plain text "Hi, how are you?".

c. Cipher: This is a cryptographic algorithm or mathematical function used for encryption and decryption.

d. Encryption: Converting plain text to cipher text is referred to as encryption. It requires an encryption algorithm and a key.

e. Decryption: Converting cipher text to plain text is referred to as decryption. This also requires a decryption algorithm and a key.

f. Key: Combination of numeric or alpha numeric text or special symbol is referred to as a key. Encryption or decryption key plays a vital role in cryptography because encryption/decryption algorithm directly depends on it.

g. Cryptography: The science of encryption and decryption.

h. Cryptanalysis: The science of breaking encryption algorithm.

i. Cryptology: study of cryptography and cryptanalysis.

j. Cryptosystem: This is an algorithm with all its possible plain text, cipher text and keys.

k. Cryptoanalyst: Someone who attempts to break an encryption algorithm.

2.8 Types of Ciphers

There are two basic types of ciphers. They can be block ciphers or stream ciphers. Block ciphers operate on blocks of plaintext usually of 64 bits but sometimes longer while Stream ciphers operate on streams of plaintext and ciphertext one bit or byte (sometimes even one 32-bit word) at a time (Schneier, 1995).

2.8.1 Block Ciphers

Block ciphers encrypt plaintext in units of blocks of fixed length and likewise decrypt cipher text in units of blocks. In a block cipher, a block of symbols from A is operated on jointly by the encryption algorithm, so that in general one may view a block cipher as a nonsingular mapping from the set of plaintext n-tuples A^n into the set of cipher n-tuples C^n. For cryptosystems which use the same key repeatedly, block ciphers are cryptographically stronger than stream ciphers. Consequently, most contemporary cryptosystems are block ciphers, although one-time key systems are used in applications where the very highest security is required (Simmons, 1979). Figure 2.2 is a structure of a block cipher.

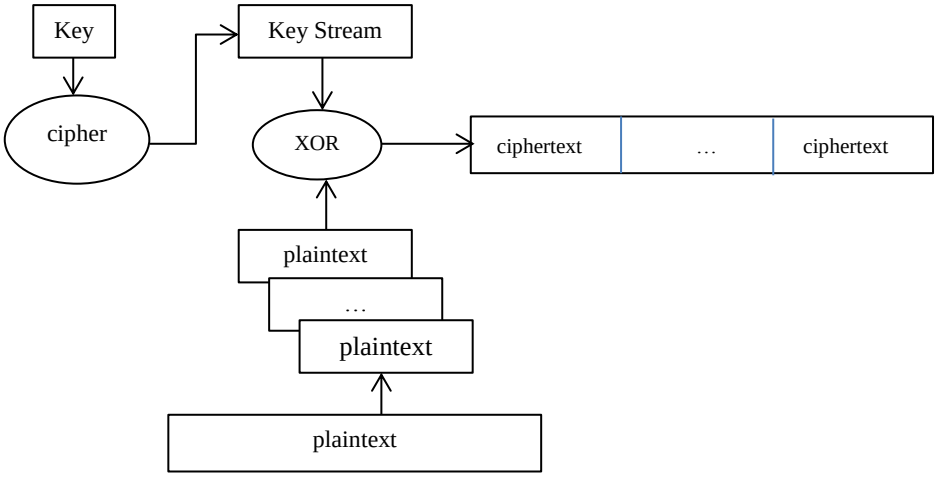


Figure 2.2: Block Cipher Operation (Lafourcade, 2008)

Large messages are encrypted by partitioning into *n*-bit blocks then a cryptographic mode of operation is chosen. A cryptographic mode usually combines the basic cipher, some sort of feedback, and some simple operations. The operations are simple because the security is a function of the underlying cipher and not the mode. Even more strongly, the cipher mode should not compromise the security of the underlying algorithm (Schneier, 1995).

a. Block Cipher Modes of Operation

i. Electronic Code Book (ECB)

ii. Cipher Block Chaining (CBC)

iii. Cipher Feedback Mode (CFB)

iv. Output Feedback (OFB)

i. Electronic Codebook Book (ECB)

Each block of the same length is encrypted separately using the same key K as shown in Figure 2.3. Decryption is in reverse as shown in Figure 2.4. In this mode, only the block in which the flipped bit is contained is changed. Other blocks are not affected. The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks; thus, it does not hide data patterns well and it doesn't provide data confidentiality.

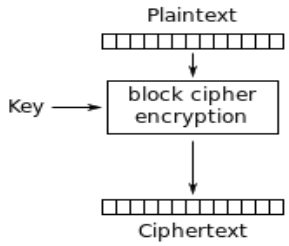


Figure 2.3: ECB Mode Encryption (Lafourcade, 2008)

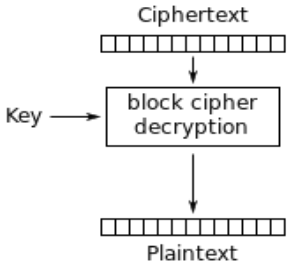


Figure 2.4: ECB Mode Decryption (Lafourcade, 2008)

ii. Cipher Block Chaining (CBC)

IBM invented the cipher-block chaining (CBC) mode of operation in 1976. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, a random initialization vector must be used in the first block. Transmission errors in one ciphertext block completely destroy the plaintext and change the decryption of the next block. Figure 2.5 shows the encryption process while 2.6 shows the decryption process. The mathematical formula for CBC encryption is:

$$C_i = E_K(P_i \oplus C_{i-1}), C_0 = IV \quad (2.1)$$

while the mathematical formula for CBC decryption is:

$$P_i = D_K(C_i) \oplus C_{i-1}, C_0 = IV \quad (2.2)$$

where C_i is the cipher text; E_k is the encryption key; P_i is the plain text; C_0 is the dummy initial ciphertext block also known as Initialization Vector (IV) and D_k is the decryption key.

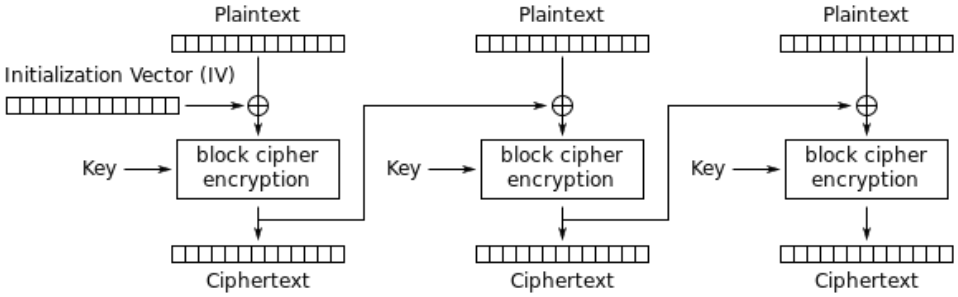


Figure 2.5: CBC Mode Encryption (Lafourcade, 2008)

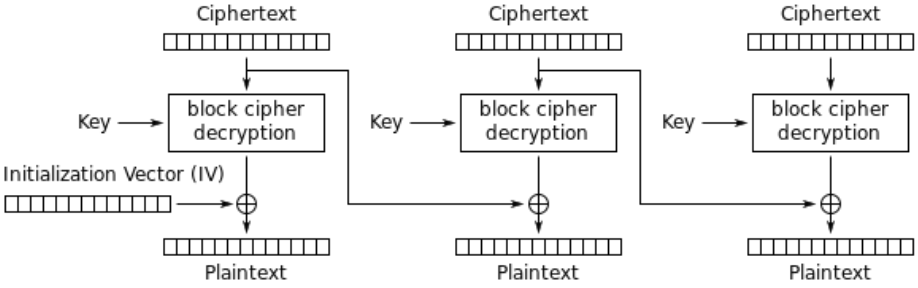


Figure 2.6: CBC Mode Decryption (Lafourcade, 2008)

iii. Cipher FeedBack (CFB)

The cipher feedback (CFB) mode, a close relative of CBC, makes a block cipher into a self-synchronizing stream cipher. Transmission errors completely destroy the following block, but only affect the wrong bits in the current block. CFB decryption is almost identical to CBC encryption performed in reverse. Figure 2.7 depicts the encryption process while Figure 2.8 depicts the decryption process.

The encryption formula is:

$$C_i = E_K(C_{i-1}) \oplus P_i \quad (2.3)$$

The decryption formula is:

$$P_i = E_K(C_{i-1}) \oplus C_i C_0 = IV \quad (2.4)$$

where C_i is the cipher text; E_k is the encryption key; P_i is the plain text; C_0 is the dummy initial ciphertext block, IV is the Initialization Vector and D_k is the decryption key.

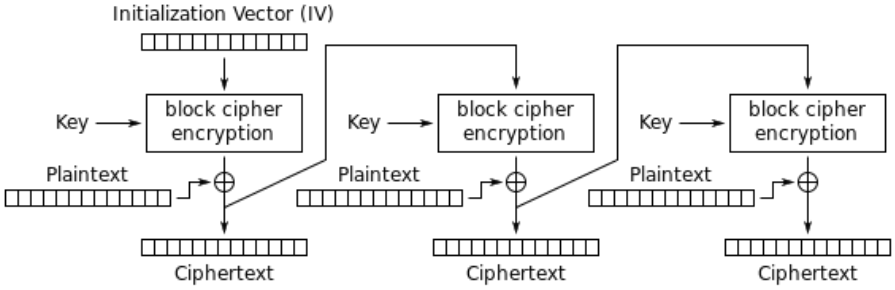


Figure 2.7: CFB Mode Encryption (Lafourcade, 2008)

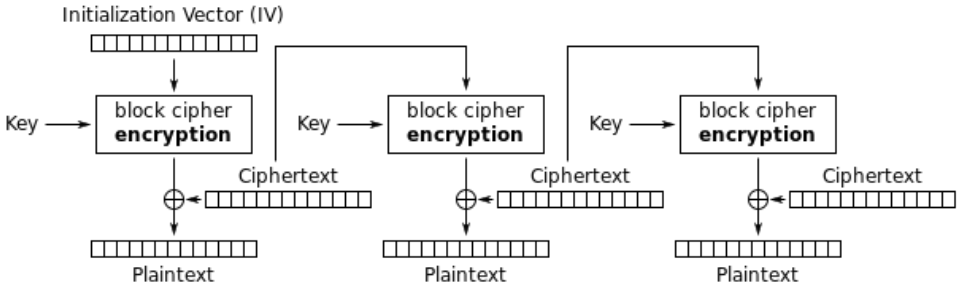


Figure 2.8: CFB Mode Decryption (Lafourcade, 2008)

iv. Output FeedBack (OFB)

The output feedback creates a pseudo random stream independent of the plaintext. Different pseudo random streams are obtained by starting with a different random IV for every message. Transmission errors only affect the wrong bits. Because of the symmetry of the XOR operation, encryption and decryption are exactly the same. This can be seen in Figure 2.9 for encryption and Figure 2.10 for decryption.

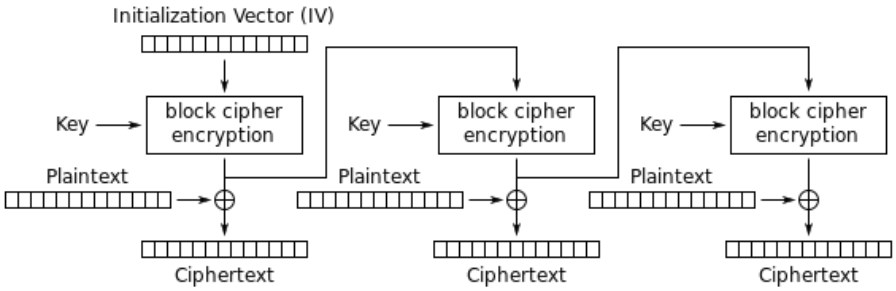


Figure 2.9: OFB Mode Encryption (Lafourcade, 2008)

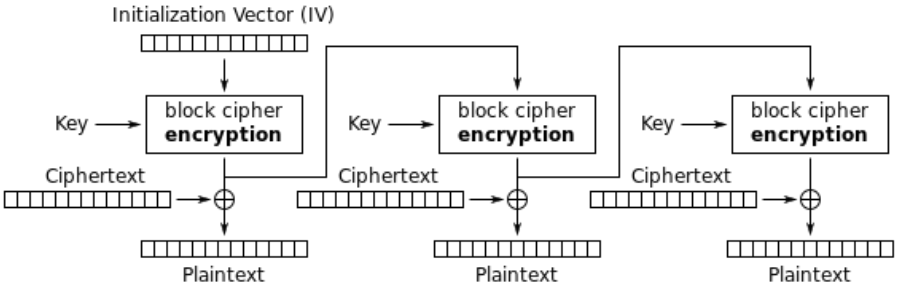


Figure 2.10: OFB Mode Decryption (Lafourcade, 2008)

2.8.2 Stream Ciphers

Stream ciphers operate on streams of plaintext and ciphertext one bit or byte (sometimes even one 32-bit word) at a time. They operate on the plaintext symbol by symbol to produce a sequence of cipher symbols from an alphabet C. Stream ciphers encrypt plaintext in one stream and decrypt cipher text likewise. A keystream generator (sometimes called a running-key generator) outputs a stream of bits: $k1$, $k2$, $k3$,..., ki. This keystream (sometimes called a running key) is XORed with a stream of plaintext bits, $p1$, $p2$, $p3$,..., pi, to produce the stream of ciphertext bits $c1$, $c2$, $c3$ …ci as depicted in Figure 2.11.

$$ci = pi \bullet ki \tag{2.5}$$

At the decryption end, the ciphertext bits are XORed with an identical keystream to recover the plaintext bits.

$$pi = ci \bullet ki \tag{2.6}$$

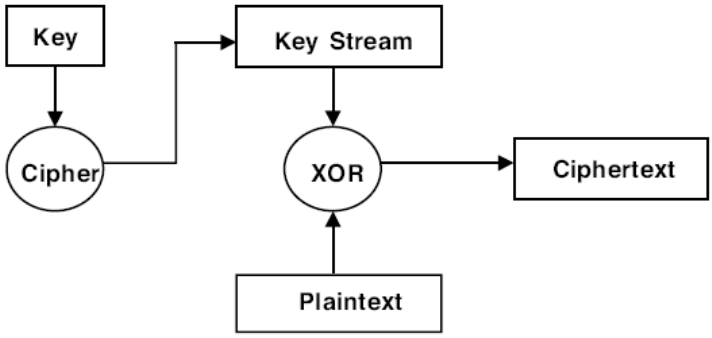


Figure 2.11: Stream Cipher Operation (Tantawy, 2010)

2.9 Symmetric Encryption

Ciphers are also categorized based on the key distribution and use. Thus, they can be symmetric (private key) or asymmetric (public key). Symmetric key encryption which is also known as single-key, secret-key, and private-key or one-key encryption has been in use since ancient times and has a wide variety of different implementations ranging from simple substitution ciphers such as Caesars Cipher to complex and supposedly "mathematically unbreakable" algorithms such as AES. Symmetric key encryption makes use of a single key that must be kept secret, this key is used for both the encryption and decryption of messages to be sent or stored (Schneier, 1995). Each secret key cryptography algorithm typically works in two phases:

a. key set-up phase
b. ciphering or encrypt and decrypt phase.

The two requirements for secure use of symmetric encryption are:

a. Strong encryption algorithm
b. Secret key known only to sender / receiver

Encryption is mathematically represented as:

$$Y = E_K(X) \quad (2.7)$$

where E_K is the encryption key applied as a function to plaintext X to give an output of ciphertext Y. Decryption on the other hand, is represented as:

$$X = D_K(Y) \quad (2.8)$$

where the decryption key D_K is applied as a function to the ciphertext Y, to produce the plaintext X (Schneier, 1995).

2.9.1 Advantages of Symmetric Encryption

Symmetric encryption has advantages and disadvantages. Some of the advantages are highlighted by Townsend (2009) as follows:

a. **Simple:** This type of encryption is easy to carry out. All users have to do is specify and share the secret key and then begin to encrypt and decrypt messages.
b. **Encrypt and decrypt your own files:** If you use encryption for messages or files which you alone intend to access, there is no need to create different keys. Single-key encryption is best for this.
c. **Fast:** Symmetric key encryption is much faster than asymmetric key encryption and this helps solve performance issues.
d. **Uses less computer resources:** Single-key encryption does not require a lot of computer resources when compared to public key encryption.
e. **Prevents widespread message security compromise:** A different secret key is used for communication with every different party. If a key is compromised, only the messages between a particular pair of sender and receiver are affected. Communications with other people are still secure.

Symmetric Algorithms are used for securing tapes, digital streaming and for protecting data at rest. Examples of symmetric key algorithms by Townsend (2009) are as follows:

a. Data Encryption Standard(DES) (56bits)
b. Triple DES (3DES) (168 bits)
c. Advanced Encryption Standard (AES)
d. International Data Encryption Algorithm (IDEA) (128 bits)
e. Rivets Cipher 4 (RC4) (variable length key)

2.10 Advanced Encryption Standard

The AES accepted candidate, Rijndael, was designed by Joan Daemen and Vincent Rijmen from Belgium and was published in 1998 and approved for use by US Government in 2002 (Lafourcade, 2008). It is an iterated block cipher allowing for variable key length and allows for a choice from a number of different block sizes. Rijndael supports block sizes of 128-bits, 192-bits and 256-bits. It is byte orientated, compared to the bit orientated nature of DES. It uses various substitutions and transpositions plus key scheduling, in different rounds. The number of rounds or iterations applied is dependent on the sizes of the block and the key used. For example if the block size is 128-bits and let *m* be the size of key and *r* the number of rounds is given by $r = k/32+6$.

At the start a 128-bit block of plain-text is used as the initial state. This initial state will be passed through a number of key-dependent transformations, finally returning a 128-bit block of cipher-text. A state is treated as a 4x4 matrix, where $A_{i,j}$ will represent a single byte with $0 \leq i, j \leq 3$, i referring to the rows and j referring to the columns. For example $A_{0,0}$ is the first byte and $A_{1,0}$ is the 5^{th} byte (Lafourcade, 2008).

The algorithm is believed to be secure, having undergone a lot of cryptanalysis (Townsend, 2009 and Lafourcade, 2008). The only attacks are based on side channel analysis that is, attacking implementations that inadvertently leak information about the key. It also has great speed at both hardware and software implementations

Rijndael makes use of four basics operators to allow for transformation from one state, say A = ($A_{i,j}$), to another state. The set of operators used by Rijndael are outlined.

a. Byte Substitution

This is a non-linear permutation where each byte is replaced with another according to a lookup table. It operates on each byte in the current state independently, allowing for parallelism. In this phase 8-bytes of a 16-byte phase are taken and multiplied by an 8 x 8 matrix, that is matrix multiplication of 8 x 8 matrix by 8x1 column vector resulting in a 8x1 column vector. This can be efficiently implemented by making use of a 256-bit lookup table or an S-box. Figure 2.12 shows the byte substitution process.

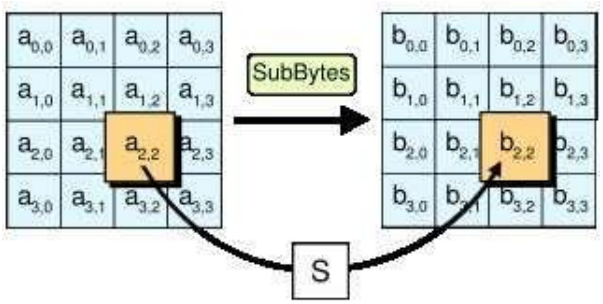


Figure 2.12: Byte Substitution (Lafourcade, 2008)

b. Shift Rows

This is a cyclic shift of the bytes in a state. This could be represented as $B_{i,j}$ = $A_{i,(j+1)}$mod4. The first row will undergo no changes, however the second row will shift one column, the third row shifts two columns and the fourth row will shift three columns as shown in Figure 2.13.

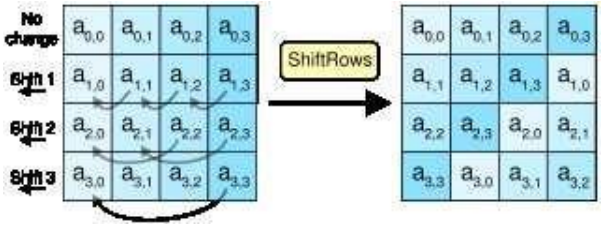


Figure 2.13: Shift Rows (Lafourcade, 2008)

c. Mix columns

This is a mixing operation which operates on the columns of the state, combining the four bytes in each column. Each of the columns A_i undergoes a linear transformation. A transformation is applied to one column at a time and is equivalent to multiplying the columns contents by 4 x 4 matrix, that is, matrix multiplication of 4 x 4 matrix with 4 x 1 column matrix containing the columns values. Figure 2.14 shows the mix operation (Lafourcade, 2008).

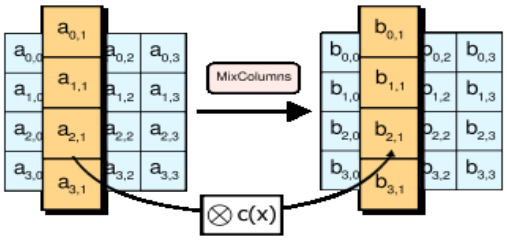


Figure 2.14: Mix Columns (Lafourcade, 2008)

d. Round Key Addition

For every round, a round key, RK, is generated from the cipher key via the key scheduling function. The round key is the same length as the encryption block and is represented in 4 x 4 matrix similar to the representation of the plain-text. Each byte of the state is combined with the round key using the XOR as displayed in Figure 2.15.

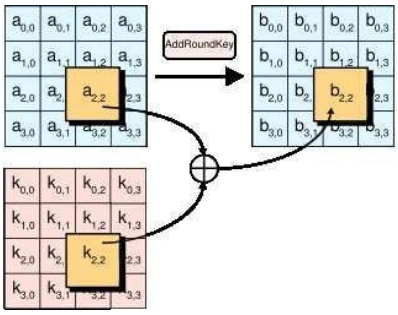


Figure 2.15: Round Key Addition (Lafourcade, 2008)

e. Key Scheduler

The key scheduler consists of two sections, the key expansion function and round key selection. A key expansion function is used to expand the cipher key to produce the required number of bits for the round keys. The required number of key bits is equal to N(R+1) where N is the required block size and R is the number of rounds to be completed. In round key selection, if a block size of 128-bits is assumed, the most significant 128-bits are used for the first round, then the next most significant 128-bits are used for the next round and the selection continues in like manner (Lafourcade, 2008).

2.11 Choice of Encryption Granularity and Algorithm

In principle, different granularity choices are possible for database encryption, such as encrypting at the level of whole tables, attributes, tuples and elements. Encrypting at the level of tables and attributes implies that the whole table or attribute involved in a query would always be returned, providing no means for selecting the data of interest and leaving to the client the burden of query execution on a possibly huge amount of data. Nevertheless, attribute level encryption supports more SQL statements than other options and it minimizes the number of bytes encrypted (Mattsson, 2005). Here, only columns containing important data are encrypted which result in lower CPU load compared with the whole database level encryption. If a column is used as a primary

key or used in comparison clauses (WHERE clauses, JOIN conditions) the database will have to decrypt the whole column to perform operations involving those columns. Supporting encryption at the finest possible level of single cells is inapplicable because it would severely affect performance since the client would be required to execute a very large number of decrypt operations to interpret result of queries (Hacigumus *et al.*, 2002a).

Even having adopted strong algorithms, such as AES, the cipher text could still disclose plain text information if an inappropriate mode is chosen. For example, if encryption algorithm is implemented in electronic codebook mode (ECB), identical plaintext blocks are encrypted into identical cipher text blocks, thus disclosing repetitive patterns. In database context, repetitive pattern are common as many records could have same attribute values. The 128-bit key in the Cipher Block Chaining (CBC) mode is strong enough to withstand cryptanalysis, with less consumption of computing resources and less performance overhead (Townsend, 2009). For most purposes, 128-bit symmetric keys are sufficient (Bellovin, 2009). Since the size of a token determines the efficiency of the algorithm, the smaller the token, the more efficient (Townsend, 2009). The CBC mode is a non-deterministic encryption scheme. This means that encryption of same value two times will give different cipher texts with a very high probability. It is achieved by using randomization during encryption of data.

2.12 Query Efficiency

Database performance plays a vital role in the overall functioning of the database due to the fact that informed decisions are taken by Enterprises based on analysis of databases. This performance can be measured basically in terms of the efficiency of data retrieval.

According to Raybourn (2013), efficiency refers to the time required for an algorithm to perform any of its functions. The functions in this context are the encryption/decryption time for a given data and the time required for query execution over a data set.

Efficiency relates to the resource usage of a scheme which can be expressed in a variety of ways. Common ways to measure efficiency include the size of the cipher text, size of the token (key), number and type of computations during encryption and decryption. For an outsourced database, the focus is to minimize the cost of the encryption algorithm with respect to time (Townsend, 2009).

2.13 Hash Map

Hash Map is derived from hash table. It is formed by transforming a range of key values to a range of array index values by using a hash function. A hash map could be implemented using Closed hashing (open addressing) or Open hashing (closed addressing). The latter will be used in the implementation of this work. Open hashing is a large array of O(n) elements, called buckets. Each bucket is a set containing O(1) elements and the elements of the set as a whole are partitioned among all the buckets. This ensures optimal performance especially when the hash function and resizing threshold are tuned to determine the load factor $\propto$ (TechyHelp, 2014).

To determine the bucket that stores the element, use a hash function $h(e)$ that given a set element *e* returns the index of a bucket that should contain that element. Therefore,

$$h(n)=n \bmod m \tag{2.9}$$

for *n* number of elements (keys) and *m* number of buckets (TechyHelp, 2014).

In a well dimensioned hash map, the map size is proportional to the number of entries O(*n*) and the average cost (number of instructions) for each lookup, and the lookup is

independent of the number of elements stored in the table. The time complexity denoted by big O for average case is O(1) for the search, insert and delete operations (TechyHelp, 2014).

If there are *n* elements in the set and the bucket array is length *m*, then it is expected that the load factor, $\propto = {}^{n}/_{m}$ element per bucket. It is ensured that the load factor $\propto$ never exceeds $\propto_{max}$ where max = 0.75 according to java's hashmap class recommendation. So, all operations are O(1) on average and collisions are avoided. Hashing randomizes the data order and provides efficient access to information based on the key. Using a hash table with a map interface standardizes the time it takes to perform lookup operations (TechyHelp, 2014, June 29).

2.14 Benchmarking

Benchmarking is widely used for evaluating computer systems, and benchmarks exist for a variety of levels of abstraction, from the CPU, to the database software, to complete enterprise systems. Gray (1993) identified four criteria for a successful benchmark: relevance to an application domain, portability to allow benchmarking of different systems, scalability to support benchmarking large systems, and simplicity so the results are understandable.

2.15 Review of Related Literature

The Database-as-a-Service (DaaS) model was first introduced by Hacigumus *et al.,* (2002a). In his work, protecting the database records of a client from an untrusted database service provider was considered, by developing a model: NetDB2.

A number of research activities have been conducted about the problems of securing outsourced data and performing computations on it efficiently, as well as balancing the

trade-off between the security and performance. Based on the scope of this research, the related literature is categorized into three:

a. Querying encrypted data in an outsourced database.
b. Privacy of data
c. Balancing query efficiency and security.

a. Querying encrypted data in an outsourced database

Querying data in encrypted form on an outsourced database focuses on devising techniques that allow computations on data that is encrypted using conventional encryption algorithms or the design and use of customized cryptographic schemes such as order-preserving or prefix-preserving encryption that allow computations to be performed on the data and they are applicable to numerical data or string data (Elmasri, 2008); but they do not provide a strong protection as they tend to leak a lot of information (Jacob, 2010).

Hacigumus *et al.* (2002b) explored techniques to execute SQL queries over encrypted data by processing as much of the query as possible at the server side without decryption and the remainder at the client side. This was done by using an algebraic framework (bucketization) for query rewriting to split (partition) the query for computation at both the server and client side thus minimizing computation at the client side by using metadata as index. Although privacy from the Service Provider was achieved with reasonable overhead, the technique employed could not handle aggregate functions like SUM, COUNT, AVG, MIN and MAX on encrypted numerical data. It was inapplicable for string data as well because the response of a query generated lots of additional data leading to high computational and communication overhead.

To overcome the limitations in the work of Hacigumus *et al.* (2002b), Hacigumus *et al.* (2004) used an encryption scheme called privacy homomorphism which enables arithmetic operations on encrypted data, to support aggregation functions on outsourced data. Mykletun and Tsudik (2006) demonstrated that the homomorphic encryption scheme in Hacigumus *et. al.* (2004) could be broken by a cipher-text only attack and proposed another solution to support aggregation functions in the DaaS model. The solution entailed pre-computation of aggregate values before encryption and storing them separately, such that an aggregate query returns the pre-computed values. This minimized client-server computation. The concept of mixed databases (some attributes encrypted and some not) was also introduced.

Brinkman (2007) in his Ph.D thesis, addressed the problem of securing data stored on an untrusted server which is honest but curious by using a trapdoor mechanism to implement keyword search. An Xtensible Markup Language (XML) text is encrypted in such a way that it is possible to search for a certain word. The server is given a key that is specific for that particular word. With this key the server is able to scan the encrypted text and find occurrences of the word. The server learnt only the locations of the word, if it occured at all.

Generally, searchable encryption techniques such as the *probabilistic scheme* proposed by Song *et. al.* (2000) or the *fully homomorphic encryption* by Gentry (2009) tend to be computationally expensive and unable to scale as the size of the data increases.

b. Privacy of Data in Outsourced Databases

Certain literature focused on the privacy of outsourced data. Aggarwal *et al*. (2005) based his research on the notion of fragmentation and encryption to preserve the privacy

of data, by concealing the associations between attributes of a relation based on predefined security constraints. He proposed outsourcing of data management to two untrusted, non-communicating servers while preserving data privacy by partitioning data in a relation so that the contents at any one server are guaranteed not to breach data privacy. Query execution is done by transmitting appropriate sub-queries to each database, then piecing together the results at the client side. Aggarwal's assumption that the separate servers cannot communicate is a limitation for any practical situation. Also, there is significant communication and computation overhead since for executing each query, the shares of the database must be retrieved to reconstruct the database.

To modify Aggarwal *et al.*'s (2005) solution, Ciriani *et al*. (2009) proposed to store one fragment on the owner side and the other at the server. To further the work, Ciriani *et al*. (2010), proposed a solution to ensure privacy and minimize the query execution cost by employing fragmentation criteria to split the information into different fragments in such a way to break the sensitive associations represented through confidentiality constraints and to minimize the amount of information represented only in encrypted format. The resulting fragments may be stored at the same server or at different servers. Finally, the encryption key is given to the authorized users needing to access the information. The work dwells more on the construction of heuristic algorithms that guide the fragmentation of data. In a cloud setting, data is distributed across several servers such that fragmentation may not be required.

c. Balance of Query Efficiency and Security

Since the aim of this research is to ensure that query performance and efficiency are balanced without a tradeoff to the security, literatures that tend towards this direction were also reviewed. Damiani *et al.* (2003) proposed techniques for the execution of

queries on untrusted servers while preserving efficiency, by employing a hash-based method which uses an encrypted invertible index for selection queries, so that an attacker has no certainty that two records are equal when they have the same index. He also adapted the B+-tree structures made over plain-text data, encrypted by the data owner, and saved on the untrusted server to execute range queries. One limitation to Damiani's work is that it needed to perform many queries that equal to height of the B+ tree which is stored at the server to retrieve results. This imposes high computation and communication overhead on the client-side.

Popa *et al.* (2011), in their work: CryptDB implemented an approach that is built on top of the existing relational database management systems (RDBMS) and protects the privacy of database records in the cloud. It uses the concept of "onions". An onion consists of multiple layers of encryption schemes on a single data value before storing on the server. The removal of onion layer led to performance problems. Any query which needed to remove a layer of onion ran slowly when it was executed for the first time because of the time taken to remove the onion layer.

Alhanjouri and Al Derawi (2012) proposed a mechanism to query encrypted data and make a tradeoff between the performance and the security by using the hash map data structure stored in the metadata component of the layer. The hash map stores the mapping between the plain text and the encrypted text as key:value pair. The work is implemented as a layer above the DBMS to manage the query process. The layer is placed on the same place with the database management system which is a limitation in the outsourced database scenario. Also, the encryption/decryption key is kept on the server side.

Sharma *et al*. (2013) did a research that aimed to strike a balance between security and efficient query processing on encrypted databases by allowing users to query directly over the encrypted column without decrypting all the records thereby improving the performance of the system. They employed a technique which suggested two tables for a single main table. One table stored sensitive columns in encrypted format and the other table stored the encryption/decryption keys in encrypted format along with the content of the first table in plain text thereby hiding the relationship. Data retrieval was limited to less than 40% of the total. Also, the method is not suitable for the cloud environment.

In his dissertation, Kaul (2013) introduced a framework for solving the problem of executing queries efficiently at the cloud resident server while maintaining data security. PhantomDB maintained data security by using the onion method of Popa *et al.* (2011) which were carefully chosen to allow efficient query processing at the server. PhantomDB introduced the concept of Arithmetic Engine and Round Communication, which allowed it to support all the standard SQL constructs with the exception of similarity operators. A hybrid storage model which takes unique features of PhantomDB into consideration while deciding the layout of relational data on disk was introduced to guarantee efficiency. A framework that was efficient without altering the structure of the DBMS will be preferable.

It is pertinent to note that the problems considered in this research and the solutions proposed, differ from existing approaches. Since this research focuses on enhancing the performance and efficiency of queries on cloud databases with respect to time, the encryption scheme considered balances the tradeoff of security and performance through the use of selective attribute based encryption to secure sensitive data thus

minimizing the amount of information represented in encrypted format. The preservation of data privacy is achieved through encryption and not fragmentation as in the case of Aggarwal (2005) and Ciriani (2010). Query execution is achieved in constant time with the use hash map but unlike Alhanjouri and Al Derawi (2012), the application is designed for cloud databases. It is implemented and tested using a recognized benchmark and the numerical results are obtained. Also, the technique is integrated into the structure of the DBMS without any alterations to the storage model as in the case of Kaul (2013) as encryption is not expected to alter the database structure (Jacob, 2010). On the whole, our method is simple, fast and achieves a high scalability with a time complexity of O(1) in most cases.

CHAPTER THREE

ARCHITECTURE AND DESIGN OF SEQURE SQL

3.1 Introduction

This chapter starts by introducing the framework of the proposed design known as *SecureSQL*. The proposed system model which comprises the design is explained. The architecture of the system which gives details of the algorithms and the major components involved is also presented. The tools and platform used in the implementation of the system is also highlighted.

3.2 System Model

At the most rudimentary level, a cloud storage system just needs one data server connected to the Internet. A subscriber copies files to the server over the Internet. When a client wants to retrieve the data, the client accesses the data server with a web-based interface, and the server then either sends the files back to the client or allows the client to access and manipulate the data itself.

Figure 3.1 gives an overview of the proposed system model. The model employs the client-server architecture. The client is trusted and the server is not trusted. This implies that the server is honest but curious. Most of the computations are to be performed on the client side. Data is encrypted on the client side using selective attribute-based encryption, before deployment to the cloud server. The client layer consists of the application which is accessed through a web browser. It connects through the internet to the Cloud Service Provider (CSP) and then to the data center, both of which make up the server side. When the owner is uploading or updating data, the application interface allows him to select the entity or relation he wants to encrypt. The attributes contained in the relation are displayed for the owner to select the attributes to be encrypted and the

relation involved. The data is then deployed to the google SQL database.

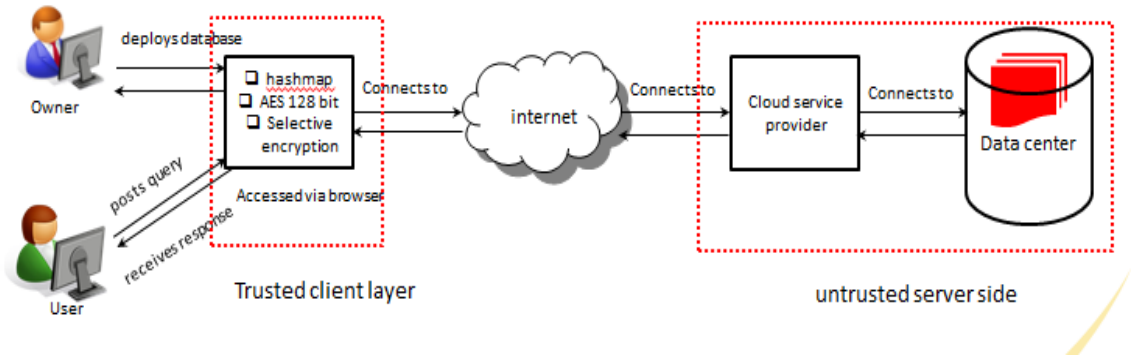


Figure 3.1: Overview of the Proposed System Model

The main functions performed are:

a. Deploying a database

b. Posting and processing queries

The system model comprises the following main entities:

a. Data owner: A person or organization who produces and outsources encrypted data to a server and also provides some additional information called metadata to client side for the purpose of query transformation.

b. Client: A trusted front-end entity that translates the queries posed into equivalent one for processing on encrypted data at server side.

c. The user: An entity that requires to access data (encrypted or unencrypted) stored at server by querying through a client front-end entity.

d. The cloud service provider (CSP) or Server: An external entity that returns the response of query to the client who in turn decrypts and returns the result to the user.

3.3 Assumptions

The assumptions made are based on a security model that guarantees privacy and they are all achievable.

a. Server is honest but curious (performs computations correctly but tries to glean additional information).
b. Client is fully trusted and won't be compromised hence stores the encryption/decryption keys.
c. The communication channel between client and server is secure by using TLS (Transport Layer Security) and SSL (Secure Sockets Layer).
d. Server does not have any domain knowledge about the data being stored by client.

3.4 System Architecture

The system architecture is a breakdown of the major functionalities of the system. The trusted client layer is a customized application which is built using java applets to enable secured web browser access and is connected to the database which is outsourced to a public cloud. The architecture is shown in Figure 3.2.

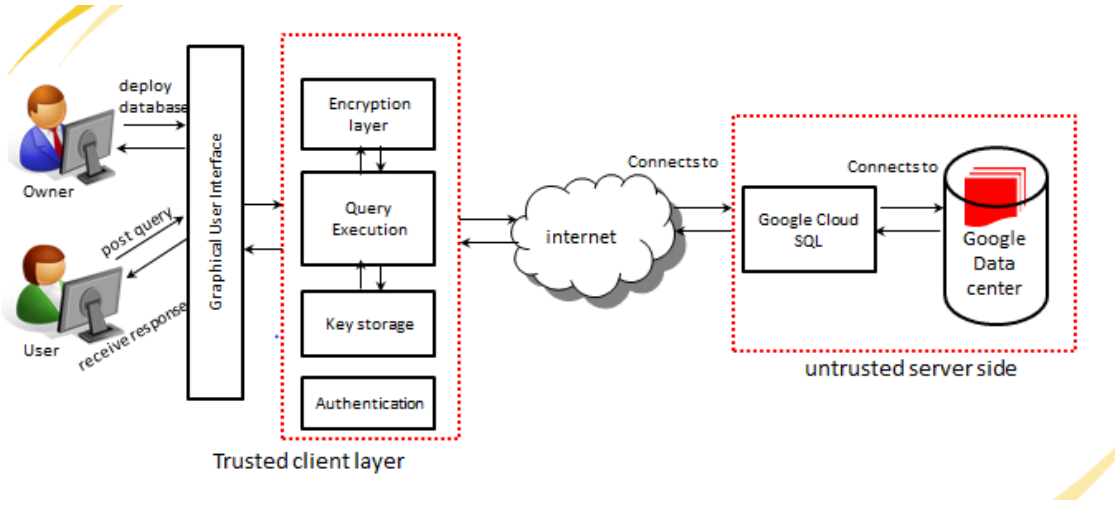


Figure 3.2: Proposed System Architecture

The encryption layer is further decomposed and shown in Figure 3.3. The query execution process is also split into the various processing units and shown in Figure 3.4.

3.4.1 Key Storage

The encryption keys are used for encrypting data on the client side and decrypting data on the server. Both the keys and the schema are not accessible to the service provider.

3.4.2 Authentication

Authentication is used to identify the users of the system and ensure access control for the entities involved. The owner privileges and that of the user are clearly distinguished. A user should not encrypt or upload data to the database. The user can perform queries and receive results.

3.4.3 Encryption/Decryption Engine

The *batch encryption* captures the relation, attribute and primary key in other to encrypt data already stored in the resident database. *Single row encryption* is used to perform encryption update on records to be inserted into the database and this is outlined in Figure 3.3.

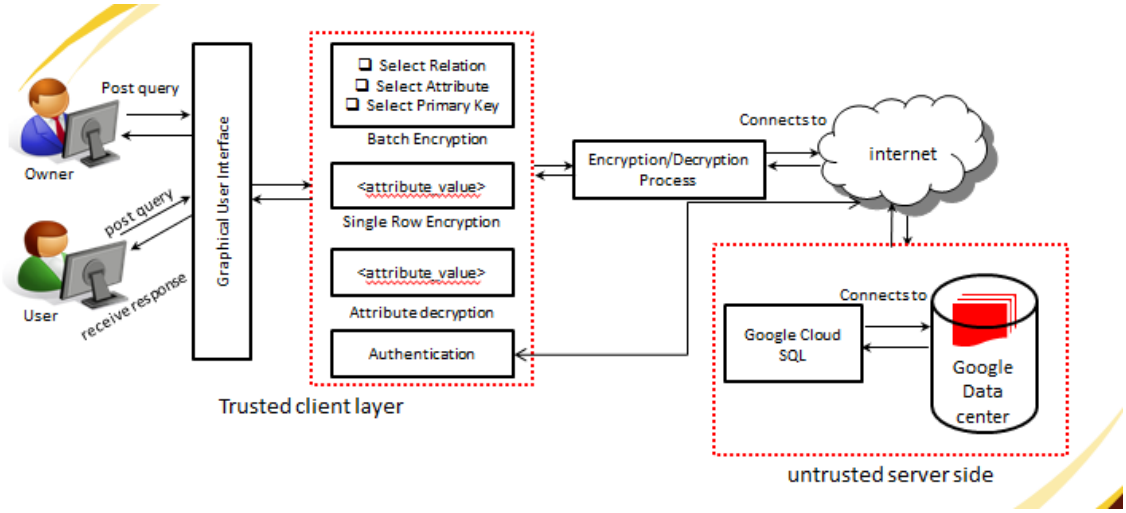


Figure 3.3: Encryption/Decryption Engine Architecture

The decryption process retrieves results which are still in encrypted form then decrypts on the client side and compiles the result (this is necessary because if the query involves both unencrypted attributes as well as encrypted attributes, the obtained result has to be compiled before it is displayed to the user).

In the decryption process, the angular braces "< >" act as a delimiter which is used by an inbuilt syntax analyzer. It indicates the beginning and end of an attribute that requires encryption to be applied to it. Hence, to query an encrypted attribute, the delimiter is used to group it so that the decryption function can be applied to it. Example:

SELECT <first_name>, last_name, <salary> FROM employees LIMIT 10;

This statement indicates that the attributes first_name and salary are encrypted. The result is plaintext values of first_name, last_name and salary records displayed.

3.4.4 Query Execution

A query analyzer is used to check for syntactic and semantic validity of the query. It checks that we have a valid sequence of tokens by breaking down a query into an array of tokens, each containing a word, and then examining each token one character at a time using regular expressions. The SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY tokens are analyzed one after the other.

a. Select
b. From
c. Where
d. Group By
e. Having
f. Order By

Structures with recognizable patterns are then treated according to the required rule. The final string is a refined query ready for execution. Since the encryption is selective, the syntax analyzer identifies which fields require decryption or further encryption by applying a set of rules to all columns enclosed in angular braces. This is done for each query posted.

The query transformation relies on some metadata (the hash map) which stores a key *k* for secure hashing and indexing. Different columns use different keys and indexing expressions.

As shown in Figure 3.4, users that need to access encrypted data submit their query which is then analyzed to know which attributes are to be accessed. Every attribute is associated with index information. Each plaintext query is mapped onto a corresponding query on the indexing content and executed in that form at the untrusted server. The hash map is initialized to fetch the key corresponding to the needed data then send the query to the cloud database for execution. Result of the query is returned to client for decryption and compilation and the final plaintext sent to the user. If the result is null, then a message is displayed to the user's interface as appropriate.

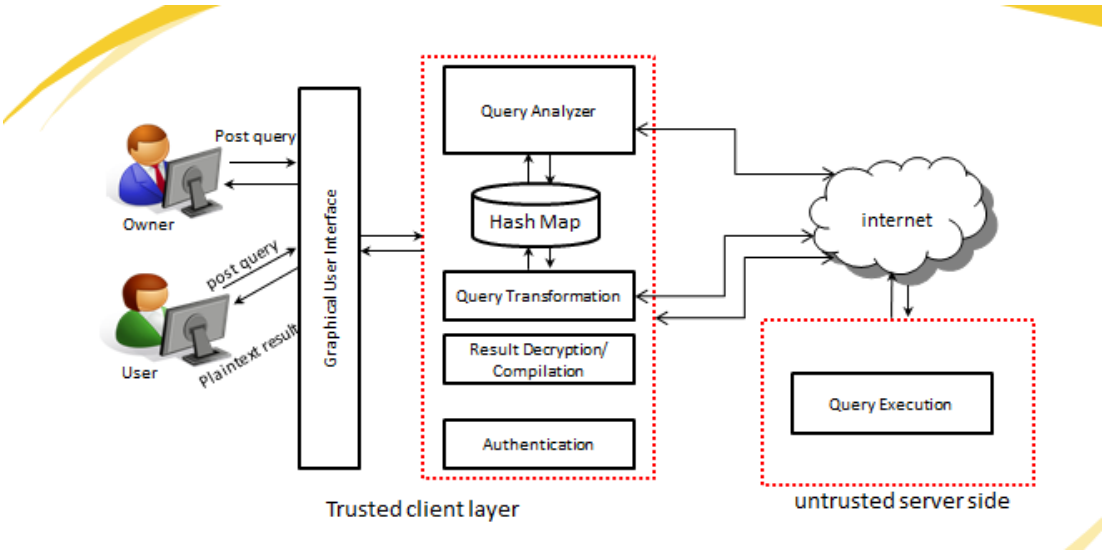


Figure 3.4: Query Execution Process

Users that only need to access the clear text content of the database submit the query directly to the server that stores the database and receive the clear text result with standard approaches.

Table 3.1: Query Execution Algorithm

S/No.	Algorithm
1	Start:
2	Define NewQuery as String
3	Define ValuesMap as Dictionary
4	LOOP (string Y in query.split(‘ ‘))
5	TEST: Y[0] == ‘<’ and Y[Y.length-1] == ‘>’ THEN
6	TEST: ValuesMap.HasKey(y) THEN
7	NewQuery += ValuesMap.GetValue(Y);
8	ELSE
9	NewQuery += AES_DECRYPT(Y);
10	ELSE
11	NewQuery += Y;
12	ENDLOOP
13	RETURN NewQuery;
14	End

Table 3.1 is an algorithm that explains the query execution process. ***Line 2*** declares a string variable to hold our query; ***line 3*** declares a dictionary to hold our key-value pairs in a hash map; ***line 4*** breaks query into bits using an empty space as delimiter; ***line 5*** determines if the column is to be encrypted or decrypted; ***line 6*** looks into the hash map first for the value; ***line 7*** if found, get value from the hash map instead of running a fresh decryption querying the database; ***line 9*** otherwise, run a new decryption for it; ***line 11*** if it is an ordinary column not requiring encryption or decryption; ***line 13*** new query returned for execution.

3.5 System Design

The design of the system is an implementation of the system model. Privacy of data is paramount especially because the database server is outsourced and not trusted. This informs the use of a provably secure encryption scheme – Advanced Encryption Standard which is implemented in a non-deterministic mode on a simple user-friendly interface. This is done from the client-side application and applied on the data in the database.

Performance with respect to queries is the key, even as the database is increased in scale, and the servers are remote, it is expected that the efficiency should not be compromised. Thus, hash map is used to achieve some constancy in time. On the whole, the design is built on the client-side without any alteration to the DBMS structure to ensure an easy change over for organizations that desire to deploy there database to the cloud.

The hashmap function $h(e)$ is used to store the encrypted values as key:value pair by mapping the key e to a given range. The implementation of hash map in this system ensures that the key:value pairs (plaintext:ciphertext) are stored in the metadata along with the schema of the database.on the server. The hash map keys are values gotten from already existing data stored on the server, but when loaded on to the hash map at execution time, the keys become available in memory on the client side. Hash map is initialized in a millionth of a second which is quite negligible. It ensures that simple querying conditions do not require decrypting fields before being executed and this positively influences the time.

Example*: SELECT salary, last_name FROM employees WHERE id='12GT098';*

If *'id'* is the encrypted field, the query will not be executed. Thus, the hash map then serves as a lookup from which a replacement would be done at run-time by matching keys with corresponding values. So, as explained in the query execution process, the encrypted column is enclosed in angular braces < > which acts as a placeholder, and therefore treated as requiring decryption.

SELECT salary, last_name FROM employees WHERE id='<12GT098>';

The experimental setup, consist of the google cloud SQL storage infrastructure with MySQL database. Data generated using TPCH dbgen is used for the experiment. The primary keys and other schema dependencies are as defined by TPCH.

3.6 Implementation Tools and Platform

Certain tools were used in the development and implementation of the proposed system. In addition to the method, these tools distinguish the work and the consequent results obtained.

3.6.1 MySQL Workbench

MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more.

MySQL Workbench enables a DBA, developer, or data architect to visually design, model, generate, and manage databases. It includes everything a data modeler needs for creating complex ER models, forward and reverse engineering, and also delivers key features for performing difficult change management and documentation tasks that normally require much time and effort.

3.6.2 Google Cloud SQL

In choosing a cloud service, consideration is made of that which provides the same set of features, in this case high performance and scalable querying service for massive datasets. Amidst options such as Amazon Redshift, Google BigQuery, Microsoft Azure SQL, Amazon Dynamo, Google CloudSQL is selected to implement and test the system because of its obvious advantages.

Google Cloud SQL is a MySQL database that lives in Google's cloud. It is used for storing the relational (SQL) database. It has all the capabilities and functionality of MySQL, with a few additional features and a few unsupported features. Google Cloud SQL is easy to use, does not require any software installation or maintenance, and is ideal for small to medium-sized applications. It can be accessed using MySQL Client, and other administration and reporting tools that work with MySQL databases.

3.6.3 TPC-H Benchmark

TPC-H called the "Transaction Processing Performance Council is an ad-hoc decision support benchmark that consists of a suite of business oriented queries designed to exercise system functionalities in a manner representative of complex business analysis applications. The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users. The queries and the data that populate the database have been chosen to have broad industry-wide relevance while maintaining a sufficient degree of ease of implementation. The benchmark is composed of 22 read-only queries. These queries are performed on considerably large amounts of data, have a high degree of complexity, and were chosen to give answers to critical business questions. Queries are created by a program named *qgen* while data is generated by a program called

dbgen. The data size ranges from 1GB up to 100,000GB. The data is loaded and the queries are executed on a regular DBMS system.

TPC-H stresses the specific part of the database system which is being measured. Since the interest of this research lies in high performance (query evaluation time) as the data scales (trend of query evaluation time with increasing data size and number of nodes), it comes across as ideal.

The TPC-H database consists of 8 tables (part, partsupp, lineitem, orders, customer, supplier, nation, region) and the data can be generated for a set of fixed scale factors (SF) (1, 10, 30, 100, 300, 1000, 3000, 10000, 30000, 100000). Table 4.1 shows the number of records for each table.

Table 3.1: TPCH Tables and Number of Records

Table name	**Part**	**Partsupp**	**Lineitem**	**orders**	**customer**	**Supplier**	**nation**	**Region**
No of Rows	SF*200,000	SF*800,000	SF*6,000,000	SF*1,500,000	SF*150,000	SF*10,000	25	5

The scale factor also specifies how many GB of data will be generated. For example, for a SF of 30, 30GB of data will be generated, out of which the lineitem table will have 23GB of data. For this experiment 1GB of data will be generated, that is using SF 1.

3.6.4 DBGen (Database Generator)

DBGEN is a utility that allows the creation of a series of flat files which contain the data for a TPC-H schema.

CHAPTER FOUR

IMPLEMENTATION, RESULTS AND DISCUSSION

4.1 Introduction

This chapter begins by generating the test data used for the database. It goes on to the software and system development by explaining the implementation of the client side application and its connection to the cloud database. Furthermore, testing of the queries is done by analyzing and evaluating the performance using statistical tools. In conclusion, the experimental results arrived at, are critically discussed.

4.2 Generation of test data with dbgen for the tpc-h database using visual studio

a. Installed Microsoft visual studio 2013 and changed the path of the system environment variables to C:\Program Files\Microsoft Visual Studio 12.0\VC\bin.
b. Downloaded and extracted tpc_h 2_17_0 to another folder.
c. Opened the tpc_h folder, opened the makefile.suite file and edited four lines:

   CC=CL

   Database=SQLServer

   Machine=win32

   Workload=TPCH
d. Opened Microsoft visual studio, opened project tpch.sln in the solution explorer. Right clicked dbgen file on the explorer interface and built. The process is shown in Figure 4.1.

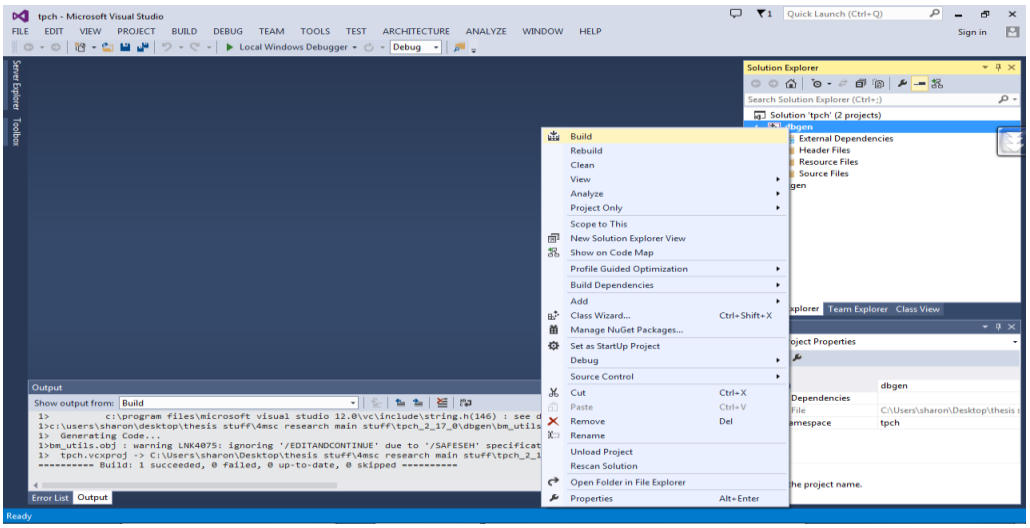


Figure 4.1: Generating dbgen.exe using Microsoft Studio

e. After a successful build, the dbgen.exe file was generated in the debug folder of the tpch_2.17_0 folder.

...tpch_2_17_0\dbgen\Debug\dbgen.exe

f. Opening the command prompt, changed the directory to:

...\tpch_2_17_0\dbgen\debug

This is shown in Figure 4.2.

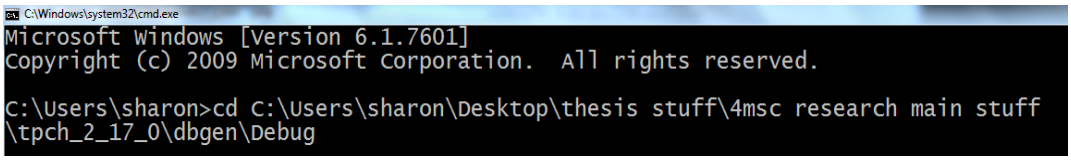


Figure 4.2: Command Prompt showing dbgen.exe directory

g. In the directory, typed dbgen.exe –h to get help options as seen in Figure 4.3.

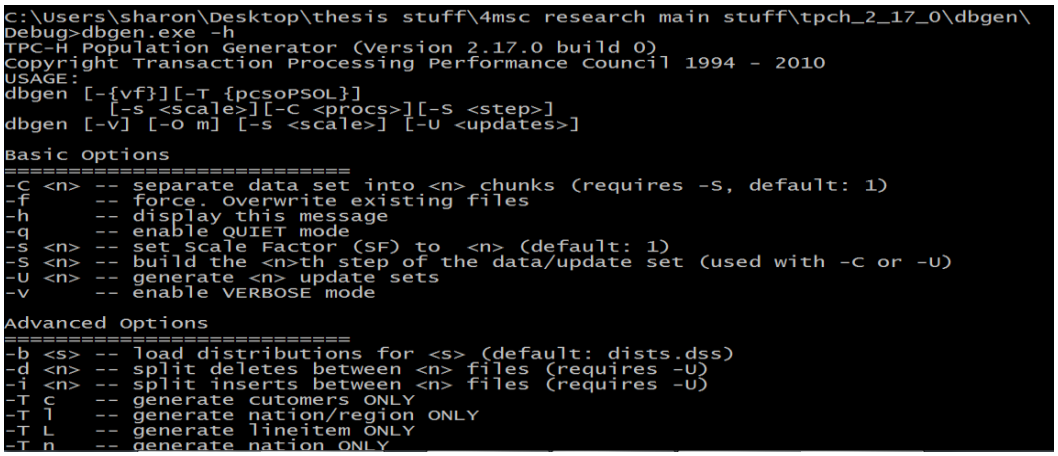


Figure 4.3: dbgen.exe Help Options

h. In other to generate 1GB database, that implies a scale factor(SF)=1, typed:

>dbgen –vf –s 1

i. This generated an error *"cant open .\dists.dss",* so the dists.dss file was copied from the dbgen folder to the debug folder as a solution. Figure 4.4 shows the error screen.

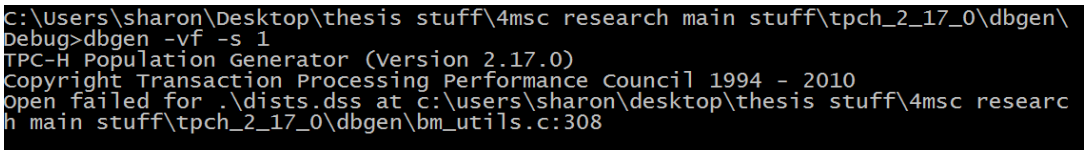


Figure 4.4: Error in Data Generation

j. Then *>dbgen –vf –s 1* was run again successfully as seen in Figure 4.5.

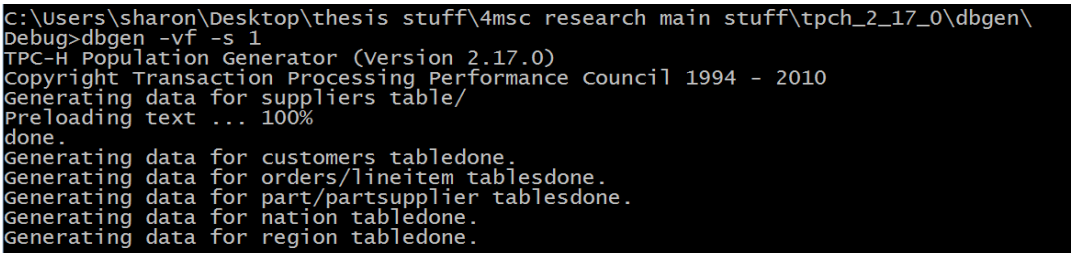


Figure 4.5: Generation of Eight Flat File(.tbl) Tables

k. Data was generated in the debug folder for eight tables as *.tbl*.

4.3 Data loading into the database from the flat .tbl files

Using the outlined steps, the data was loaded into the database and prepared for use.

a. Create a new schema/database from MySQL workbench:

   CREATE DATABASE tpch;

b. Choose the database to use:

   USE tpch;

c. Create the table structures using the dss.ddl file in the tpch_2_17_0 folder:

   -- Sccsid: @(#)dss.ddl 2.1.8.1

   CREATE TABLE NATION (N_NATIONKEY INTEGER NOT NULL,
   N_NAME CHAR(25) NOT NULL,
   N_REGIONKEY INTEGER NOT NULL,
   N_COMMENT VARCHAR(152));

   CREATE TABLE REGION (R_REGIONKEY INTEGER NOT NULL,
   R_NAME CHAR(25) NOT NULL,
   R_COMMENT VARCHAR(152));

   CREATE TABLE PART (P_PARTKEY INTEGER NOT NULL,
   P_NAME VARCHAR(55) NOT NULL,
   P_MFGR CHAR(25) NOT NULL,
   P_BRAND CHAR(10) NOT NULL,
   P_TYPE VARCHAR(25) NOT NULL,
   P_SIZE INTEGER NOT NULL,
   P_CONTAINER CHAR(10) NOT NULL,
   P_RETAILPRICE DECIMAL(15,2) NOT NULL,
   P_COMMENT VARCHAR(23) NOT NULL);

   CREATE TABLE SUPPLIER (S_SUPPKEY INTEGER NOT NULL,
   S_NAME CHAR(25) NOT NULL,
   S_ADDRESS VARCHAR(40) NOT NULL,
   S_NATIONKEY INTEGER NOT NULL,
   S_PHONE CHAR(15) NOT NULL,
   S_ACCTBAL DECIMAL(15,2) NOT NULL,
   S_COMMENT VARCHAR(101) NOT NULL);

   CREATE TABLE PARTSUPP (PS_PARTKEY INTEGER NOT NULL,
   PS_SUPPKEY INTEGER NOT NULL,
   PS_AVAILQTY INTEGER NOT NULL,
   PS_SUPPLYCOST DECIMAL(15,2) NOT NULL,
   PS_COMMENT VARCHAR(199) NOT NULL);

   CREATE TABLE CUSTOMER (C_CUSTKEY INTEGER NOT NULL,

C_NAME VARCHAR(25) NOT NULL,
C_ADDRESS VARCHAR(40) NOT NULL,
C_NATIONKEY INTEGER NOT NULL,
C_PHONE CHAR(15) NOT NULL,
C_ACCTBAL DECIMAL(15,2) NOT NULL,
C_MKTSEGMENT CHAR(10) NOT NULL,
C_COMMENT VARCHAR(117) NOT NULL);

CREATE TABLE ORDERS (O_ORDERKEY INTEGER NOT NULL,
O_CUSTKEY INTEGER NOT NULL,
O_ORDERSTATUS CHAR(1) NOT NULL,
O_TOTALPRICE DECIMAL(15,2) NOT NULL,
O_ORDERDATE DATE NOT NULL,
O_ORDERPRIORITY CHAR(15) NOT NULL,
O_CLERK CHAR(15) NOT NULL,
O_SHIPPRIORITY INTEGER NOT NULL,
O_COMMENT VARCHAR(79) NOT NULL);

CREATE TABLE LINEITEM (L_ORDERKEY INTEGER NOT NULL,
L_PARTKEY INTEGER NOT NULL,
L_SUPPKEY INTEGER NOT NULL,
L_LINENUMBER INTEGER NOT NULL,
L_QUANTITY DECIMAL(15,2) NOT NULL,
L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,
L_DISCOUNT DECIMAL(15,2) NOT NULL,
L_TAX DECIMAL(15,2) NOT NULL,
L_RETURNFLAG CHAR(1) NOT NULL,
L_LINESTATUS CHAR(1) NOT NULL,
L_SHIPDATE DATE NOT NULL,
L_COMMITDATE DATE NOT NULL,
L_RECEIPTDATE DATE NOT NULL,
L_SHIPINSTRUCT CHAR(25) NOT NULL,
L_SHIPMODE CHAR(10) NOT NULL,
L_COMMENT VARCHAR(44) NOT NULL);

d. Insert data into the tables created with the generated dummy data by executing the commands below in workbench:

load data local infile '... /part.tbl' into table part fields terminated by '|' lines terminated by '\n';
load data local infile '.../partsupp.tbl' into table partsupp fields terminated by '|' lines terminated by '\n';
load data local infile '.../customer.tbl' into table customer fields terminated by '|' lines terminated by '\n';
load data local infile '.../nation.tbl' into table nation fields terminated by '|' lines terminated by '\n';
load data local infile '.../orders.tbl' into table orders fields terminated by '|' lines terminated by '\n';

load data local infile '.../region.tbl' into table region fields terminated by '|' lines terminated by '\n';
load data local infile '.../supplier.tbl' into table supplier fields terminated by '|' lines terminated by '\n';
load data local infile '.../lineitem.tbl' into table lineitem fields terminated by '|' lines terminated by '\n';

e. Alter the schema dependencies (from the dss.ri file located in the tpch folder) by adding primary and unique keys:

-- Sccsid: @(#)dss.ri 2.1.8.1

-- TPCD Benchmark Version 8.0

-- For table REGION
ALTER TABLE REGION
ADD PRIMARY KEY (R_REGIONKEY);

-- For table NATION
ALTER TABLE NATION
ADD PRIMARY KEY (N_NATIONKEY);

ALTER TABLE NATION
ADD FOREIGN KEY NATION_FK1 (N_REGIONKEY) references REGION(R_REGIONKEY);

-- For table PART
ALTER TABLE PART
ADD PRIMARY KEY (P_PARTKEY);

-- For table SUPPLIER
ALTER TABLE SUPPLIER
ADD PRIMARY KEY (S_SUPPKEY);

ALTER TABLE SUPPLIER
ADD FOREIGN KEY SUPPLIER_FK1 (S_NATIONKEY) references NATION(N_NATIONKEY);

-- For table PARTSUPP
ALTER TABLE PARTSUPP
ADD PRIMARY KEY (PS_PARTKEY,PS_SUPPKEY);

-- For table CUSTOMER
ALTER TABLE CUSTOMER
ADD PRIMARY KEY (C_CUSTKEY);

ALTER TABLE CUSTOMER

```
ADD FOREIGN KEY CUSTOMER_FK1 (C_NATIONKEY) references
NATION(N_NATIONKEY);

-- For table LINEITEM
ALTER TABLE LINEITEM
ADD PRIMARY KEY (L_ORDERKEY,L_LINENUMBER);

-- For table ORDERS
ALTER TABLE ORDERS
ADD PRIMARY KEY (O_ORDERKEY);

-- For table PARTSUPP
ALTER TABLE PARTSUPP
ADD FOREIGN KEY PARTSUPP_FK1 (PS_SUPPKEY) references
SUPPLIER(S_SUPPKEY);

ALTER TABLE PARTSUPP
ADD FOREIGN KEY PARTSUPP_FK2 (PS_PARTKEY) references
PART(P_PARTKEY);

-- For table ORDERS
ALTER TABLE ORDERS
ADD FOREIGN KEY ORDERS_FK1 (O_CUSTKEY) references
CUSTOMER(C_CUSTKEY);

-- For table LINEITEM
ALTER TABLE LINEITEM
ADD FOREIGN KEY LINEITEM_FK1 (L_ORDERKEY) references
ORDERS(O_ORDERKEY);

ALTER TABLE LINEITEM
ADD FOREIGN KEY LINEITEM_FK2 (L_PARTKEY,L_SUPPKEY) references
    PARTSUPP(PS_PARTKEY, PS_SUPPKEY);
```

4.4 TPCH Schema Diagram

The TPCH schema diagram was generated by performing a reverse engineering of the populated database using Microsoft visual studio and this is displayed as Figure 4.6.

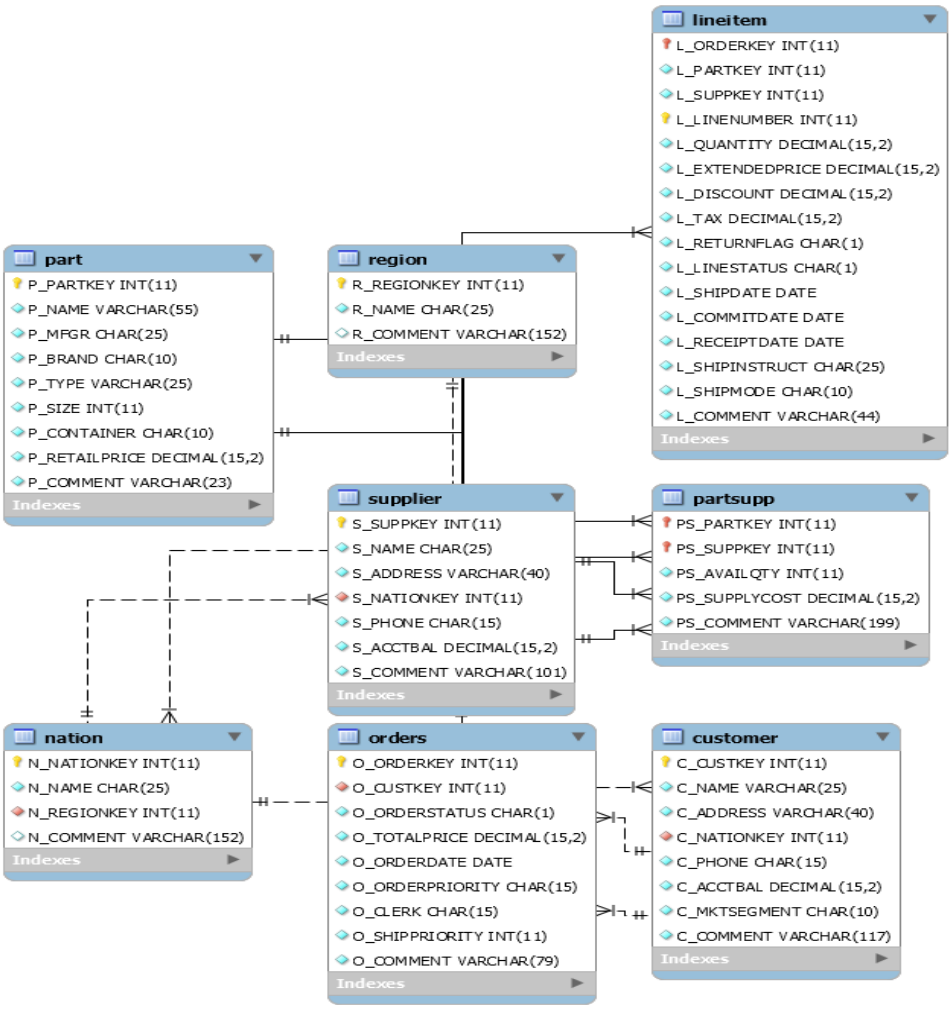


Figure 4.6: TPCH Schema

4.5 System Implementation

This section gives a breakdown of the implementation of the application modules developed, from the application interface to the functionalities described in the architecture

4.5.1 Code Implementation

a. **AES.java class:** This is a java program for the AES-128 bit encryption algorithm in CBC mode. It also does the handling of the keys which are stored as java variables.

b. **BatchEncrypt.java class:** This module is responsible for the batch encryption of pre-selected columns as further explained later.

c. **SQLHelper.java class:** This module is an implementation of the hash map support and the query syntax analyzer.

d. **SecuresSQL.java class:** All the functionalities explained are brought together under this module, thus it is the main class.

4.5.2 Client Side Application

All the interface modules are programmed in java using a combination of JtabbedPane and Jtable components.

With the JtabbedPane class, you can have several components, such as panels, share the same space. The user chooses which component to view by selecting the tab corresponding to the desired component. The resulting interface is shown as Figure 4.7, then Figure 4.8 shows the interface with an active query set.

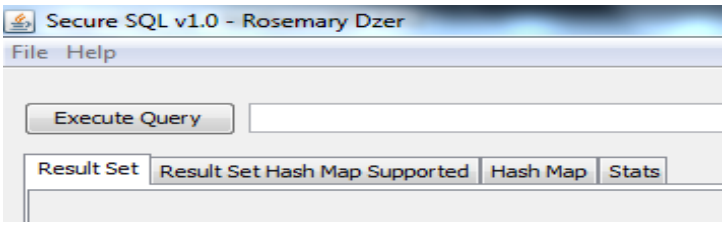


Figure 4.7: *SecureSQL* Graphical User Interface (GUI)

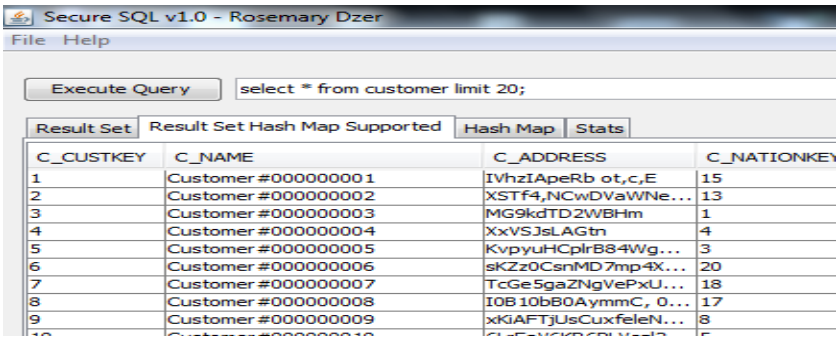


Figure 4.8: *SecureSQL* GUI with Query result set

The Jtable is used to display and edit regular two-dimensional tables of cells. The Jtable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily. For example, to set up a table with 10 rows and 10 columns of numbers:

```
TableModel dataModel = new AbstractTableModel() {
    public int getColumnCount() { return 10; }
  public int getRowCount() { return 10;}
public Object getValueAt(int row, int col) { return new Integer(row*col); }
    };
    Jtable table = new Jtable(dataModel);
    JscrollPane scrollpane = new JscrollPane(table)
```

Jtables are typically placed inside of a JscrollPane. By default, a Jtable will adjust its width such that a horizontal scrollbar is unnecessary.

A set of other controls is used in order to reach the desired purpose, what concerns the functionality of the application, including Labels, Text boxes, Combo Boxes, Data Grid, Buttons, Group Boxes, Panels and Tab controls. All of these controls, available in the program, are fitted to the corresponding Jforms that are used in the application.

Each Jtable is loaded with dataset from the query returned. DataSet is a memory-resident representation of data that provides a consistent relational programming model independent of the data source. The DataSet represents a complete set of data including tables, constraints, and relationships among the tables. Because the DataSet is independent of the data source, a DataSet can include data local to the application, as well as data from multiple data sources.

4.5.3 Interface Tabs

The application GUI comprises of tabs with different functionalities as explained.

a. **Result Set:** Displays result of query without HashMap support.
b. **Result Set Hash Map Supported:** Displays the result of query with HashMap Support.
c. **Hash Map:** Displays the content of the Hashmap initialized during program startup phase.
d. **Stats:** Shows execution time for each query with or without HashMap.
e. **Batch Encrypt:** A button that initializes the encryption process by opening a window to capture user inputs
f. **Encryption Key:** The module stores the encryption key which was earlier explained to be symmetric that is, one key to encrypt and decrypt.

4.5.4 Encryption Process

Encrypting selective column values was implemented on the client side application with AES-128 bit key in CBC mode using the batch encrypt method to encrypt whole columns at a time and the insert statement for insertion and encryption of single rows at a time.

a. Batch Encrypt Method

Figure 4.9 shows the batch encryption process from start to completion. The batch encrypt method written in java captures the relation name, column name and the primary key of the column whose values is to be batch encrypted. One column values is encrypted at a time. It processes based on the unique primary key to avoid duplication. Both numeric and textual characters are processed. Before this step, it is ensured that the field length is long enough to accept the cipher text which is much longer than the plain text in other to avoid truncating the cipher value.

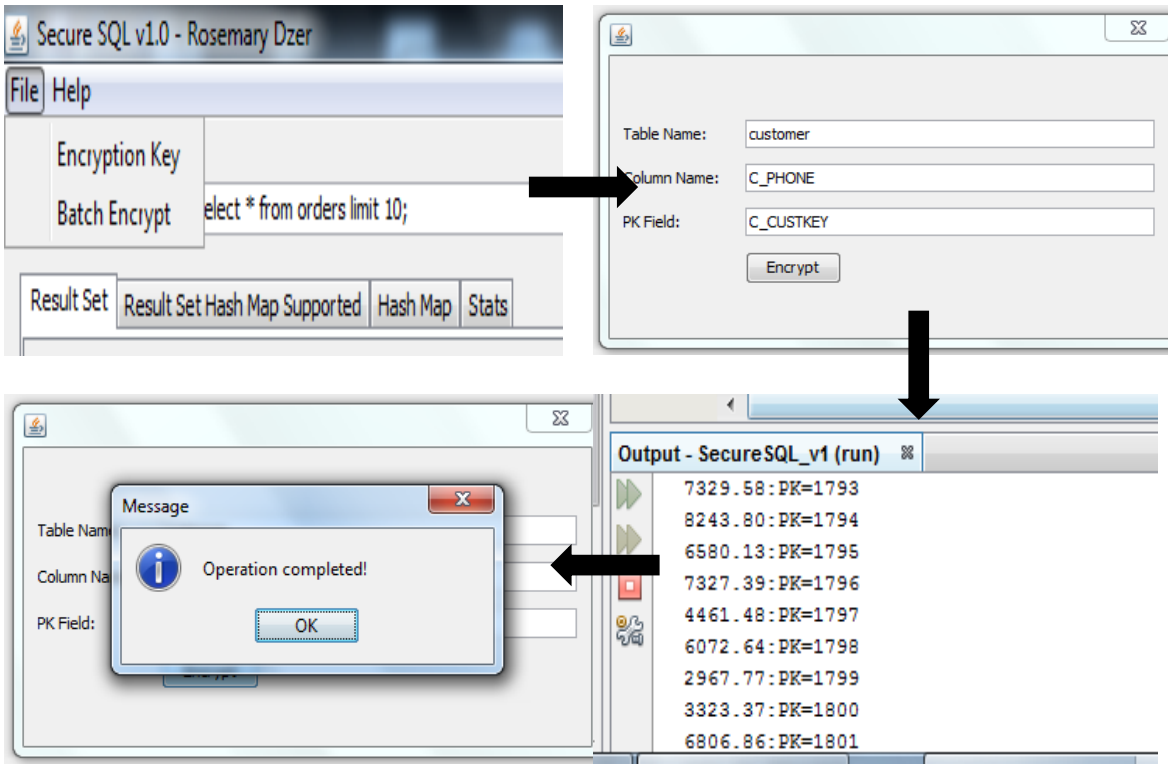


Figure 4.9: Batch Encrypt Processes

b. Single row encryption

For a schema that is not populated or addition of rows to an already populated database, the insert statement is used and the values intended for encryption are enclosed in angular braces. Example for table supplier, Figure 4.10 shows that S_ADDRESS, S_PHONE and S_ACCTBAL contain encrypted data.

S_SUPPKEY	S_NAME	S_ADDRESS	S_NATIONKEY	S_PHONE	S_ACCTBAL
1	Supplier#000000001	N kD4on9OM Ipw3,gf0JBoQDd7t...	17	VlRIQfTbOBpuoEBNx9Aw5A==	ktfEu+Z5CEPcpxUd/vitNw==
2	Supplier#000000002	89eJ5ksX3ImxJQBvxObC,	5	Pz4ne8hsKyhgoVaanWHJg==	Mb0cR4sHYCrDdZ8sZGrrYw==
3	Supplier#000000003	q1,G3Pj6OjIuUYfUoH18BFTKP5aU...	1	5ltNi0fGtVDXg1yacCzK+w==	Nt8u2JD4bX+HUKyUXVXemA==
4	Supplier#000000004	Bk7ah4CK8SYQTepEmvMkkgMwg	15	hA5R8eHYiehLutxUY8Ilug==	Qo7oEzck8xLPzfFc3hNzeQ==
5	Supplier#000000005	Gcdm2rJRzl5qlTVzc	11	CDJT+8wGP7Cm9CgVzgPyjg==	+aGSPnMSknrjK4l1pb/9mQ==
6	Supplier#000000006	tQxuVm7s7CnK	14	SM0tnJfF/aN1SgfXh8L5vQ==	Pk12tXtchNjr6xGpQqrBPA==
7	Supplier#000000007	s,4TicNGB4uO6PaSqNBUq	23	Iu3PscLyex947odAtQASTg==	nM7t212K5ODuBHh5GSp4vA==
8	Supplier#000000008	9Sq4bBH2FQEmaFOocY45sRTxo6...	17	HHLsW7fanosLTlUP8rIzHQ==	usNOV6mf0Q0oL7nCs8rbvQ==
9	Supplier#000000009	1KhUgZegwM3ua7dsYmekYBsK	10	BDR4vRkw21uxKETUo7gFkw==	yaZbmLPAeJoGcEx09V++lQ==
10	Supplier#000000010	Saygah3gYWMp72i PY	24	4CiIwpqRtKp8kjwPvDw/Qw==	JMlkHOJRQuRS38Uey3+Qag==

Figure 4.10: Single row encryption

Thus, inserting additional encrypted data to these fields is as depicted:

INSERT INTO supplier VALUES('10,001', 'Supplier #000010001', '<abu, zaria>', '24', '<28-918-335-1736>', '<3991.91>', 'newly added row');

4.5.5 Columns encrypted in the TPCH database

Table 4.1 shows the selected columns from the tables in the TPCH schema which are encrypted. Figure 4.11 is a screen shot of the SQL statements that created a unified view of all encrypted columns for use in hash map loading and initialization at runtime.

Table 4.1: Selective Columns Encrypted in the TPCH Schema

S/N	TABLE NAME	COLUMN NAME	DATA TYPE	NO OF RECORDS
1	Customer	C_PHONE	VARCHAR(50)	150,000
		C_ACCTBAL	VARCHAR(50)	
2	Part	P_RETAILPRICE	VARCHAR(50)	200,000
3	Supplier	S_PHONE	VARCHAR(50)	10,000
		S_ACCTBAL	VARCHAR(50)	

```
Limit to 10 rows
1  CREATE VIEW `tpch`.`vw_BulkMap` AS
2  SELECT `C_PHONE` AS K FROM customer
3  UNION
4  SELECT `C_ACCTBAL` AS K FROM customer
5  UNION
6  SELECT `P_RETAILPRICE` AS K FROM part
7  UNION
8  SELECT `S_PHONE` AS K FROM supplier
9  UNION
10 SELECT `S_ACCTBAL` AS K FROM supplier
```

Figure 4.11: A Unified View of Selected Encrypted Columns in the TPCH Database

4.5.6 Decryption Process

In other to select any encrypted values and display them in decrypted format, the field name is enclosed in angular braces and must be written in the same case as specified in

the schema. Example: As seen in Figures 4.12, the C_PHONE data is encrypted thus the statement:

SELECT C_CUSTKEY, <C_PHONE> from customer LIMIT 10;

The result of C_PHONE in plain text is then displayed.

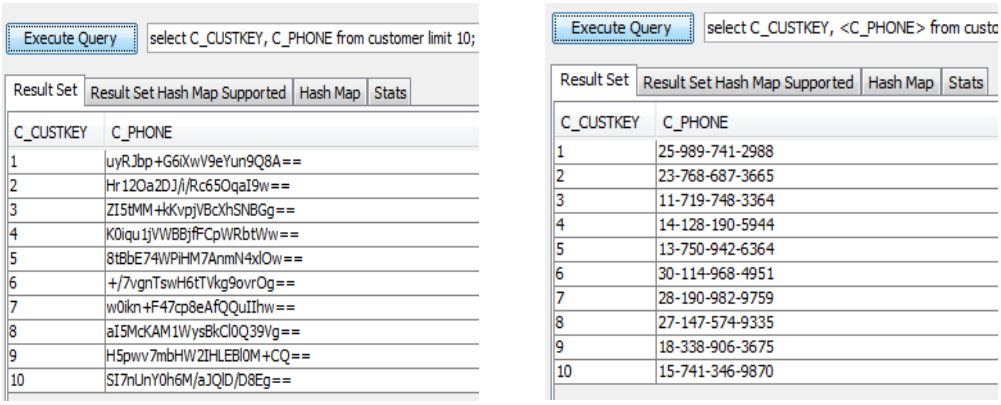


Figure 4.12: Decryption in Process

4.5.7 Update Process

In other to perform update on one or more tables in the TPCH schema, the update statement is used and any data values intended for encryption enclosed in braces. Example:

UPDATE customer SET C_PHONE='<16-741-346-9870>' WHERE C_CUSTKEY='10';

This statement changes the value of C_PHONE where C_CUSTKEY is 10 to 16-741-346-9870 if it was not previously so, then encrypts the same value.

4.5.8 Delete Process

In other to delete existing data values from one or more tables in the TPCH schema, the usual delete statement is used. Example:

DELETE FROM customer where C_CUSTKEY='10';

This statement deletes an entire row that contains data referenced by C_CUSTKEY equals 10.

4.6 Google Cloud SQL instance connection

Cloud SQL instances are fully managed, relational MySQL databases. Google handles replication, patch management and database management to ensure availability and performance. The following are sequential steps for connecting SecureSQL and the local instance of TPCH to google cloud SQL instance using MySQL monitor.

a. Go to console.developers.google.com and sign in with a google account. Click on create project and enter the name of a project, select agree and create – *benchmarkingtpch2015*.
b. On the left panel of the project page, go to storage and click on cloud SQL, then sign up.
c. Enter billing information.
d. Click on cloudSQL and create instance – tpchinstance choosing the region - US Central and tier – D2 – 1GB RAM, then create.
e. Create a user by clicking on the instance name under access control (user name, password and % to allow any host).
f. Request IPv4 from the access control panel – *173.194.248.242*.
g. Authorize allowed networks by entering the Ipv4 address - *173.194.248.242* and the host IP address - *192.220.66.2*
h. Set root password from the user options
i. Authorize access to the instance from MySQL client command prompt

   > mysql –host=*173.194.248.242* –user=*rosemary* –password

   >Enter password:
j. The mysql monitor is launched as shown in figure 4.13.

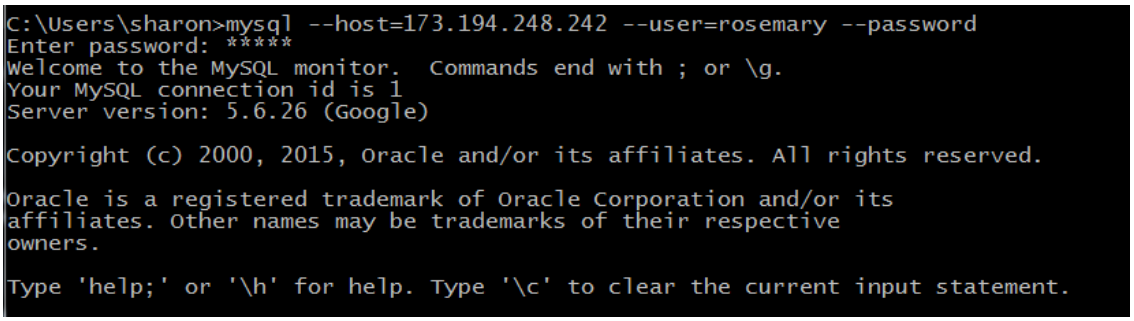


Figure 4.13: Authorize Access to Google Cloud Instance

4.7 Export and Import of database to google cloud SQL

a. Export data from tpch database (.sql file) on the host machine using mysql workbench with the mysql dump command *(tpch data and structure.sql).* Figure 4.14 shows the export in process.

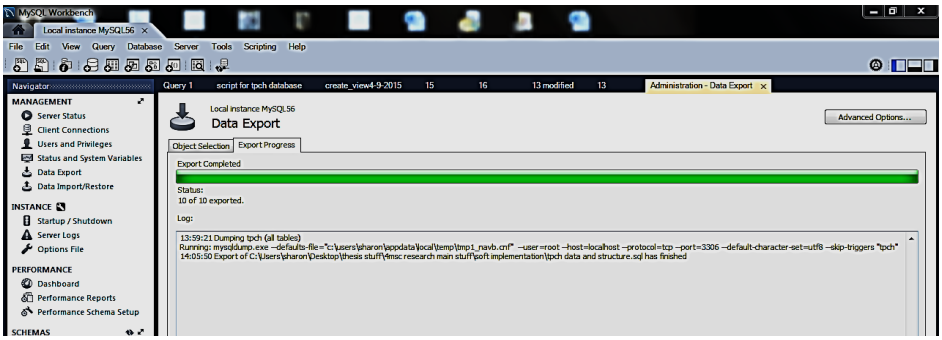


Figure 4.14: Exporting Data in .sql Format

b. Install *gsutil* tool which enables the dumping of the .sql file in google cloud storage bucket. Execute *python.exe C:\gsutil\gsutil.py config -b* via the python interpreter directory on the command prompt interface. This opens a web browser to display an authentication code that is copied and pasted on the command prompt. The google cloud console is opened via a browser displaying the project Id (benchmarkingtpch2015), copy and paste in the command prompt. Dump the .sql files to google cloud storage as shown in Figure 4.15.

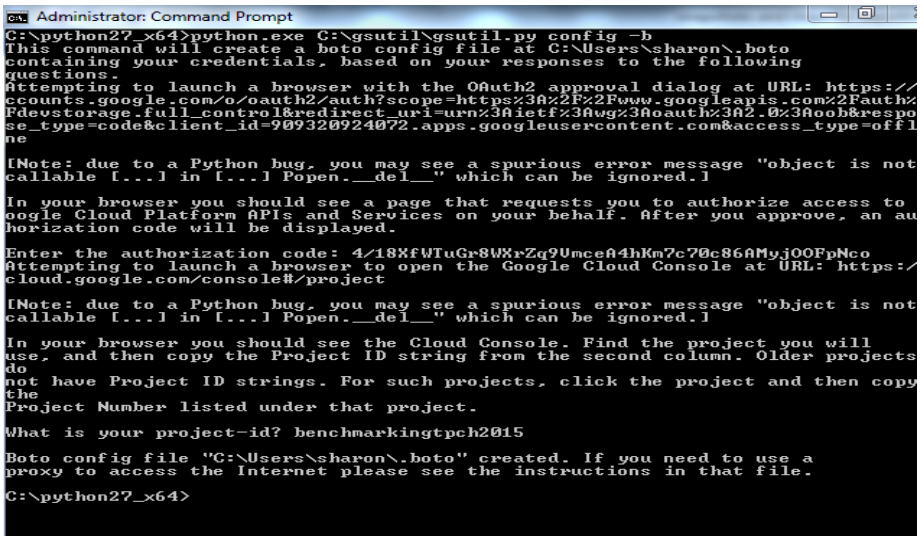


Figure 4.16: Dumping of the .sql files on Google Cloud Storage

c. From the python directory or the google console, type: *python.exe C:\gsutil\gsutil.py mbgs://tpchbucket*, to create a bucket (tpchbucket) on google cloud storage.

d. Upload "*the tpch data and structure .sql"* file to the created bucket as seen in Figure 4.17.

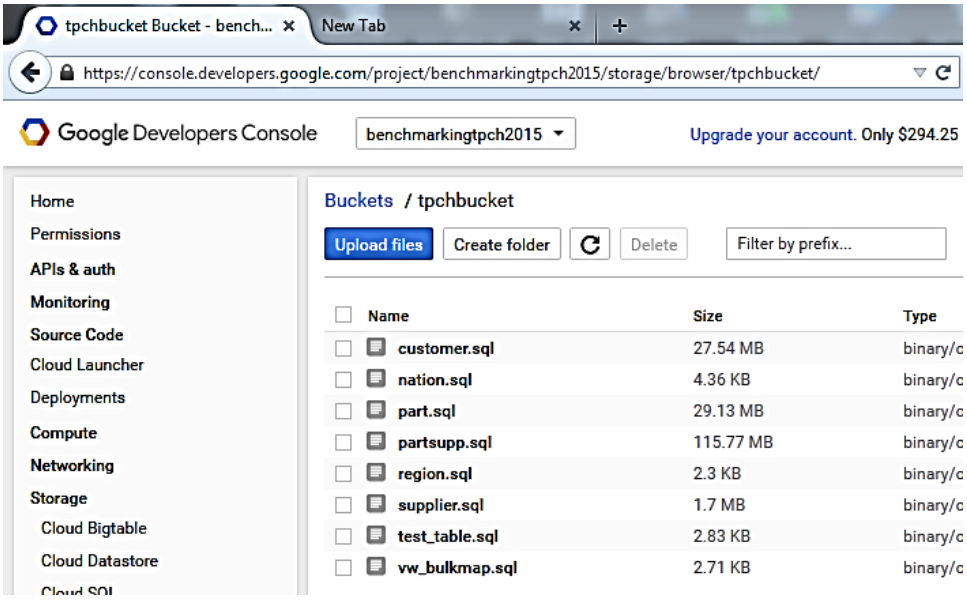


Figure 4.17: Uploading .sql files to Cloud Storage Bucket

e. Import the data from the *'tpchbucket'* in the cloud storage to google cloud SQL database. The import process is shown in Figure 4.18.

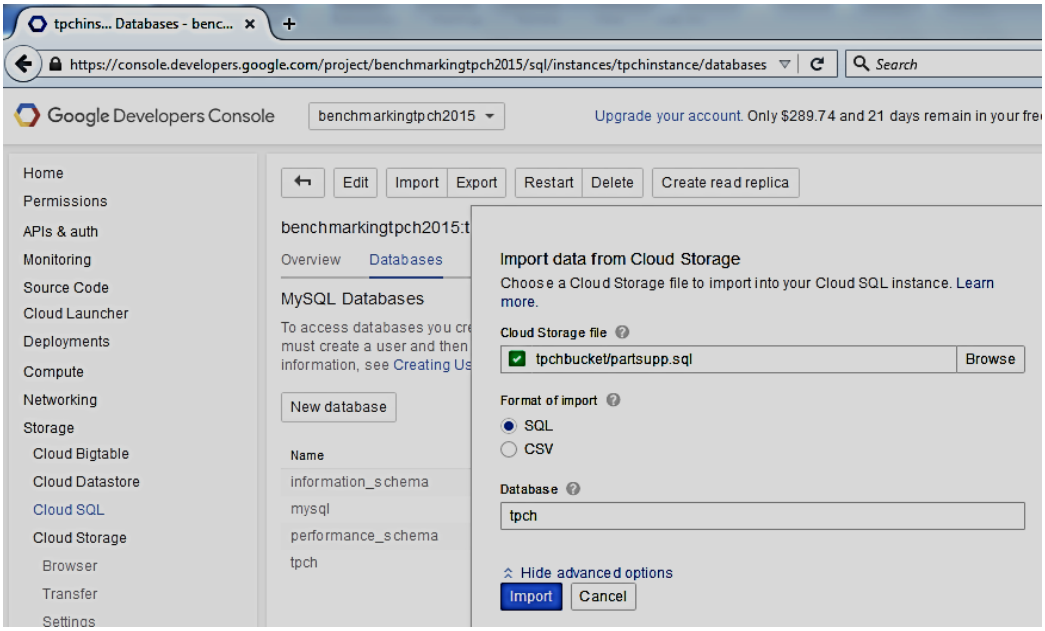


Figure 4.18: Importing Data from Cloud Bucket to Cloud SQL

f. Connect to google cloud SQL from the MySQL client interface (refer to section 4.6) or set the connection parameters from the client application, to view all uploaded tables in the tpch database and run queries as seen in Figure 4.19.

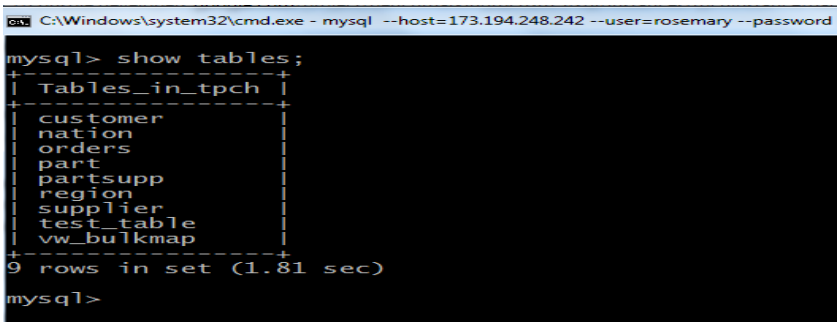


Figure 4.19: Viewing the Cloud Tables from mysql Client

g. Connecting *SecureSQL* application to google cloud SQL requires changing the connection parameters (username, password and servername) from the SQLHelper.java file and pointing the application to the cloud as shown in Figure 4.20.

```
34
35      public SQLHelper() {
36          // initialize class variables
37          //password = userName = "rosemary";
38          //userName = "root";
39          userName = "rosemary";
40          password = "mysql";
41          dbms = "mysql";
42          //dbName = "employees";
43          dbName = "tpch";
44          //serverName = "localhost";
45          serverName = "173.194.248.242";
46          portNumber = "3306";
47          tokens = new ArrayList<>();
48      }
49
```

Figure 4.20: Connecting *SecureSQL* to Google Cloud SQL

4.8 Query Testing, Results and Discussion

In order to test the performance, query response time was taken as a measurement across DML statements with different conditions as expressed in the 22 TPCH queries. Two main questions are of interest:

a. What is the scale up of the traditional method and the proposed method, that is, how does the query runtime vary with increasing dataset size?

b. How many of the TPCH queries actually are able to execute on the proposed system?

Question 1 addresses scalability. This is the trend of query evaluation time with increasing data size (varying workloads). Each statement was iterated at least 10 times and for every attempt query response time was noted and finally average was calculated for all iterations. The results for specific TPC-H benchmark queries (1, 2, 6 and 16) are presented. These specific queries contain all the characteristics exhibited by all the queries for string data (equality (=) and pattern matching (*LIKE*)) and numerical data (equality and range matching (<. >, =)). The number of records is on the x axis and the time of the different methods on the y axis. Question 2 is focused on the query response time for different types of queries in the TPC-H benchmark and whether they were

successfully executed. The result for query processing using *secureSQL* is hereby presented.

Query 1 captures a scan of the *lineitem* relation which contains 6 million records to retrieve data from nine columns involving computations of aggregate and range values with the *GROUP BY* and *ORDER BY* clause. Table 4.2 shows the mean execution time and Figure 4.21 is a line graph of time against the number of records, plotted from the data in Table 4.2

*select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extendedprice * (1 - l_discount)) as sum_disc_price, sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge, avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as avg_disc, count(*) as count_order from lineitem where l_shipdate <= date '1998-12-01' group by l_returnflag, l_linestatus order by l_returnflag, l_linestatus;*

Table 4.2: Mean Execution Time for Query 1

	Mean Time (in seconds) for Query 1	
No of Records	**with hashmap support**	**without hashmap support**
10	0.108	93.133
100	0.210	87.825
1000	5.890	86.113
10000	0.103	100.975

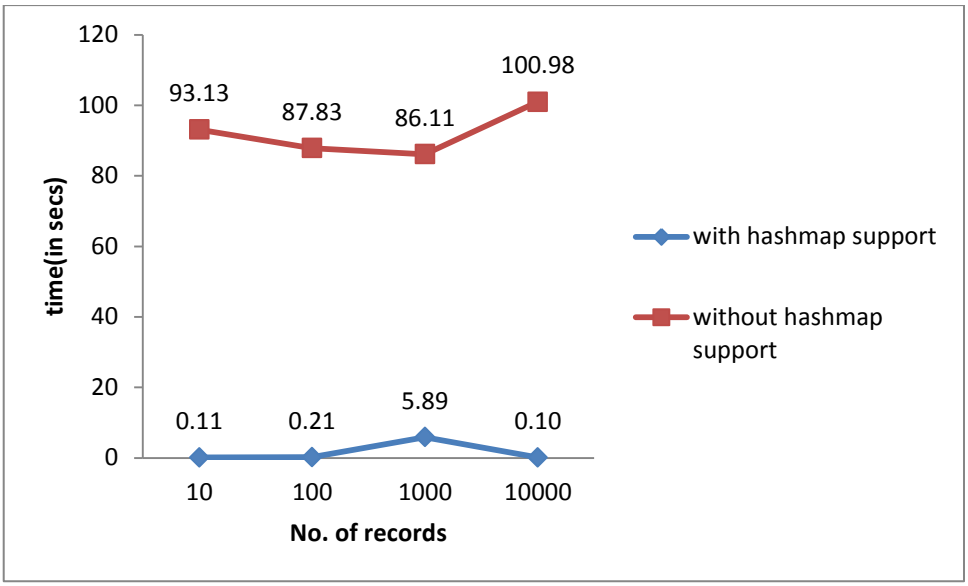


Figure 4.21: Graph of Time vs Records for Query 1

Query 2 entails a scan over five relations with nested sub-queries, equality tests and the *ORDER BY* clause to retrieve data from seven columns. Table 4.3 shows the mean execution time and Figure 4.22 is a line graph of time against the number of records, plotted from the data in Table 4.3.

select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment from part, supplier, partsupp, nation, region where p_partkey = ps_partkey and s_suppkey = ps_suppkey and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = ':3' and ps_supplycost = (select min(ps_supplycost) from partsupp, supplier, nation, region where p_partkey = ps_partkey and s_suppkey = ps_suppkey and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = ':3') order by s_acctbal desc, n_name, s_name, p_partkey;

Table 4.3: Mean Execution Time for Query 2

	Mean Time (in seconds) for Query 2	
No of Records	**with hashmap support**	**without hashmap support**
10	0.018	0.062
100	0.013	0.034
1000	0.020	0.047
10000	0.071	0.111

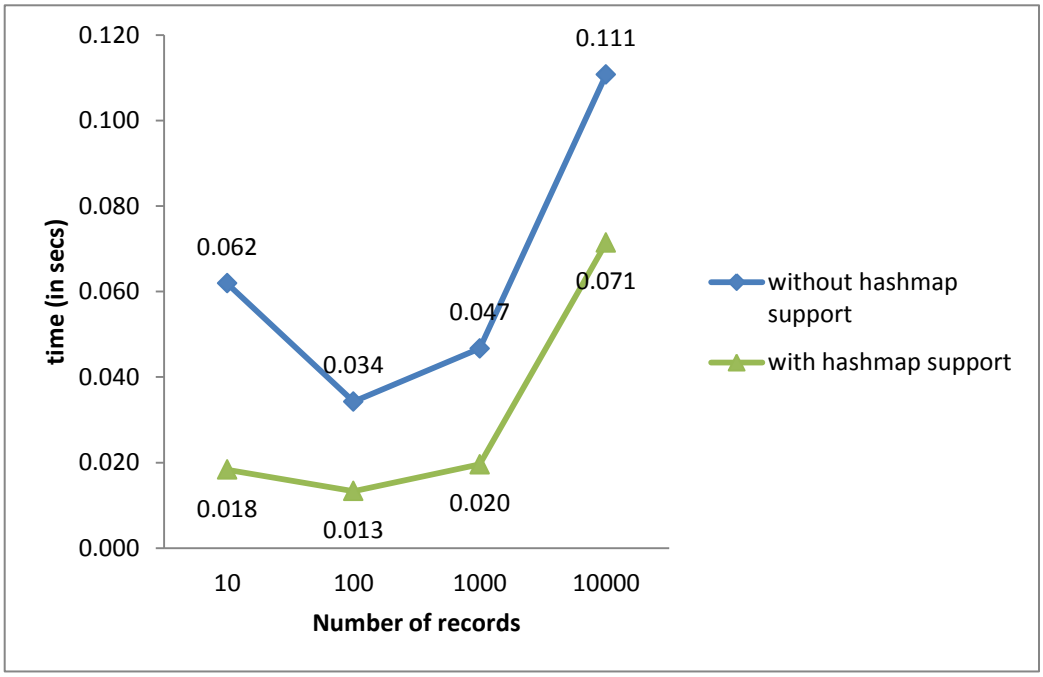


Figure 4.22: Graph of Time vs Records for Query 2

Query 6 selects data from one relation involving one column and computes for aggregate values. Table 4.4 shows the mean execution time and Figure 4.23 is a line graph of time against the number of records, plotted from the data in Table 4.4.

*select sum(l_extendedprice * l_discount) as revenue from lineitem where l_discount between 2 - 0.01 and 2 + 0.01 and l_quantity<3;*

Table 4.4: Mean Execution Time for Query 6

	Mean Time (in seconds) for Query 6	
No of Records	**with hashmap support**	**without hashmap support**
10	0.063	32.321
100	0.048	29.828
1000	0.074	28.013
10000	0.056	27.348

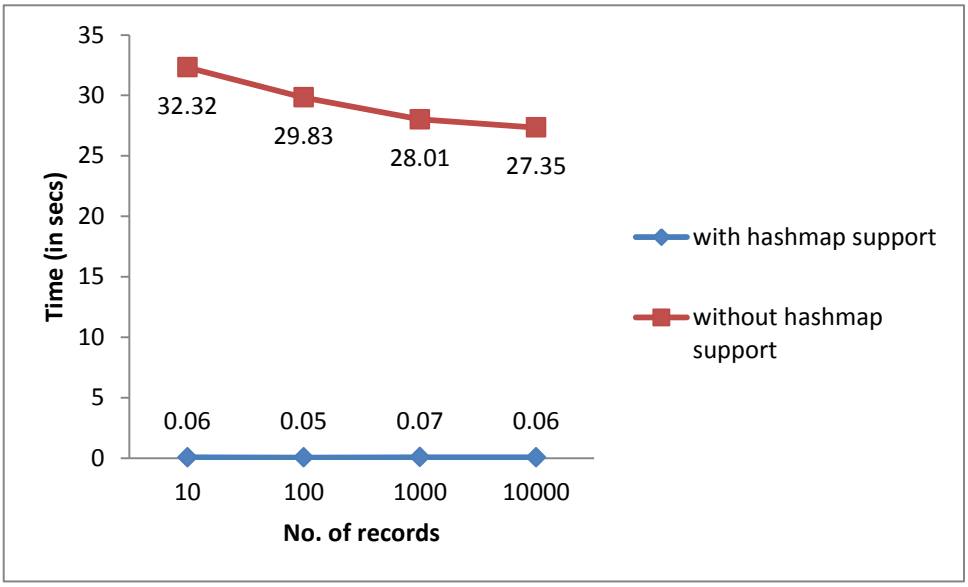


Figure 4.23: Graph of time vs records for query 6

Query 16 traverses two relations involving nested sub queries and computations of equality, pattern matching, the *GROUP BY* and *ORDER BY* clause to retrieve data from four columns. Table 4.5 shows the mean execution time and Figure 4.24 is a line graph of time against the number of records, plotted from the data in Table 4.5.

select p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt from partsupp, part where p_partkey = ps_partkey and p_type not like ':2%' and ps_suppkey not in (select s_suppkey from supplier where s_comment like '%Customer%Complaints%') group by p_brand, p_type, p_size order by supplier_cnt desc, p_brand, p_type, p_size;

Table 4.5: Mean Execution Time for Query 16

	Mean Time (in seconds) for Query 6	
No of Records	**with hashmap support**	**without hashmap support**
10	0.084	1250.133
100	0.078	1283.086
1000	0.031	1211.201
10000	0.062	1182.444

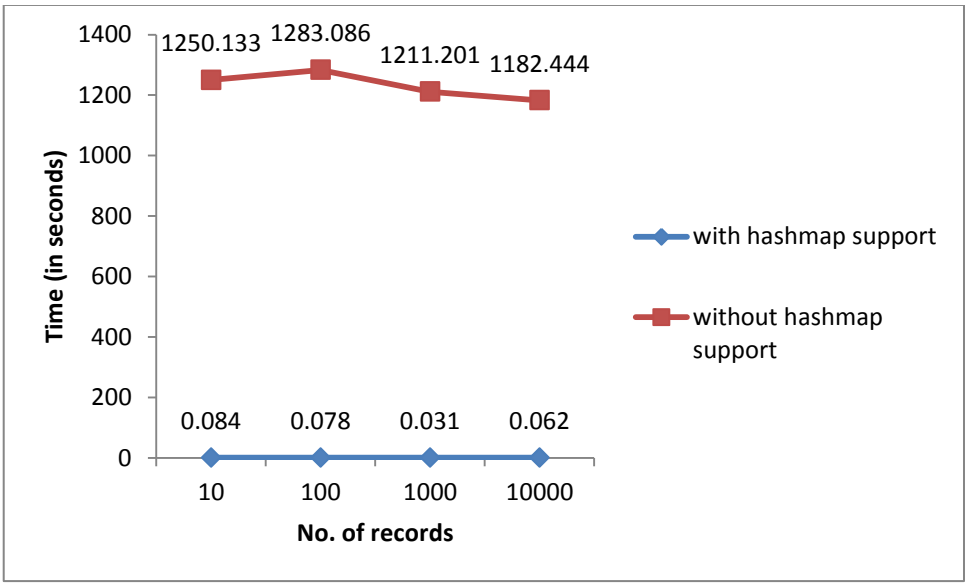


Figure 4.24: Graph of Time vs Records for Query 16

Figure 4.25 is a linear graph plotted with the twenty two TPC-H queries on the x-axis and execution time on the y-axis using base10 log scale, for the raw execution time and the hash map supported method.

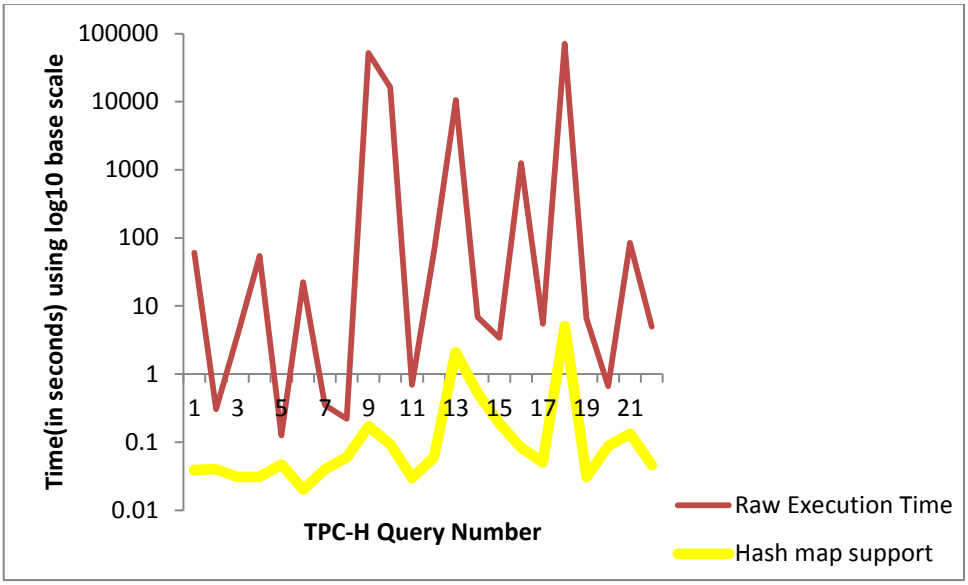


Figure 4.25: Graph of Time vs Queries using Log Scale

4.9 Observation and Discussion

Looking at the scalability for increasing dataset size, Figure 4.21 shows some constancy in time with data size increase for query 1, though at some point (10,000 records), the 'without hash map support' execution time increases. For query 2, Figure 4.22 indicates that time increases linearly with the dataset - O(n), without the hashmap support and with the hash map support primarily because of the nested sub queries. Figure 4.23 for query 6 and Figure 4.24 for query 16, show very constant time for varying data sets using the hash map support.

In answer to the question in section 4.8: "How many of the TPCH queries actually are able to execute on the proposed system?" Figure 4.25 depicts the time it took to execute each TPC-H query, with the hash map support and without the hash map support. The

wide margin between both methods necessitated the use of log scale on the time axis, in other to achieve a better visual rendition.

On a general note, it can be observed that there is a significant variation between the execution time using the traditional method and the time using the proposed method. Also, with increasing number of records, the proposed method maintains a degree of constancy in time.

Some exceptionally large figures were observed in the response time of few queries. This was usually the case when the query involved complex joins on multiple tables, *GROUP BY, HAVING* clause and nested sub queries. This presents where the network becomes a bottleneck which happens because the network bandwidth is lower than the speed at which the tuples are generated by the *SELECT* operator. Resource utilization at the client side is another factor. Besides the exceptions, these modest overheads show that using *secureSQL,* to query over encrypted data is practical in many cases with optimal performance. This clearly implies that our model is not restricted to simple query constructs but is able to handle even complex queries involving nested sub queries and joins.

4.10 Comparison of Methods

Most related works use different approaches and varying parameters to measure their efficiency thus it becomes difficult to construct a head-to-head comparison with their methods. Based on some outlined factors, the comparisons outlined in Table 4.6 are made.

Table 4.6: Comparison of Methods

Author/Method	**Query Type Support**				**Searching Time**	**Encryption Mode/Algorithm**	**No. of TPCH queries handled**	**Key Handling**
	Equality	**Pattern**	**Range**	**Aggregation Functions**				
Hacigumus *et al.*(2002b)	yes	No	Yes	No	O(log n)	Full	2	server
Alhanjouri and Al Derawi (2012)	yes	Yes	No	No	O(1)	full/AES-256 bit	Na	server
Kaul (2013)	yes	No	Yes	Yes	Na	full/multiple	16	client
Proposed method	yes	Yes	Yes	Yes	O(1)	selective/AES-128 bit	20	Client

Using the logical representation of data by 1's and 0's bits to mean true and false, yes and no respectively, Table 4.6 is modified to obtain Table 4.7 for the query type support(yes=1, no=0) and key handling (server=0, client=1). Consequently, a graph is obtained from Table 4.8 and represented as Figure 4.26.

Table 4.7: Comparison of Methods Redefined

Author/Method	**Query Type Support**				**Key Handling**
	Equality	**Pattern**	**Range**	**Aggregation Functions**	
Hacigumus *et al.*(2002b)	1	0	1	0	0
Alhanjouri and Al Derawi(2012)	1	1	0	0	0
Kaul (2013)	1	0	1	1	1
Proposed method	1	1	1	1	1

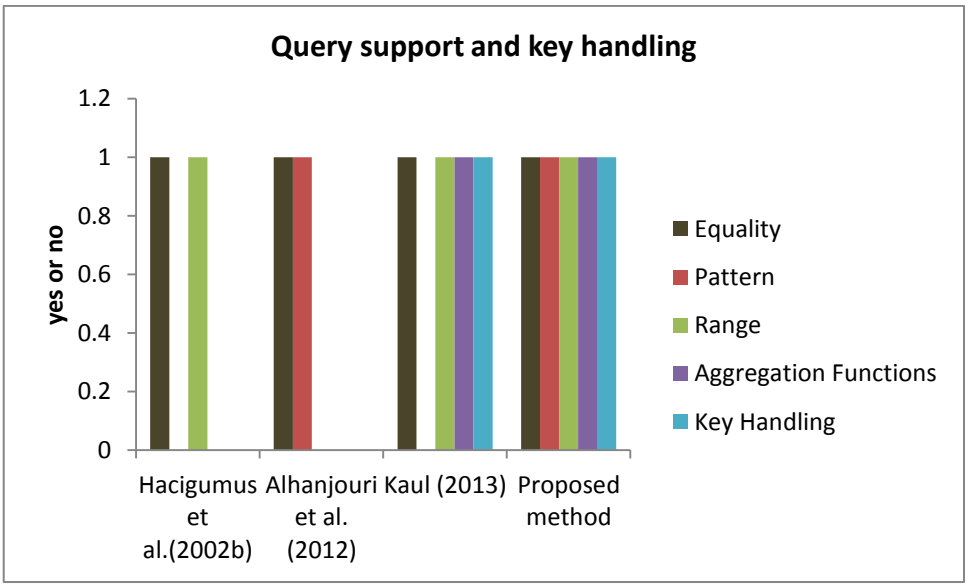


Figure 4.26: Chart of Query Type Support and Key Handling

Figure 4.27 is a chart which represents the number of TPC-H queries handled by each method compared in Table 4.7.

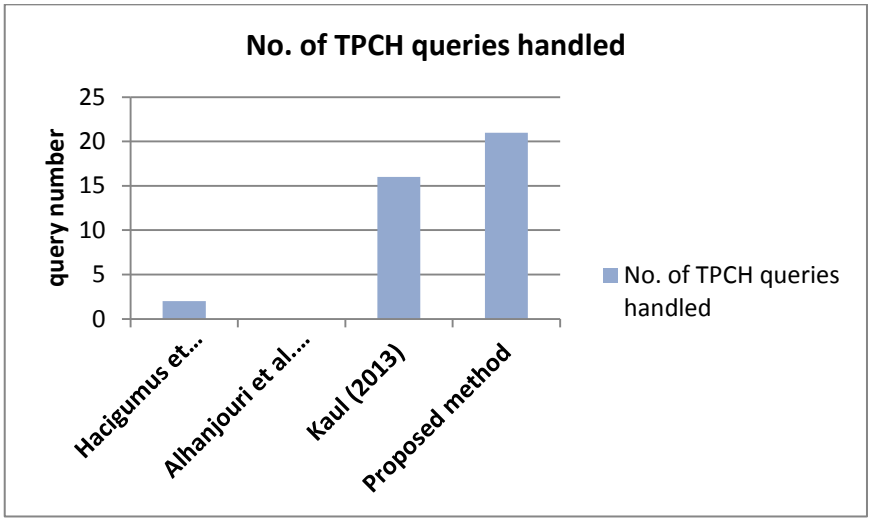


Figure 4.27: Comparison of Queries Handled

Figure 4.28 is a graph of time against records for various time complexities using the big O notation. O(1) shows a better performance as compared with O(log n) and O(n). Hence, the advantage of the proposed method is evident.

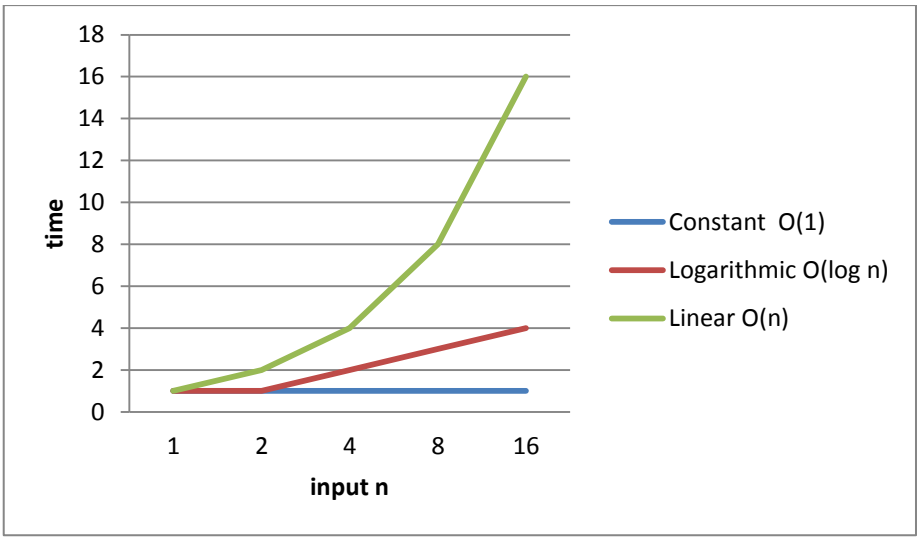


Figure 4.28: Graph of Time vs Number of Records

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 Summary

The efficiency of data retrieval in large outsourced databases, especially as it relates to the privacy of such data has been an open challenge primarily due to the fact that traditional query languages cannot work with encrypted data. Reviewed literature showed that most of the architectural models and encryption techniques proposed to ensure efficient performance of queries as well as privacy of outsourced data have limitations which range from high computational overhead to restricted query computation.

Based on these limitations, this research proposed an architectural framework which focused on enhancing server-side data retrieval and query efficiency through the use of hash map and AES 128-bit encryption algorithm. The implementation of this framework known as *secureSQL* was built on the client-side without any alteration to the DBMS structure. Server-side data retrieval was made possible through a simple graphical user interface to encrypt data before uploading to a cloud server; perform queries and decrypt encrypted values on the fly. A query processing engine was also developed to process encrypted data. Selective attribute-based encryption which was incorporated into the encryption algorithm ensures that computational overhead is minimized.

Observation of the results obtained (refer to section 4.9) show that there is a significant variation between the execution time using the traditional method of decrypting entire records before performing queries and the time using the proposed method. Also, with increasing number of records, the proposed method maintains some degree of constancy

in execution time thereby supporting the O(1) time complexity assertion for the use of the hash map data structure (refer to section 2.13 and section 2.6). A tabular and graphical comparison (refer to section 4.10) with some related works show that the proposed method scores more points.

SecureSQL model guarantees efficiency and is able to execute 20 out of the 22 TPC-H benchmark queries while ensuring privacy. This is proof that it is not restricted to simple query constructs but is able to handle even complex queries involving nested sub queries and joins.

In a nutshell, this research has dealt with relational databases that are hosted on the cloud; how they are secured, the effects of the security on data retrieval and further proposed methods to reduce these effects to a minimal level.

5.2 Conclusion

The recent explosion of digital content ownership has both increased the popularity of data outsourcing and fueled concerns over data security. The need to facilitate storage and processing of large amounts and types of sensitive data is of particular importance in modern enterprise especially where the server is not trusted and client resources are limited. This research worked at eliminating the tradeoff between security and efficient query processing through its *secureSQL* design as is evidenced in the results. The design of an efficient technique to enhance the privacy and performance of SQL queries on cloud databases which was the first objective has been achieved through the combined use of Selective Attribute-Based Encryption, Hash Map structure and AES-128 bit encryption algorithm. The model was implemented using all the implementation tools in section 3.7, thus, the second objective is also achieved.

Finally, the observations in section 4.9 show a wide margin in the query performance using the conventional method and the proposed method. The third objective is also achieved.

As long as the performance degradation due to encryption is under control, users would choose to use outsourced databases to store private information rather than traditional databases.

5.3 Recommendation for future work

Some aspects related to this research could not be investigated as they were out of scope while others were investigated but not implemented due to limitation of time. These aspects have therefore been recommended as future work outlined:

a. Explore further techniques of data retrieval security and efficiency that deal with multimedia databases (images, music and videos) because of the recent explosion of mobile technology and rapid expansion of online media services.
b. Design of an intelligent system that determines which columns are sensitive and automatically encrypts the data.
c. Make the encryption/decryption keys dynamic so that they could be set from the application user interface.
d. With advances in web computing and mobile technology, XML is widely being used as a standard to exchange data over inter-networked heterogeneous systems and web based applications. The hash map method could be extended to develop query execution techniques over the encrypted XML data.
e. Investigate and analyze similar techniques implemented in this work for non-relational databases.

f. A data partitioning function which defines the conditions for data sensitivity can be incorporated to determine the sensitivity of data for certain attributes before applying the encryption function.

REFERENCES

Aderounmu G.A.(2012). "Cloud Computing: A Definition, Challenges and Opportunities". [PowerPoint slides]. Available from: http://www.cpn.gov.ng/?page=show&cat=6&subc=20

Aggarwal G., Bawa M., Ganesan P., Garcia-Molina H, Kenthapadi K., Motwani R. and Xu Y. (2005). Two can keep a Secret: A Distributed Architecture for Secure Database Services. Retrieved August 20, 2014 from: https://database.cs.wisc.edu/cidr/cidr2005/papers/P16.pdf

Agrawal R., Evfimievski A. and Srikant R. (2003). Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 86–97, New York, NY, USA, 2003.

Agrawal R., Kiernan J., Srikant R. and Xu Y. (2004). Order Preserving Encryption for Numeric Data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 563–574, New York, NY, USA, 2004.

Al Tamimi, A. (2003). "Performance Analysis of Data Encryption Algorithms". Retrieved April 23, 2015 from: http://www.cse.wustl.edu/~jain/cse567-06/encryption_perf.htm.

Alhanjouri M. and Al Derawi A.(2012) "A New Method of Query over Encrypted Data in Database using Hash Map", *International Journal of Computer Applications (IJCA)*, **41**(4): 46-51, March 2012. Published by Foundation of Computer Science, NY, USA. Retrieved from: http://research.ijcaonline.org/volume41/number4/pxc3877580.pdf

Badger L., Patt-Corner R., Grance T. and Voas. J. (2011). "DRAFT Cloud Computing Synopsis and Recommendations. *Recommendations of the National Institute of Standards and Technology" (NIST)*. May, 2011. Retrieved from: http://csrc.nist.gov/publications/drafts/800-146/Draft-NIST-SP800-146.pdf

Bell R. (2009). A Beginners guide to Big O notation.[web log post]. Accessed August 2, 2015 from: https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/

Bellovin S.M. (2009). Modes of Operation [Power point slides]. Retrieved September 12, 2014 from: https://www.cs.columbia.edu/~smb/classes/s09/l05.pdf.

Borgmann M., Hahn T., Herfert M., Kunz T., Richter M., Viebeg U. and Vowe S. (2012). *On the Security of Cloud Storage Services*.[Technical report]. Fraunhofer Institute for Secure Information Technology SIT, Germany. Retrieved June 2, 2014 from:

https://www.sit.fraunhofer.de/fileadmin/dokumente/studien_und_technical_reports/Cloud-Storage-Security_a4.pdf

Bouganim L. and Guo Y. (2009). Database Encryption. Encyclopedia of Cryptography and Security, pp.1-9. Retrieved September 15, 2015 from: http://www-smis.inria.fr/~bouganim/Publis/BOUGA_B6_ENC_CRYPT_2009.pdf.

Brinkman R. (2007). Searching in Encrypted Data. (Unpublished Ph.D Thesis). University of Twente, The Netherlands. Retrieved June 2, 2014 from: http://doc.utwente.nl/57852/1/thesis_Brinkman.pdf.

Carver J., VanVoorhis J. and Basili V. (2002). "Understanding the impact of assumptions on experimental validity". [Technical Report] .Retrieved July 5, 2015 from: https://www.cs.umd.edu/~basili/publications/proceedings/P103.pdf

Ciriani V., De Capitani di Vimercati S., and Foresti S. (2010). Combining Fragmentation and Encryption to Protect Privacy in Data Storage. *ACM Transactions on Information and System Security, **13**(3): Article 22.*

Ciriani V., De Capitani di Vimercati S., Foresti S., Jajodia S., Paraboschiz S., and Samarati P. (2009). Fragmentation Design for Efficient Query Execution over Sensitive Distributed Databases. Retrieved March 10, 2014 from: http://spdp.di.unimi.it/papers/icdcs09.pdf

Codd E. F. (1970). A Relational Model of Data for large shared data banks. *Communications of the ACM*, **13**(6):377, 387.

Cowie B. and Irwin B. (2009). An investigation into the field of cryptography and cryptographic protocols. Retrieved August 14, 2014 from: http://www.cs.ru.ac.za/research/g06c5476/Honours/LiteratureReviewCowie.pdf

Cryptography (2014). What are the practical differences between 256-bit, 192-bit, and 128-bit AES encryption?[Web log post]. Accessed June 20, 2015 at: http://crypto.stackexchange.com/questions/20/what-are-the-practical-differences-between-256-bit-192-bit-and-128-bit-aes-enc.

Damiani E., De Capitani di Vimercati S., Jajodia S., Paraboschi S. and Samarati P. (2003). Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), pp. 93-102*.

Dave P. (2009, April 28). SQL SERVER – Introduction to SQL Server Encryption and Symmetric Key Encryption Tutorial with Script [Web log post]. Retrieved on August 18, 2015 from: http://blog.sqlauthority.com/2009/04/28/sql-server-introduction-to-sql-server-encryption-and-symmetric-key-encryption-tutorial-with-script/.

Davida G.I., Wells D.L. and Kam J.B. (1981). A Database Encryption System with Subkeys. *ACM Trans. Database Syst.,* **6**(2):312-328.

Donkena K. and Gannamani S. (2012). Performance Evaluation of Cloud Database and Traditional Database in terms of Response time while retrieving the data. (Unpublished Master's Thesis). Blenkinge Institute of Technology, Sweden. Available June 2, 2015 from: http://docplayer.net/3806742-Performance-evaluation-of-cloud-database-and-traditional-database-in-terms-of-response-time-while-retrieving-the-data.html.

Elmasri R. (2008). *Fundamentals of database systems*. Pearson Education India.

Fernandez E.B., Summers R.C. and Wood C. (1980). *Database Security and Integrity*. Addison-Wesley Massachusetts. Pp. 38, 56-62. Available from: http://www.researchgate.net/publication/44503251_Database_security_and_integrity__Eduardo_B._Fernndez_Rita_C._Summers_Christopher_Wood.

Gentry C. (2009). *A fully homomorphic encryption scheme*. (Unpublished PhD thesis). Stanford University. Retrieved June 2, 2014 from: https://crypto.stanford.edu/craig/craig-thesis.pdf.

Gerardnico (2014, January 23). TPC - TPC-H Decision Support Benchmark Sample Schema [Web log post]. Accessed June 10, 2015 from: http://gerardnico.com/wiki/performance/tpc-h.

Google Cloud Platform documentation (2015). Using Google Cloud SQL. Accessed September 10, 2015 from: https://cloud.google.com/appengine/docs/php/cloud-sql/.

Google Developers Console (2015). Creating a Project - thesisbenchmarking. Accessed September 10, 2015 from: https://console.developers.google.com/project/thesisbenchmarking.

Gray J. (Ed.)(1993). *The Benchmark Handbook for Database and Transaction Systems* (2nd Ed.). Morgan Kaufmann Publishers, San Francisco. Available September 20, 2014 from: http://www.odbms.org/2014/03/benchmark-handbook-1993/

Hacıgümüş H., Iyer B. and Mehrotra S. (2004). Efficient execution of aggregation queries over encrypted relational databases. In *Proceedings of the 9th International Conference on Database Systems for Advanced Applications, Jeju Island, Korea.* Pages 633-650.

Hacıgümüş H., Iyer B., Li C. and Mehrotra S. (2002a). Providing Database as a Service. In *Proceedings of the 18th International Conference on Data Engineering* (*ICDE*'02). Sanjose CA. Pages 29-38. DOI: 10.1109/*ICDE*.2002.994695.

Hacıgümüş H., Iyer B., Li C. and Mehrotra S. (2002b). Executing SQL over encrypted data in the database service provider model. In *SIGMOD Conference*, pages 216–227.

Haidong Ji (2011, March 31). Data generation with TPC-H's dbgen for load testing [Web log post]. Accessed June 23, 2015 from: http://planet.mysql.com/entry/?id=27807.

Halitsch (2014). Implementation of TPC-H schema into MySQL DBMS [Web log post]. Accessed February 27, 2015 from: https://sites.google.com/site/halitsch88/Implementation-TPC-H-schema-into-MySQL-DBMS.

Hore B., Mehrotra S. and Tsudik G. (2004). A privacy-preserving index for range queries. In *Proceedings of the Thirtieth international conference on Very large data bases – **30***: 720–731 VLDB '04. VLDB Endowment.

Ibeh N. (2015, August 27). FRSC, NIMC to harmonize biometric data. Premium Times, September 30, 2015. Accessed September 2, 2015 from: http://www.premiumtimesng.com/news/more-news/189110-frsc-nimc-to-harmonize-biometric-data.html.

Jacob S. (2010). Cryptanalysis of a Fast Encryption Scheme for Databases and of its Variant. IACR Cryptology ePrint Archive 2010: 554. Retrieved June 15, 2015 from: https://eprint.iacr.org/2010/554.pdf

Jensen C.D. (2000). Cryptocache: a secure sharable file cache for roaming users. In *Proc. of the 9th Workshop on ACM SIGOPS* European workshop: beyond the PC: new challenges for the operating system, pages 73-78, Kolding, Denmark, September 2000.

Kamara S. and Lauter K.(2010). Cryptographic cloud storage. In *RLCPS*, 2010

Kantarcioglu M. (2009). Modes of Operation [Power point lecture slides]. Retrieved June 11, 2014 from: https://www.utdallas.edu/~muratk/courses/crypto09s_files/modes.pdf.

Kaul A. (2013). Query Processing in Encrypted Cloud Databases. (Unpublished Master's Thesis). Indian Institute of Science, Bangalore. Retrieved March 25, 2014 from: http://dsl.serc.iisc.ernet.in/publications/thesis/akshar.pdf.

Kim P. (2009). Project/EFIM/TPC-H [Web log post]. Accessed March 22, 2015 from: http://www.pilhokim.com/index.php?title=Project/EFIM/TPC-H.

Kumar G. and Chelikani A. (2011). Analysis of Security Issues in Cloud Based E-learning. (Unpublished Master's Thesis). University of Boras. Retrieved June 2, 2015 from: http://bada.hb.se/bitstream/2320/9271/1/2011MAGI23.pdf.

Lafore R. (1999). *Teach yourself data structures and algorithms in 24 hours*. SAMS Publishing.

Lafourcade P. (2008). Symmetric Encryption et al.[Power point slides]. Retrieved June 20, 2015 from: http://www-verimag.imag.fr/~plafourc/teaching/Symmetric_Encryption_08.pdf.

Larsen G.A. (2004, January 21). Benchmarking Performance of a Query - Part 1 Elapsed Time. Retrieved February 27, 2015 from: http://www.databasejournal.com/features/mssql/article.php/3298411/Benchmarking-Performance-of-a-Query---Part-1-Elapsed-Time.htm.

Lex D., Karen M., Tim G., Inger J. and Daniel F. (2009). *Beginning Oracle SQL*. Apress, USA.

Li J., Sharma N.K. and Szekeres A. (2013). "Performance Analysis of Cloud Relational Database Services". Retrieved from: http://courses.cs.washington.edu/courses/cse544/13sp/final-projects/p18-lijl.pdf

Liu D. (2014). Securing Outsourced Databases in the Cloud. Available on January 28, 2015 from: https://scholar.google.com/citations?user=5l6QNzwAAAAJ&hl=en

Liu J. and Ting L.H. (2010). Dynamic Route Scheduling for optimization of Cloud Database. Presented at the *Intelligent Computing and Integrated Systems (ICISS)* pp. 680-682

Mateljan V., Cisic D. and Ogrizovic D. (2010). Cloud Database-as-a-Service (DaaS). ROI. In *MIPRO, 2010 Proceedings of the 33rd International Convention, NY* pages 1185-1188. IEEE Press.

Mattsson U. T. (2005). Database Encryption – How to balance Security with Performance. Retrieved on June 20, 2014 from: http://hosteddocs.ittoolbox.com/UM070805.pdf

Min-Shiang H. and Wei-Pang Y. (1997). Multilevel Secure Database Encryption with subkeys. Data and Knowledge Engineering 22, 117-131. Retrieved on August 12, 2014 from: http://isrc.ccs.asia.edu.tw/www/myjournal/P007.pdf

Mykletun E. and Tsudik G. (2006). Aggregation Queries in the Database-as-a-Service Model. Retrieved August 20, 2014 from: http://www.ics.uci.edu/~gts/paps/mt06.pdf

MySQL 5.5 reference manual documentation (2015). Measuring the Speed of Expressions and Functions. Retrieved July 20, 2015 from: http://dev.mysql.com/doc/refman/5.5/en/select-benchmarking.html.

National Institute of Standards and Technology (2011). The NIST definition of cloud computing. Retrieved from: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf on September 20, 2014.

Obiniyi A.A., Dzer R. M. and Abdullahi S.E. (2015). "Balancing Query Performance and Security on Relational Cloud Database: An Architecture". *International Journal of Computer Applications (0975 – 8887)* ***118(****15): 34 – 37.*

Ozsoyoglu G., Singer D.A. and Chung S.S. (2003). Anti-Tamper Databases: Querying Encrypted Databases. Available from: http://www.researchgate.net/publication/46298787_Anti-tamper_databases_Querying_encrypted_databases

Panda (2013, August 21). Use TPC-H to create large test data sets [Web log post]. Retrieved July 20, 2015 from:http://www.sqlpanda.com/2013/08/use-tpc-h-to-create-large-test-data-sets.html.

Phan T.A.M. (2013). Cloud Databases for Internet-of Things Data. (Unpublished Master's Thesis). Technical University of Denmark. Available December 12, 2013 from: http://www.internetofthings.fi/publications

Popa R.A., Redfield C.M.S., Zeldovich N., and Balakrishnan H. (2011). CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011),* Cascais, Portugal, October 2011.

Raybourn T. (2013). Bucketization Techniques for encrypted databases: Quantifying the impact of Query Distributions.(Unpublished Master's Thesis). Bowling Greenstate University. Available December 2, 2014 from: https://etd.ohiolink.edu/ap/10?0::NO:10:P10_ACCESSION_NUM:bgsu1363638271.

Rogaway P. (2011, February 10). Evaluation of Some Blockcipher Modes of Operation [Report]. Retrieved June 11, 2014 from: http://web.cs.ucdavis.edu/~rogaway/papers/modes.pdf.

Roshan R.R., Bashaha I.N. and Zahra M. (2013). A Survey on Querying Encrypted Data for Database as a Service. Retrieved from: http://dx.doi.org/10.1109/CyberC.2013.12

Sakhi I. (2012). Database Security in the Cloud. (Unpublished Master's Thesis). Available September 20, 2015 from: http://bbs.chinacloud.cn/showtopic-14772.aspx

Schneier B. (1995). *Applied cryptography (2nd ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc. New York, NY, USA. Pp. 277-280.

Sharma M., Chaudhary A. and Kumar S. (2013). Query Processing and Performance and Searching over encrypted data by using an Efficient Algorithm. Retrieved on June 20, 2014 from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.303.6449

Sharma R. and Trivedi R.K. (2014). Literature review: Cloud Computing –Security Issues, Solution and Technologies. *International Journal of Engineering Research.* **3**(4): 221-225. Retrieved September 12, 2014 from: http://www.ijer.in/ijer/publication/v3s4/IJER_2014_408.pdf

Shmueli E., Vaisenberg R., Elovici Y. and Glezer C. (2009). Database encryption: An overview of contemporary challenges and design considerations. SIGMOD Record, September 2009 (Vol. 38, No. 3). Retrieved on November 4, 2014 from: https://www.researchgate.net/publication/220416426_Database_encryption_an_overview_of_contemporary_challenges_and_design_considerations.

Simmons G.J. (1979). Symmetric and Asymmetric Encryption. [Computing Surveys]. **11**(4). Retrieved March 11, 2014 from: http://www.engr.uconn.edu/~zshi/course/cse5302/ref/CSurveys_SymmAsymEncrypt-simmons.pdf

Sion R. (2008). Towards Secure Data Outsourcing. In Gertz E., Jajodia M.E. and Sushil R. *Handbook of Database Security*. (pp 137-161). Springer US. Available from: http://dx.doi.org/10.1007/978-0-387-48533-1_6

Song D.X., Wagner D. and Perrig A. (2000). Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, *Berkeley*, *CA*, *USA*. pages 44-55.

Sqlserverbible (2013). Retrieved May 24, 2014 from: http://www.sqlserverbible.com/files/databasedesignroi.pdf

StackExchange – Information Security (2012). Why most people use 256 bit encryption instead of 128 bit? [Online forum post]. Accessed August 20, 2015 from: http://security.stackexchange.com/questions/14068/why-most-people-use-256-bit-encryption-instead-of-128-bit.

Stackoverflow (2013). AES128 vs AES256 using bruteforce [Online forum post]. Accessed June 10, 2015 from: http://stackoverflow.com/questions/18847580/aes128-vs-aes256-using-bruteforce.

Stephen T.M., Kaashoek F. Madden S. and Zeldovich N. (2013). Processing Analytical Queries over Encrypted Data. *MIT CSAIL*

Stewart J.M., Chapple M. and Gibson D. (2012). *CISSP: Certified Information Systems Security Professional Study Guide, 6th Edition.* Wiley Publishing.

Tantawy M. M. (2010). Cryptography (One Day Cryptography Tutorial). [Powerpoint presentation] Retrieved on June 21, 2014 from: http://www.slideshare.net/techdude/unit-3-cryptography.

TechyHelp (2014, June 29). How Java Hashmap works. [Video tutorial] Retrieved December 10, 2014 from: https://www.youtube.com/watch?v=TQls-N_TqMw.

Townsend P. (2009). Intro to AES Encryption-part 3. [Video tutorial] Accessed November 22, 2014 from: https://www.youtube.com/watch?v=Xna-qBWgn90

Transaction Processing Performance Council (2014): Benchmarking TPC-H. Available from: http://www.tpc.org.

Velte A.T., Velte T.J. and Elsenpeter R. (2010). *Cloud Computing: A practical Approach*. The McGraw-Hill Companies.

Wang Z, Wang W. and Shi B. (2005). "Storage and Query over Encrypted Character and Numerical Data in Database." *Proceedings of the fifth International Conference on Computer and Information Technology,* pp.77-81.

Webopedia (2012). Cloud Database. [Web log post]. Accessed September 20, 2014 from: http://www.webopedia.com/TERM/c/cloud-database.html

William F.B. and Ricardo B. (1992). *Information Retrieval Data Structures and Algorithms.* New Jersey. Prentice Hall. Pp 23-28.

Xiao Z and Xiao Y. (2014). "Security & Privacy in cloud computing," *Communications Surveys & Tutorials, IEEE, (99): 1-17.* Available September 2, 2014) from: http://www.cpn.gov.ng/?page=show&cat=6&subc=20.

APPENDIX: SAMPLE PROGRAM CODE

Code that sets the AES key

```
public class AES {

   private static SecretKeySpec secretKey;
   private static byte[] key;
   private static String decryptedString;
   private static String encryptedString;

   public static void setKey(String myKey) {

      MessageDigest sha = null;
      try {
         key = myKey.getBytes("UTF-8");
         System.out.println(key.length);
         sha = MessageDigest.getInstance("SHA-1");
         key = sha.digest(key);
         key = Arrays.copyOf(key, 16); // use only first 128 bit
         System.out.println(key.length);
         System.out.println(new String(key, "UTF-8"));
         secretKey = new SecretKeySpec(key, "AES");
      } catch (NoSuchAlgorithmException | UnsupportedEncodingException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }
   }
```

Code for batch encryption

```
private void jButtonEncryptActionPerformed(java.awt.event.ActionEvent evt) {
      // TODO add your handling code here:
      String table = this.jTextFieldTableName.getText();
      String column = this.jTextFieldColumn.getText();
      String pk = this.jTextFieldPKField.getText();
      if (null != pk && null != column && null != table) {
         // we'll collect all records from this table
         // then run an update to encrypt them
         String sql = new StringBuilder("SELECT
").append(column).append(",").append(pk).append(" FROM ").append(table).toString();
         SQLHelper helper = new SQLHelper();
         SQLHelper helperUpdater = new SQLHelper();
         // set encryption key
         AES.setKey("5201ASD");
         try {
            Statement stmt = helper.getConnection().createStatement();
            ResultSet rs = stmt.executeQuery(sql);
            Statement stmtUpdater = helperUpdater.getConnection().createStatement();
```

```
            while (rs.next()) {
               // run update on this field
               AES.encrypt(rs.getString(column));
               String encryptedValue = AES.getEncryptedString();
               String pkVal = rs.getString(pk);
               stmtUpdater.executeUpdate(
                  new StringBuilder("UPDATE ")
                  .append(table)
                  .append(" SET ")
                  .append(column)
                  .append("='")
                  .append(encryptedValue)
                  .append("' WHERE ")
                  .append(pk)
                  .append("='")
                  .append(pkVal)
                  .append("' LIMIT 1")
                  .toString());
               System.out.println(rs.getString(column) + ":PK=" + rs.getString(pk));
            }
         } catch (SQLException ex) {
            Logger.getLogger(BatchEncrypt.class.getName()).log(Level.SEVERE,    null,
ex);
         } finally {
            helper.closeConnection();
            helperUpdater.closeConnection();
         }
         JOptionPane.showMessageDialog(this, "Operation completed!");
         // close this dialog
         setVisible(false);
      }
   }
```

Code for the hash map

```
public SecureSQL() {
      initComponents();
      // initialize hashmap
      hashMap = new TreeMap();
      SQLHelper helper = new SQLHelper();
      // we're clear to try
      try {
         // initialize a hashmap key value pair
         String query = "SELECT K FROM vw_BulkMap LIMIT 1000";
         ResultSet rs = helper.executeQuery(query, null, false);
         GenericTableModel model = new GenericTableModel(rs, helper.getTokens(),
true, true);
         this.jTableHashMap.setModel(model);
         // acquire newly loaded hashmap
```

```
            this.hashMap = model.getHashMap();
            sb.append("Hashmap initialized...");
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            helper.closeConnection();
        }
    }
```