An application to provide an interface to a mySQL database located in the cloud featuring data privacy

by

Andres Avelino Campos Sainz

A Project

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

March 2013

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the project submitted by

Andres Avelino Campos Sainz

Project Title: An application to provide an interface to a mySQL database located in the cloud featuring data privacy

Date of Final Oral Examination: 13 March 2013

The following individuals read and discussed the thesis submitted by the student Andres Avelino Campos Sainz, and they evaluated his presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Jyh-haw Yeh, Ph.D.	Chair, Supervisory Committee
Tim Andersen, Ph.D.	Member, Supervisory Committee
Alark Joshi, Ph.D.	Member, Supervisory Committee

The final reading approval of the thesis was granted by Jyh-haw Yeh, Ph.D., Chair of the Supervisory Committee. The thesis was approved for the Graduate College by John R. Pelton, Ph.D., Dean of the Graduate College.

ABSTRACT

With the increased tendency to allocate massive amounts of data in the cloud, it is imperative to protect the information from being used for any other purpose than the one intended by the actual owner(s). Databases located in the cloud are especially prone to sniffing and/or spoofing attacks given the ordered nature of its data fields, which are readily available for fast retrieval with short command queries. Using data encryption it is possible to hide the content or semantic meaning of the information, providing in this way, data privacy.

Other previous attempts to encrypt all the contents from a mySQL database, leave the data in a secure state but then such data is no longer useful to perform any updates or queries using conventional mySQL scripts. Then it becomes necessary to decrypt the whole database before being able to access the information again resulting in an inefficient and inconvenient solution.

Motivated primarily by the challenge of this problem, this project consisted on implementing an application capable of not only encrypting the whole database during setup and updates, but also, the encrypted data can be accessed seamlessly by a Graphical User Interface application, interpreting any returning data; this is achieved by means of carefully encoded queries, using various encryption and decryption methods in such a way that the user of the application will not even encounter the difference between using a non-secure database versus the mySQL secure database implemented in this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABREVIATIONS

GUI	Graphical User Interface.
RDS	Relational Database management System.
SQL	Scripting Query Language.
mySQL	Relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases.
AWS	Amazon Web Services.
AES	Advanced Encryption System.
URL	Uniform (or universal) resource location.
SHDB	Semantic Hiding Database
JDBC	Java Database Connector
NC	Numeric Comparison only
NCA	Numeric Comparison and Arithmetic
SC	String Comparison only
SCS	String Comparison and Substring matching
JVM	Java Virtual Machine

CHAPTER 1: INTRODUCTION

With the increased tendency to allocate massive amounts of data in the cloud, it has become very important to protect the information from being accessed by any third party user different than the actual owner(s) of the information. We have to recognize that by using any cloud database service we are trusting that a very considerable amount of sensitive information will be handled safely by people that we do not know, and we also have to consider that the information in the database will be replicated through many servers in several unknown locations around the world. For this reason, databases located in the cloud are especially prone to sniffing and/or spoofing attacks given the ordered nature of its data fields, which are readily available for fast retrieval with mySQL queries. Using data encryption it is possible to hide the content or semantic meaning of the information, however, performing encryption into a mySQL database, modifies the data semantics in such a way that it no longer makes sense to attempt queries with the conventional mySQL expressions that would be used originally, not only because the information in the database is encrypted but also because the original characteristics of the information can be lost if the encryption is not done following previously established rules and also if the queries are not transformed to use encrypted data as well. This is a challenging problem; however we designed the necessary algorithms to handle this complex situation.

Now, we not only wanted to come up with a application that accomplishes data privacy for databases located in the cloud, supporting conventional mySQL update and query script operations, but it was very important that this application still could be used easily, hiding all the complexity required to provide the privacy feature. We recognized that it would not be convenient to come up with a solution that implies running a program from the command console and required elaborated function calls and parameters to be typed in the command line, instead, a Graphic User Interface (GUI) needed to be implemented to minimize any hassle during use, so by implementing the application with a GUI we wanted to offer an easy and friendly interface capable of laying out several functions in a single screen, which can be controlled intuitively and that required minimal skills from the average user, therefore enhancing productivity and providing as much transparency as possible. At the end the goal is to ensure that a user does not need to be aware of the encryption and decryption processes occurring at the background during the mySQL queries, updates or any other auxiliary operation available from the application.

Finally, we also wanted to restrict the use of the application only to users with permissions previously granted, providing in this way enhanced security, so even before the application can be started, we wanted to perform authentication via login and password, also an URL needs to be specified, indicating the cloud service that we want to use.

For our application we use different types of encryptions like a Fully Homomorphic Encryption algorithm (FHE), the popular Advanced Encryption System (AES), and other customized techniques to perform ordered encryption.

1.1 Organization of Project Report

The rest of the Project Report is organized as follows. Chapter 2 describes related work describing previous attempts to provide data privacy and its pitfalls. Chapter 3 details the motivation and objectives of this Project, it also explains the overall design for the solution being implemented from a generalized perspective, going down to more detailed information like the design and layout of objects, the most relevant methods and its mutual interactions, and finally provides a comprehensive summary of the functionality. Chapter 4 covers the actual implementation, describing in detail the work that was completed, the challenges that were faced, important considerations, results and also describes the future work that can be done to include enhanced features for this application.

CHAPTER 2 RELATED WORK

2.1 Cloud Database Encryption

Previous attempts to provide data privacy in relational databases have existed before, however they do not leave the database in an active state. The reason is that such strategies consist on grabbing the whole database content, and perform a single large encryption operation, where the semantic content is totally lost and renders the database useless for any kind of query or update scripting operation. Then this kind of operation is barely good enough to store the database in a secure mode leaving the database disabled for any kind of operation, this is known as security for data at rest [1]. When a user or administrator needs to access the actual content, the database is no longer available, so, in order to enable the database for use, another single large decryption operation needs to be completed, to return the information to its original state, that is, the database needs to come back to a non secure mode, in that moment, the database is subject once again to sniffing attacks from any person gaining access to the information, like for example, a malicious database administrator or a person in charge of the server maintenance. The simple fact that the whole database has to be encrypted and decrypted continuously is inconvenient enough, and still we have the disadvantage that the database is not always semantically hidden. The attacker can simply wait for the moment that the database content is exposed by the decryption to initiate an intrusion like a sniffing or spoofing attack.

2.2 mySQL Graphic User Interfaces

Currently mySQL offers a Graphical User Interface (GUI) to interact to relational databases; however it takes some time to get familiar with the interface since it is not very intuitive. It is very common to find users that do not know where they need to locate their .unl files for mySQL to access them and use them when transferring information to a new database instance. In general, it seems like the most common operations like editing, import and export, etc., are not quite handy for the user.

Since for this project we needed to implement our own Graphical User Interface, we had the opportunity to improve around these areas. Our GUI includes an always present list of files and a very efficient file navigation tool that allows switching to different file locations with a single mouse click. Also importing and exporting data is always accessible without having to explore menu items to find these options.

CHAPTER 3: PROJECT DESIGN

3.1 Motivation and functional requirements

The main purpose of this project is to demonstrate that it is possible to perform encryption to a mySQL database (regardless of its location, size and semantic content) and we can still being able to manage the database availability for active use with any kind of queries and updates using the standard mySQL scripting language. In general, by using the application created in this project, we provide enough transparency to the user so that intrinsic operations like encryption and decryptions are totally abstracted so that users can still have a normal interaction like they would do with any other mySQL application, not containing the privacy feature (semantic hiding).

The application needed to cover three main pieces of functionality, which actually were implemented as individual modules which interact each other by simple interfaces, allowing easy function scalability, maintenance, re-usage of methods and a code organization that does not require too much effort to be followed by a programmer familiar with object oriented programming (like Java and C++).

Each of the three major areas of functionality achieves a corresponding goal, as explained below:

- **Connectivity & Authentication:** The user is required to provide an URL representing the address of the mySQL cloud service as well as a user name or login

name and a password (hidden from view when typed). This information is collected by means of a simple GUI.

- **Graphical User Interface:** Once that access is granted, a larger GUI is built and presented to the user. Hereafter this is referred as the "Main GUI". This one consists of a large window containing several objects like text areas, buttons, menus, text fields and message dialogs that provide all the necessary functionality so the user can run a diverse range of operations ranging from queries and updates, to edition, data sorting, data import and export, etc., all available from the same screen.
- **Encryption and Decryption:** Totally hidden from the GUI, the user is unaware from the internal encryption and decryption process occurring at the background during all kinds of mySQL transactions. This is the more complex piece of implementation in this project. Basically, once that the user has gained access to the Main GUI, and starts sending scripts to the mySQL server (either located in the cloud or in a local machine), the application takes care of translating every single mySQL expression into an encrypted version. The encrypted expression then, goes to the mySQL server and is interpreted at the other end (cloud or local server) allowing for the corresponding data to be updated inside of the database or retrieved to our application.

The following figure shows the overall design, covering these three major areas or implementation:

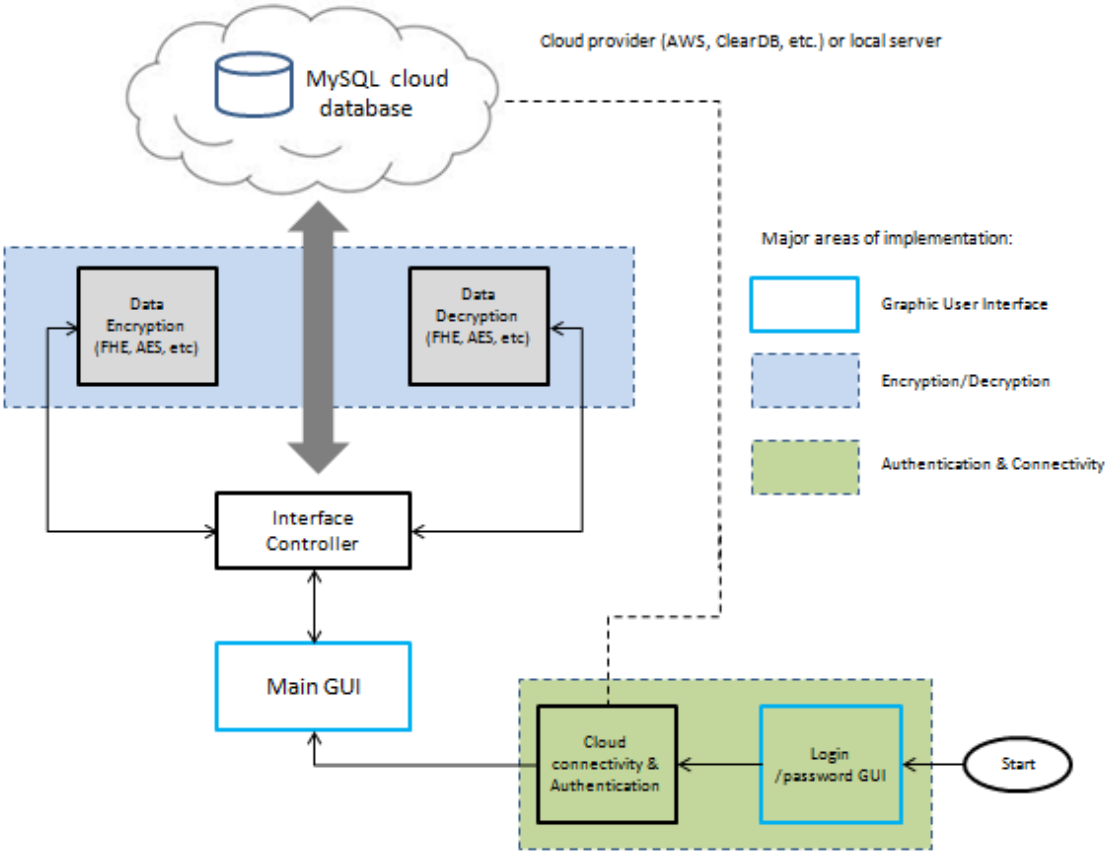

Figure 3.1. SHDB overall design

3.2 Code organization

The previous section presented a rough overview of the three major areas of implementation that satisfy the requirements of this project. However, in this section we take a closer look to the code organization by showing some block diagrams and its mutual interaction inside of the application. The Figure 3.2 below represents the design for the authentication and connectivity module.

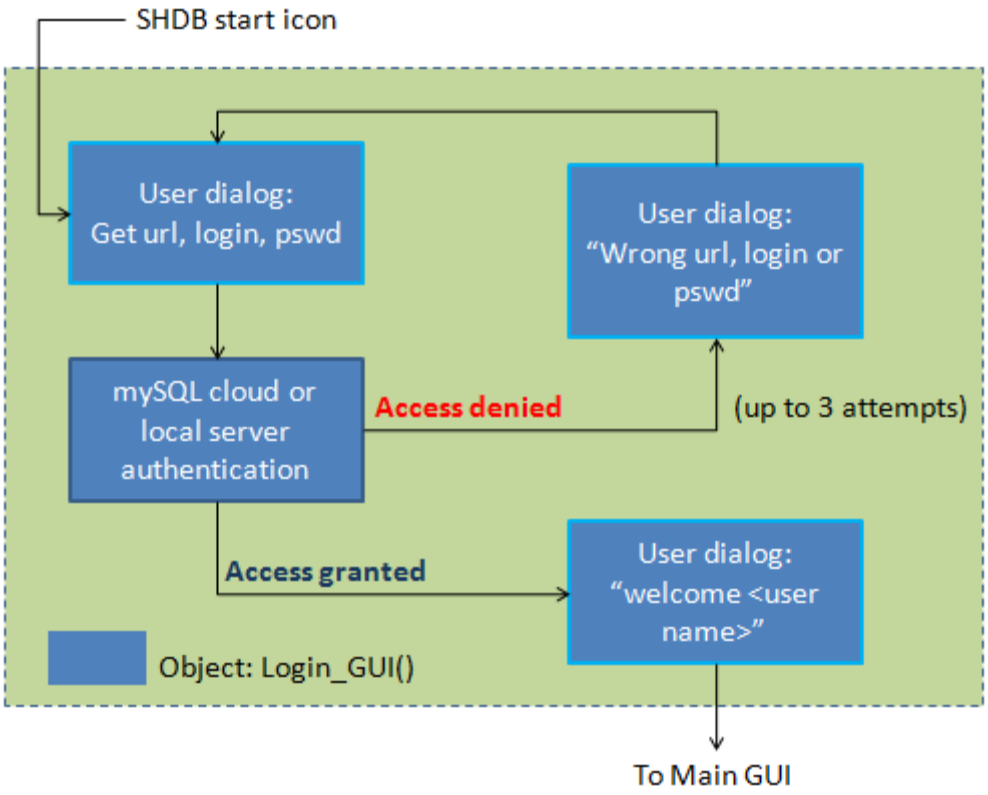

Figure 3.2. Authentication and connectivity module.

When a user initiates the application by double clicking on the SHDB start icon, an initial screen or user dialog appears, this user dialog will collect from the user three values, the URL (or address of the server that provides the mySQL database service), a user login name (for a user previously registered by an administrator) and a password field (also previously created by an administrator) which hides any typed value from view with fixed characters. Once that the user clicks on a "OK" button, the connection and authentication is established with the server. For the case of an invalid URL, login name and/or password, two more chances are provided to the user, otherwise the application is closed. Figure 3.3 shows the preliminary design for the login user interface, this is very similar to the actual screen look and feel achieved after implementation.

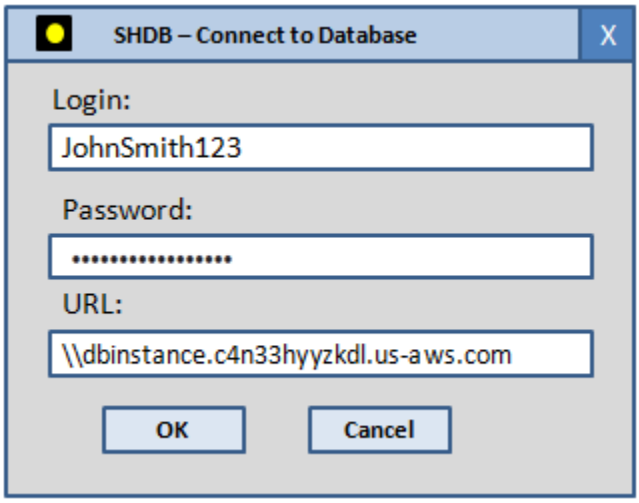

Figure 3.3. Design of the login user interface, showing prompts for login name, password and URL.

When users are to be added for access into a certain mySQL provider, the administrator needs to go to the mySQL cloud management console and define the new user name, password as well as permissions for the new user. Other providers require that a CID is defined for a determined range of IP address which can consist of several lists of ranges. The details of the management console provided by a certain mySQL cloud provider is out of the scope of this document, but information can be found at the Amazon Web Services (AWS) [2] and/or the ClearDB websites [3].

The next important block in this design is the Main GUI. This screen is shown only when a user has been successfully authenticated and it always comes with a welcome message for the user, confirming the user name and an OK button This screen offers the necessary interface to the user in order to facilitate queries and updates (according to the permissions granted by an administrator) as well as other auxiliary functions. Please refer

to Figure 3.4 for an image of the preliminary design (which is slightly different from the actual implementation).

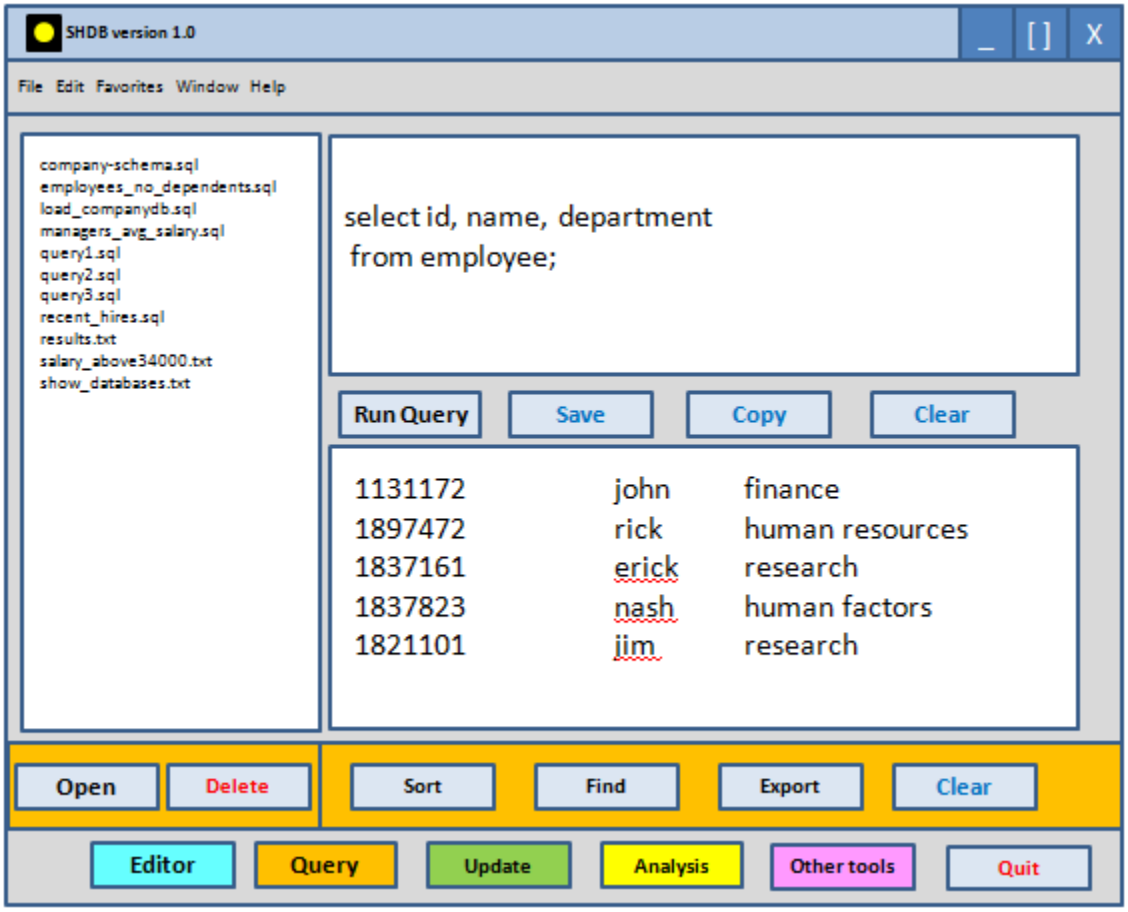

Figure 3.4. Main GUI design.

The main GUI is initially instantiated by the Interface Controller shown in Figure 3.1, and is highly interconnected to other parts of the code, for every button, menu item, text area, text field and user dialog, and action event is generated. This event is handled in such a way that for every one of them a different method from the Interface Controller is called. The most general case is for the update and query processes, which can be described as calling a method that processes the user input from the input text area, followed for some processing and collection of results which then are displayed in the

results window, however not all operations need to access the Interface Controller, that is the case of the User Dialogs which are used extensively to provide an adequate user experience (error messaging, setup of a new database, data exporting, file management, path navigation, help, etc).

Since the main focus of this project is for special handling of mySQL queries and updates we explain such areas in more detail. Other miscellaneous features are covered with less detail.

A clear distinction can be made between update and query mySQL operations, especially from the perspective of a Java DataBase Connection (JDBC) implementation. Examples of update operations are the following:

- Drop schema if exists
- Create schema
- Load database

At the other hand, examples of query operations are:

- Show databases
- Show tables
- Select <item(s)> from <table(s)> where <condition(s)>

The distinction is necessary because an update and a query need to be handled very differently in the code. Both need encryption and decryption operations in the background, also both require of a preliminary parsing process (interpretation of commands, parameters, equalities and nesting in the expressions) but the JDBC library

treats them in different ways, so in summary different functions from this library are required for an update and for a query.

3.2.1 Update operations

When the user presses the Run Update button, a String Tokenizer is started. Every single word in the mySQL expression typed by the user is examined and a decision is taken on what actions to perform based on the contents of the expression. Next, we cover the most typical update operations required to setup a database.

CREATE SCHEMA and CREATE TABLE. This is typically the first kind of mySQL script that a user or administrator needs to execute when setting up a new database instance. However, it is not composed of a single expression, but many of them. An example is provided below:

```
drop schema if exists `company`;
create schema `company`;
use `company`;

CREATE TABLE department (
      dname  CHAR(15) NOT NULL,
      dnumber INT NOT NULL,
      mgrssn  CHAR(9) NOT NULL,
      mgrstartdate DATE,
      PRIMARY KEY (dnumber),
      UNIQUE (dname)
);

CREATE TABLE employee   (
      fname CHAR(15) NOT NULL,
      minit   CHAR,
      lname   CHAR(15) NOT NULL,
      ssn CHAR(9) NOT NULL,
      bdate   DATE,
      address CHAR(30),
      sex CHAR,
      salary  DECIMAL(10,2),
      superssn    CHAR(9),
      dno INT NOT NULL,
      PRIMARY KEY (ssn)
```

```
);

CREATE TABLE dept_locations (
      dnumber INT NOT NULL,
      dlocation    CHAR(15)    NOT NULL,
      PRIMARY KEY (dnumber, dlocation)

);

CREATE TABLE project (
      pname CHAR(15) NOT NULL,
      pnumber INT NOT NULL,
      plocation   CHAR(15),
      dnum    INT NOT NULL,
      PRIMARY KEY (pnumber),
      UNIQUE (pname)
);

CREATE TABLE works_on (
        essn  CHAR(9) NOT NULL,
        pno INT NOT NULL,
        hours   DECIMAL(3,1)    NOT NULL,
      PRIMARY KEY (essn, pno)
);

CREATE TABLE dependent (
        essn  CHAR(9) NOT NULL,
      dependent_name  CHAR(15)  NOT NULL,
      sex CHAR,
      bdate DATE,
      relationship CHAR(8),
      PRIMARY KEY (essn, dependent_name)
);
```

Table 3.1. Example of typical mySQL expressions used to create a new database instance, in this case for a database named ‘company’.

In a case like this, the application processes the different expressions one by one from top to bottom. For this example all the expressions are translated into new expressions that look like the new expressions shown in Table 2 below:

```
drop schema if exists `LUSInWQxFWbj79xu5VL2tg==`;

create schema `LUSInWQxFWbj79xu5VL2tg==`;
```

```
use `LUSInWQxFWbj79xu5VL2tg==`;

CREATE TABLE `latCaFEi7UA57y7KfeHc2Q==` (
`CvblhUbRIbMn4yJhMinGjA==` CHAR(128) NOT NULL ,
`hy0yWjVQ2SaGswqsnWy7Lw==`  BIGINT  NOT NULL ,
`U81Z8kLh7DHXLej1+uHQlw==` CHAR(128) NOT NULL ,
`k0LJOl83MB2LobovYJLuwQ==` BIGINT ,
 PRIMARY KEY  (`hy0yWjVQ2SaGswqsnWy7Lw==`),
 UNIQUE  (`CvblhUbRIbMn4yJhMinGjA==`))ENGINE=InnoDB;

CREATE TABLE `1m4t1hSG+qIBhoxiwpawJw==` (
`e5Va3+bSOoxaWvrSNet3Dw==` CHAR(128) NOT NULL ,
`WkPiUzA5hT5pqJY668lyWg==` CHAR(128) ,
`qobZJ/A0EeEHwlxbld+kaQ==` CHAR(128) NOT NULL ,
`ZOTDsz7gk1P5Yhz+ksVPcA==` CHAR(128) NOT NULL ,
`CJwFz4IgnoxFlniA7edCMg==` BIGINT ,
`Vl8Kf5zCdSHVFIqHfFBz8g==` CHAR(128) ,
`QFiUc+dUiTYetY6zgmJnCA==` CHAR(128) ,
`TO/L7fedob0d+gkoE1/19g==` DEC(65,2) ,
`DtDIe5O09tikVOezObCfEQ==` CHAR(128) ,
`j/80kIzc3J5QWjv35ISKbA==`  BIGINT  NOT NULL ,
 PRIMARY KEY  (`ZOTDsz7gk1P5Yhz+ksVPcA==`))ENGINE=InnoDB;

CREATE TABLE `/6+zaceU2bmo1qU26UiJ9w==` (
`hy0yWjVQ2SaGswqsnWy7Lw==`  BIGINT  NOT NULL ,
`6qJjwpqSesSb45a7uHzwkw==` CHAR(128) NOT NULL ,
PRIMARY KEY  (`hy0yWjVQ2SaGswqsnWy7Lw==`,
`6qJjwpqSesSb45a7uHzwkw==`))ENGINE=InnoDB;

CREATE TABLE `27f0ba7J9+Pt7+XslTGVzg==` (
`t0kB3YPAjOF/868/W7q32w==` CHAR(128) NOT NULL ,
`V8GeHzVDL2B47CtJo0SxdQ==`  BIGINT  NOT NULL ,
`T6yZXSg3o4dzc9Ar33e8Ag==` CHAR(128) ,
`ei6KZldLywvE6ycmSk2BfQ==`  BIGINT  NOT NULL ,
 PRIMARY KEY  (`V8GeHzVDL2B47CtJo0SxdQ==`),
 UNIQUE  (`t0kB3YPAjOF/868/W7q32w==`))ENGINE=InnoDB;

CREATE TABLE `W57f8/oxQe1HWv+9TQHSqg==` (
`GC/AFI9hZ7X4LIa8Txme6w==` CHAR(128) NOT NULL ,
`fJmYoZM3efDAxBYoHs17pw==`  BIGINT  NOT NULL ,
`NrpuCnNWET310Z9nPhbnOg==` DEC(65,1) NOT NULL ,
 PRIMARY KEY  (`GC/AFI9hZ7X4LIa8Txme6w==`,
`fJmYoZM3efDAxBYoHs17pw==`))ENGINE=InnoDB;

CREATE TABLE `fzXHLwb+i/thXZ73BaqFCQ==` (
`GC/AFI9hZ7X4LIa8Txme6w==` CHAR(128) NOT NULL ,
`4j4udebuLV7/SWNdW6aMDw==` CHAR(128) NOT NULL ,
`QFiUc+dUiTYetY6zgmJnCA==` CHAR(128) ,
`CJwFz4IgnoxFlniA7edCMg==` BIGINT ,
`P4PRqFnHct5kuMneZ8u93g==` CHAR(128) ,
 PRIMARY KEY  (`GC/AFI9hZ7X4LIa8Txme6w==`,
`4j4udebuLV7/SWNdW6aMDw==`))ENGINE=InnoDB;
```

Table 3.2. Translated expressions, as they are prepared to be sent to the encrypted mySQL database, for the creation of a new database instance.

Note that for every single expression that contains the mySQL *create table command*, the application builds and shows a brief user dialog, which displays a list of column names, each column name has a corresponding drop down menu where different encryption types are specified as selectable options. The options listed for encryption types are Numeric with Comparison (NC), Numeric with Comparison and Arithmetic (NCA), String Comparison only (SC), String Comparison and Substring matching (SCS) and DATE (one of this values is already selected automatically by the application, which is chosen based on the actual mySQL data type). The user then has the option to use the default provided or to specify the encryption type that, based on his/her best judgment, will provide the most reliable query performance. The next figure shows an example for one of these user dialogs.

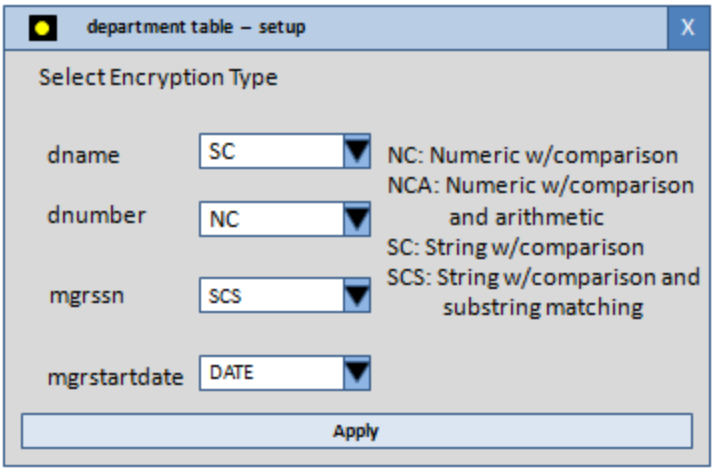

Figure 3.5 Typical user dialog for table creation, in this case, for the department table. At this point the user is given the opportunity to change the encryption types to be used during the creation of the table. However the program is already suggesting what seems to be the best options.

Once that the user is done with selecting the encryption types, and it clicks on the "Apply" button, the application then creates a .cfg file, which will be used for further encryption and decryption processes. This file will store the encrypted table names and its corresponding encryption types, this type of files are actively used by other mySQL transactions like query translation and result decryption. The following figure shows the actions behind the create table command from a programming perspective.

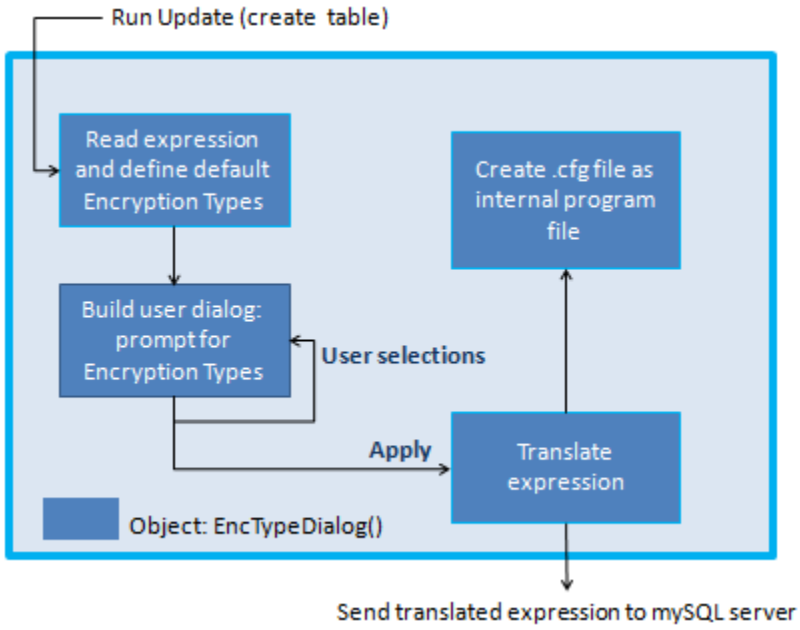

Figure 3.6 Internal actions for a create table mySQL expression

LOAD DATA. Load data is also categorized as another important update operation. The next table shows a typical example for the database company, with these commands or

expressions, the deployment of the new database recently created will be complete, since all the necessary data will be encrypted and then sent to the mySQL server.

```
use `company`;

LOAD DATA LOCAL INFILE 'employee.unl'
INTO TABLE employee
FIELDS TERMINATED BY '|' ;

LOAD DATA LOCAL INFILE 'department.unl'
INTO TABLE department
FIELDS TERMINATED BY ' ';

LOAD DATA LOCAL INFILE 'dept_locations.unl'
INTO TABLE dept_locations
FIELDS TERMINATED BY ' ';

LOAD DATA LOCAL INFILE 'project.unl'
INTO TABLE project
FIELDS TERMINATED BY ' ';


LOAD DATA LOCAL INFILE 'dependent.unl'
INTO TABLE dependent
FIELDS TERMINATED BY ' ';


LOAD DATA LOCAL INFILE 'works_on.unl'
INTO TABLE works_on
FIELDS TERMINATED BY ' ';
```

Table 3.3 Typical example for a LOAD DATA mySQL script.

Similarly to the script with *create schema* and *create table* commands, all the expressions are translated so that the table names are encrypted in the new expressions. Also, the .unl file names are used to create new internal .unl files. These new .unl files are created by reading the original .unl files provided by the user (or administrator) and then for every single data field, an encryption is applied, and this is based on the contents of the .cfg file previously created. In this way, by the time that the translated expression is sent to the

mySQL server, the data being transferred is readily available for its trip to the database. The next table shows the resultant *load data* mySQL script, after translation:

```
use `LUSInWQxFWbj79xu5VL2tg==`;

LOAD DATA LOCAL INFILE 'c:\\Program Files\\SHDB_GUI\\employee-sh.unl'
INTO TABLE `1m4t1hSG+qIBhoxiwpawJw==`
FIELDS TERMINATED BY  '|' ;

LOAD DATA LOCAL INFILE 'c:\\Program Files\\SHDB_GUI\\department-sh.unl'
INTO TABLE `latCaFEi7UA57y7KfeHc2Q==`
FIELDS TERMINATED BY  ' ';

LOAD DATA LOCAL INFILE 'c:\\Program Files\\SHDB_GUI\\dept_locations-sh.unl'
INTO TABLE `/6+zaceU2bmo1qU26UiJ9w==`
FIELDS TERMINATED BY  ' ';

LOAD DATA LOCAL INFILE 'c:\\Program Files\\SHDB_GUI\\project-sh.unl'
INTO TABLE `27f0ba7J9+Pt7+XslTGVzg==`
FIELDS TERMINATED BY  ' ';

LOAD DATA LOCAL INFILE 'c:\\Program Files\\SHDB_GUI\\dependent-sh.unl'
INTO TABLE `fzXHLwb+i/thXZ73BaqFCQ==`
FIELDS TERMINATED BY  ' ';

LOAD DATA LOCAL INFILE 'c:\\Program Files\\SHDB_GUI\\works_on-sh.unl'
INTO TABLE `W57f8/oxQe1HWv+9TQHSqg==`
FIELDS TERMINATED BY  ' ';
```

Table 3.4 Translated LOAD DATA mySQL script.

So, in summary, for a *load data local infile* expression, the file names are collected for future handling in a separated process, but first the update expression is translated

(basically the table names are encrypted), and the file names are substituted by a different internal file name (.unl file). Also the specified delimiters are reused for the new internal .unl file encryptions. Here is where the .cfg files are used to determine what encryption type to use for each column. Once that the translation is completed, the new expression is sent to the mySQL server, so the rest of the processing is completed by mySQL. The next figure shows the complete LOAD DATA process from a programming perspective.

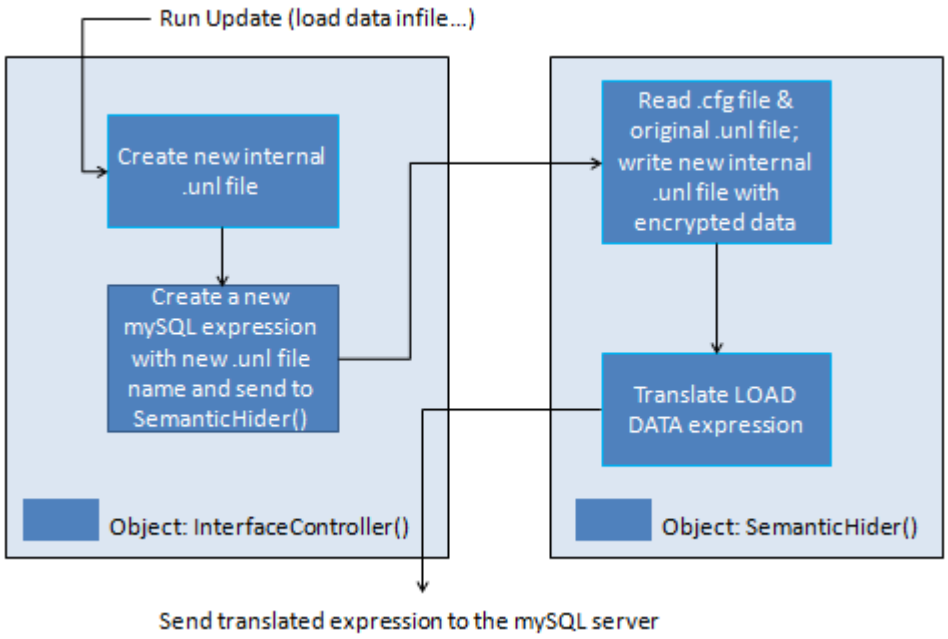

Figure 3.7 LOAD DATA internal encryption and translation process.

There are many other update operations but their handling can be much simpler. For those cases, generally the translation of the mySQL expression is good enough. See next example.

ALTER TABLE. This kind of mySQL update is very straight forward, so no internal processing is required in addition to the translation (encryption of the expression). The next table shows how this works.

```
Original:

ALTER TABLE dept_locations ADD FOREIGN KEY (mgrssn) REFERENCES
employee(ssn);

Translated expression:

ALTER TABLE `/6+zaceU2bmo1qU26UiJ9w==` ADD FOREIGN KEY
(`U81Z8kLh7DHXLej1+uHQlw==`) REFERENCES
`1m4t1hSG+qIBhoxiwpawJw==`(`ZOTDsz7gk1P5Yhz+ksVPcA==`);
```

Table 3.5. Translation of ALTER TABLE update operation. This is a very straight forward case.

The internal processing for ALTER TABLE is shown in the next figure.

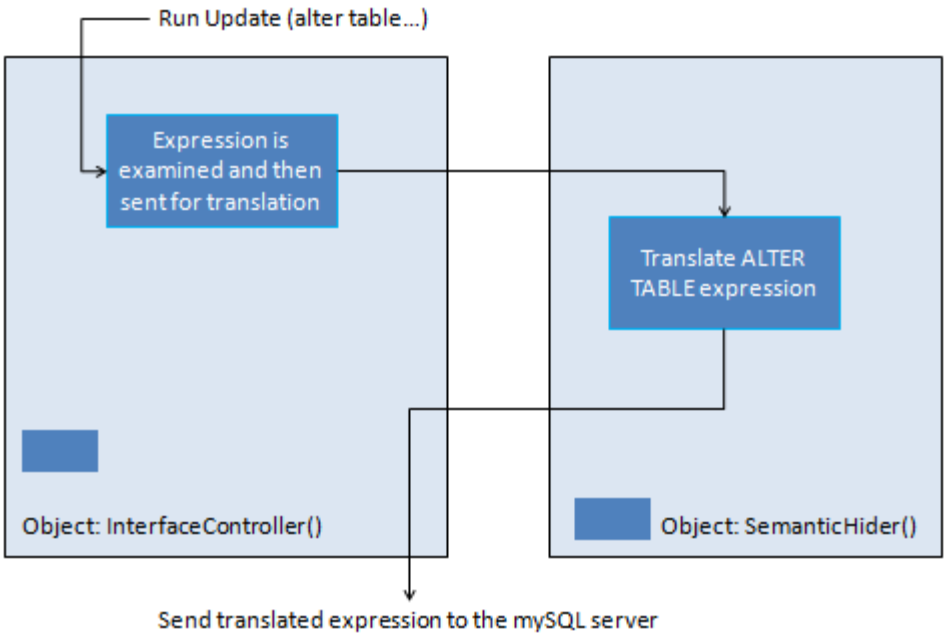

Figure 3.8 ALTER TABLE internal translation process.

3.2.2 Query operations

As mentioned before, the JDBC handles mySQL query operations in a different way. But it is not only JDBC which requires the distinction. A query operation is also very different in the sense that we are not adding or modifying any data to the database, in this case, we are retrieving information from the database in a very selective way. For a typical query operation, we needed to consider that the original query typed by the user also needs translation, because the database names, table names, values and other parameters cannot be compared with the encrypted information in the database, so we needed to perform a very careful translation that considers the initial configuration of the database (that is, the .cfg files). Also, the translation cannot be made directly, first, it is necessary to have our SHDB application to perform a preliminary interpretation of the query, so that the application can learn which .cfg files will need to be accessed and what decryption types are necessary for the equalities in the expression. Yet another consideration is that the equalities can be typed by the user altogether or also can be typed to the left or to the right of a determined value, so we need to provide the flexibility of not restricting to the user how the syntax of equalities is specified and same thing for the use of parenthesis in aggregate functions. Last but not least, the SHDB program needs to be capable of handling nested mySQL query expressions.

But so far we have covered only an overview of query translation. That is not all the work that we needed. When a query translation is complete, and then is sent to the mySQL server, we need to wait for the response from the server, then we need to collect the returning data into an internal, temporal file. Then the next step is accessing the internal .cfg files of our SHDB application to learn what decryption methods we need to

use for every corresponding column in the returning data. Finally we decrypt the data stored in the SHDB internal results file (which contains encrypted data only) and present the decrypted information to the results window of our Main GUI. That completes a typical mySQL query transaction.

The next table shows a simple example for a query translation. As mentioned above, before the translated query is sent to the mySQL server, we need to perform a syntax correction process, a parsing process, an encryption process based in the internal .cfg files and finally we send the encrypted or translated expression to the server.

Original query expression typed by the user:

```
select distinct fname, lname
from employee, works_on, project
where dno=dnum and pnumber=pno and essn=ssn;
```

Translated query expression, after syntax correction, parsing and encryption internal processing:

```
select  distinct `e5Va3+bSOoxaWvrSNet3Dw==`,`qobZJ/A0EeEHwlxbld+kaQ==` from
`1m4t1hSG+qIBhoxiwpawJw==`,  `W57f8/oxQe1HWv+9TQHSqg==`,
`27f0ba7J9+Pt7+XslTGVzg==` where `j/80kIzc3J5QWjv35ISKbA==` =
`ei6KZldLywvE6ycmSk2BfQ==`  and `V8GeHzVDL2B47CtJo0SxdQ==` =
`fJmYoZM3efDAxBYoHs17pw==`   and `GC/AFI9hZ7X4LIa8Txme6w==` =
`ZOTDsz7gk1P5Yhz+ksVPcA==` ;
```

Table 3.6 Example of Typical query translation.

The case for translations when nested query expressions are found is a bit more complex. For nested queries, the SHDB application identifies where a nested query starts and ends, then it is saved into an array of nests for later use. The nested expression is substituted with a flag that contains the index representing the position in the nest array. Also, a second array of nests is created with the purpose of saving translated nested expressions;

of course, the same index is used to make things consistent. In that way, when the main expression is completely examined, a reassembly is made by substituting the nest flags by the translated nests, forming in this way the final expression that we can send to the mySQL cloud or local server in use. The next table shows an example of nested query translation.

Nested expression provided by the user:

```
select dname from department
where (select sum(salary)
        from employee
        where dnumber=dno) >= 10000;
```

Nested expression after syntax correction, parsing, nest handling and translation.

```
select `CvblhUbRIbMn4yJhMinGjA==`
from `latCaFEi7UA57y7KfeHc2Q==`
where  ( select sum(`TO/L7fedob0d+gkoE1/19g==`)
          from `1m4t1hSG+qIBhoxiwpawJw==`
         where `hy0yWjVQ2SaGswqsnWy7Lw==`
        = `j/80kIzc3J5QWjv35ISKbA==` )  >=  430000.0;
```

Table 3.7 Example of typical nested query translation.

The use of nest arrays allows for the proper handling of multiple nests in a query expression. See next example as well:

Expression provided by the user, with two nests:

```
select fname, lname, dname
from employee, department
where ssn=mgrssn and dnumber in ( select dnum from project )
and mgrssn in ( select mgrssn from department );
```

Nested expression after syntax correction, parsing, nest handling and translation.

```
select `e5Va3+bSOoxaWvrSNet3Dw==`, `qobZJ/A0EeEHwlxbld+kaQ==`,
`CvblhUbRIbMn4yJhMinGjA==` from `1m4t1hSG+qIBhoxiwpawJw==`,
`latCaFEi7UA57y7KfeHc2Q==` where `ZOTDsz7gk1P5Yhz+ksVPcA==` =
`U81Z8kLh7DHXLej1+uHQlw==` and `hy0yWjVQ2SaGswqsnWy7Lw==` in  ( select
`ei6KZldLywvE6ycmSk2BfQ==` from `27f0ba7J9+Pt7+X
slTGVzg==` ) and `U81Z8kLh7DHXLej1+uHQlw==` in  ( select
`U81Z8kLh7DHXLej1+uHQlw==` from `latCaFEi7UA57y7KfeHc2Q==` )
```

Table 3.8 Example of query translation with multiple nests.

In summary, once that a translated query is sent to the mySQL cloud or local server, we wait for the process to be completed, that is, for the returning data to arrive to our SHDB application. Then we direct this information (which is completely encrypted) to an internal results file. From there, we use the encrypted column headers (also retrieved along with the results) as a reference to perform a look up into our .cfg files, which provide us with the decryption types to be used for our results. After performing the corresponding decryption we now present the results to the results window of the Main GUI.

The following figure show the complete processing for queries run in our SHDB application:

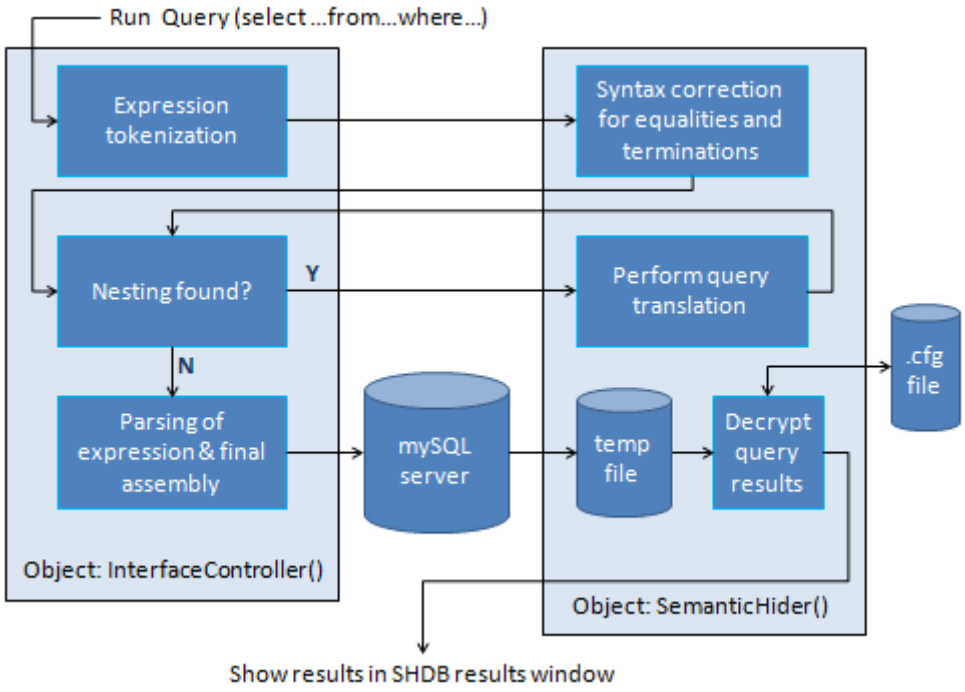

Figure 3.9 Internal SHDB query processing.

The same process shown in Figure 3.9 is applied for other type of queries like show tables, show databases, etc., except that when sent for translation, it is determined that there is no information to translate.

The extra processing required to handle encrypted data can be skipped by selecting a non secure mode, this was made to perform comparison of results as well as for demonstration purposes. Chapter 4 includes actual images of the implemented application including the option to communicate with the mySQL server in either secure or non secure mode.

3.3 Summary

An input from the user in the form of a mySQL script can consist of one or more mySQL expressions. The application is made so that every individual expression is handled individually and sequentially.

There are two major kinds of mySQL expressions: Updates and Queries. Updates and Queries are handled in a very different manner, not only because the JDBC library requires it, but also because for an update we send information to our mySQL server while for a query, we retrieve information so, in a way, they are inverse processes.

By storing encrypted data in the mySQL query, we are required to send both update and query expressions translated accordingly, that is, our expressions have to contain encrypted database names, table names, parameters and values. Of course, we do not encrypt commands, function names, special characters or equality symbols.

Some of the updates and queries require significant internal processing, but those operations are hidden from the user, providing transparency and ease of use.

For convenience, our SHDB application provides a two operation modes, secure and non secure. This is covered in more detail in the following chapter.

CHAPTER 4. IMPLEMENTATION

4.1 Implementation stages

In general, the implementation process can be summarized in the following major stages:

- Setting up a mySQL local server
- Start running queries and updates (in normal, non-secure mode)
- Implementation of a Login GUI with prompts for URL, login name and password
- Integration of Login GUI with code in charge of connection and authentication to the server
- Creating of a Main GUI with handles for major operations like queries, updates, text edition, data import and export and other processing.
- Setting up a mySQL cloud server in AWS and ClearDB service providers
- Implementation of Secure Mode to support encryption and decryption of data for my SQL update operations
- Implementation of Secure Mode to support encryption and decryption of data for my SQL query operations
- Addition of auxiliary functions to support miscellaneous features
- Testing and optimization of code
- Deployment

4.2 A glance at the source code

This section lists the java source files and its corresponding description, giving a general understanding of its role in the SHDB application:

AES_Encrypter.java : In charge of encryption of plaintext and decryption of cyphertext using the AES cryptographic system. The methods provided are encrypt(), encryptWordbyWord(), decrypt(), decryptWordbyWord() and generateKey().

DeleteDialog.java : User dialog that requests confirmation from a user before deleting a file selected for removal.

EncTypeDialog.java : This class has a lot of functionality. It is triggered by the mySQL *create table* expressions, and it uses an instance of the InterfaceController, HomomorphicEncrypter and the AES_Encrypter classes. It builds and shows a user dialog for each of the new tables during its creation process, then prompts the user for confirmation or modification of the suggested encryption types. After the user confirmation of the selected options, it runs the JDBC update process, sending the translated expressions to the mySQL server. Its more relevant methods are defaultEncr(), conVarType(), actionPerformed() and main().

ExtDialog.java : User Dialog that prompts the user for the type of file extensions to be shown on the left text are, which is also called the “File List Window”.

HomomorphicEncrypter.java : In charge of encryption of plaintext and decryption of cyphertext using the HFE cryptographic system. The methods provided are GenerateRandomInteger(), nextRandomBigInteger(), GenerateBigRandPrime(), fillbytes(), generateKeys(), Dec2Hex(), Hex2Dec(), loadKeysFromFile(), encryptString(),

encryptStringFixedEQ(), encryptNumber(), encryptNumberFixedEQ(), decryptString() and decryptNumber().

InterfaceController.java : This is one of the most important classes in the SHDB application. It is the first object called when the SHDB icon is started. There are several methods and they are: setURL(), setUSER(), setPSWD(), stopConnection(), runUpdate(), runQuery() handleUSE(), parseQuery(), readPath(), getBinPath(), getFileSize(), convPath(), readSelectedFile(), spaceHandler(), runJLT1(), runJLT2(), runJLT3(), runJLT4(),runJLT5(), filterSelectedFile(), hideFromSelectedFile(), getConfigFile(), getFileSizeInt(), savePath(), createTXTFile(), getExecPath(), readFavorites(), writeFileList(), saveFav(), saveConfig(), deleteSelectedFile(), setPath() and main().

Login_GUI.java : A user dialog in charge of collecting URL, login name and password from the user. It uses an instance of the InterfaceController class. Its most important methods are windowClosing(), componentShown(), actionPerformed(), propertyChange(), clearAndHide(), main() and run(). It uses an instance of the InterfaceController().

Main_GUI.java : The most important Graphical User Interface in this application. It is the central interface for the user and it makes all the functionality available by showing a big variety of buttons, menus, text areas, text fields and interchangeable panes. Its methods are createFDImage(), setStatus(), setResults1(), setResults2(), setResults3(), caretPosHighLight(), findPanelSetup(), resultsPanelSetup(), displayPanelSetup(), queryDisplayPanelSetup(), analysisPanelSetup(), editPanelSetup(), queryPanelSetup(), updatePanelSetup(), filterPanelSetup(), cardContainerSetup(), cardContainer2Setup(), createFileMenu(), createEditMenu(), createFindMenu(), createFilterMenu(),

createAnalyzeMenu(), createModeMenu(), createFavoritesMenu(), updateFavorites(), createHelpMenu(), checkFile(String bookfile), updateWindows(), updateWindow1(), updateWindow2(), updateWindow3(), add2LinkedList(), status_Processing(), status_Ready(), please_select_a_file(), actionPerformed(), getAction(), mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased(), lostOwnership() and main ().

Pader.java : In charge of ASCII Pading plaintexts that need to be treated by numbers for Homomorphic Encryption. Its methods are ASCIIDepad(), ASCIIDepad_string(), ASCII_Depad_array(), ASCIIPad(), CheckPadding(), convert_str2ascii() and main().

ResNDialog.java : A User Dialog used for several scenarios, like prompting for a file name when exporting results, saving a filter configuration file name, for prompting for a string that needs to be found in the editor mode.

SemanticHider.java : This class has the most difficult tasks during the execution of the SHDB application. It is in charge of performing translation of query or update mySQL expressions, it has an interpreter (parser) for queries and determines the types of encryptions based on the internal .cfg files, it is also in charge of encrypting .unl files, and it handles several calls to the encryption and decryption classes. Its methods are:
decryptQueryResults(), isQueryCommand(), isAggregate(), removeLastChar(), isAnIntegerNumber(), getEncType(), accept(), parseEqualities(), semHideQuery(), semHideUpdate(input), accept(), isColumn_VarType(), isKeyDef(), isValidExpr(), isCondition(), encryptValue(), encryptUNL(), convPath(), convVarType(), conv2DateSH(), convert2DateSH(), conv2Date(), convert2Date(), getNests(), openNests(), isNested(), tokenizator() and main().

SplitDialog.java : User Dialog to prompt users for the desired number of splits in the selected file.

4.3 Challenges and important considerations

The implementation of the SHDB application has been a lengthy learning process with many interesting challenges during the different stages of implementation. This section explains the most relevant challenges faced as well as important considerations that could not be ignored during the design and the writing of code.

Table names and database names are not case sensitive. However encrypting a plaintext in mixed case returns a different cipher that encrypting a plaintext in uppercase and also returns different values for lowercase. This is an important consideration to take into account, since the encryption process needs to be consistent for all database names and table names provided by the user. This applies only for windows mySQL. The SHDB application converts table names and database names to lowercase when necessary.

Tokenization of mySQL expressions can be difficult when the user does not use spaces, equalities and terminators in a consistent way. Therefore it was required to perform extra processing to fix the syntax of every mySQL expression entered by the user. Basically equalities are separated from variable names and values with spaces, parenthesis need to be recognized for aggregate functions and differentiated from nested expressions. Table names can have or not have quotation marks. Line feeds and carriage return ASCII symbols can be in the middle of the expression making translation more difficult. This is why the SHDB application always performs a syntax correction before completing any further processing.

Encrypting text requires padding for the Homomorphic Encryption (HFE) technique. The HFE technique uses a quadratic equation during its encryption process, that means that a plaintext needs to consist of a number; otherwise, we can get a NumberFormatException error from the Java Virtual Machine (JVM). In order to address this situation, a complex padding function was implemented and it is used whenever a text needs to be encrypted using HFE.

JDBC offers limited error handling. The Java Database Connector (JDBC) library returns very limited information around errors found by mySQL. This reduces the amount of resolution that can be used when reporting errors in the Main GUI. For this reason, most of the messages related to queries and updates errors refer to syntax errors even though the cause of the problem can be an invalid database name, table name, etc.

Not all mySQL cloud providers support the open source mySQL JDBC driver. This means that the SHDB application might not work for some cloud providers, that is the case of Google mySQL Cloud services, which does not support the open source JDBC driver, they only support their own proprietary driver which means, having to change some lines of code in the SHDB application. However AWS, ClearDB and any other cloud provider using the open source mySQL JDBC driver are compatible with SHDB.

Most of the mySQL cloud providers require a credit card number when creating a user account. This makes it inconvenient when the purpose is to try many cloud providers. Care must be taken of not falling into the end of the trial period and also when the trial period ends, charges to the credit card are automatic.

A great variety of mySQL commands exist, giving way to too many possibilities in terms of mySQL expressions and its combinations, which eventually need to be

supported. The SHDB application supports the most common types of commands and its combinations with aggregate functions, nested queries, equality types, etc. However some cases might be still unsupported. Further work is necessary and it is important to mention that upon any code changes, the best practice will be, to test as thoroughly as possible to ensure that the robustness and reliability of the application is preserved.

Tools used for development:

Version control: Tortoise SVN.

For C++ code development: Bloodshed.

For Java code development: Eclipse.

mySQL cloud providers: Amazon Web Services and ClearDB.

4.4 Results

This section presents some actual results from the implemented application. We run a few examples in the actual application showing the final look and feel of the Graphical User Interfaces and User Dialogs including real results from transactions made to the databases located with the mySQL server (from cloud provider).

Login GUI. The first interaction with the user after starting the program is the Login GUI. This one collects the URL of the mySQL service provider, the user name or login name and the password, which is hidden from view by using special characters as the password value is typed.

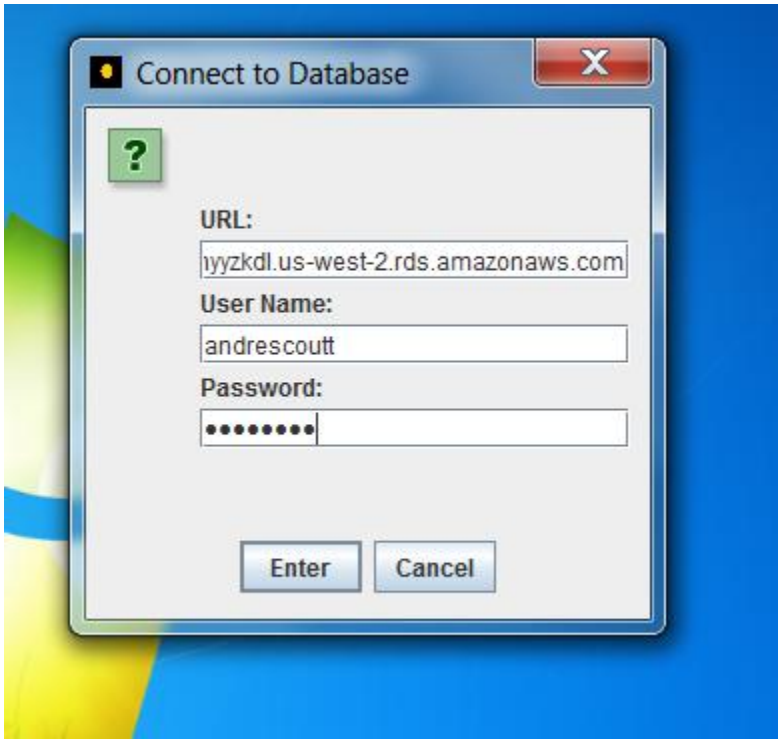

Figure 4.1 Login GUI

In the case that the user types invalid information, which cannot be validated by the mySQL server, the following error message is posted:

Figure 4.1.1 Error message for a failed Login GUI session.

The user can have up to three opportunities to provide the correct information, otherwise the application is closed.

Welcome message and Main GUI. If after entering the URL, login name and password, the user is properly authenticated and connected to the mySQL server, the SHDB application will post a welcome message as well as the Main GUI.

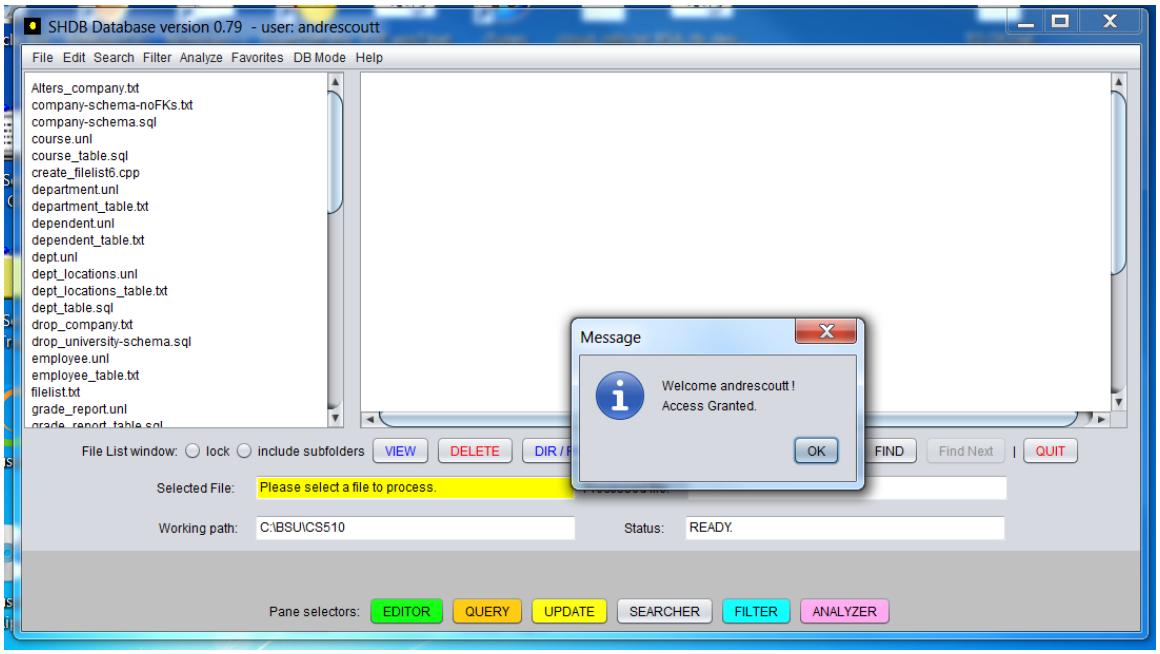

Figure 4.2 Welcome and Main GUI

After closing the Message window, the user can now start working with the SHDB application. The accessibility to different databases as well as permissions to update or delete information must be determined by the administrator who setups the access rights for the different users.

The next figure shows one of the most typical examples when setting up a new database instance. In this case, the user clicks on the second file from the list which makes the SHDB application to show the contents of the file, in this case, a mySQL script to create a database named ‘company’ and create the corresponding table names.

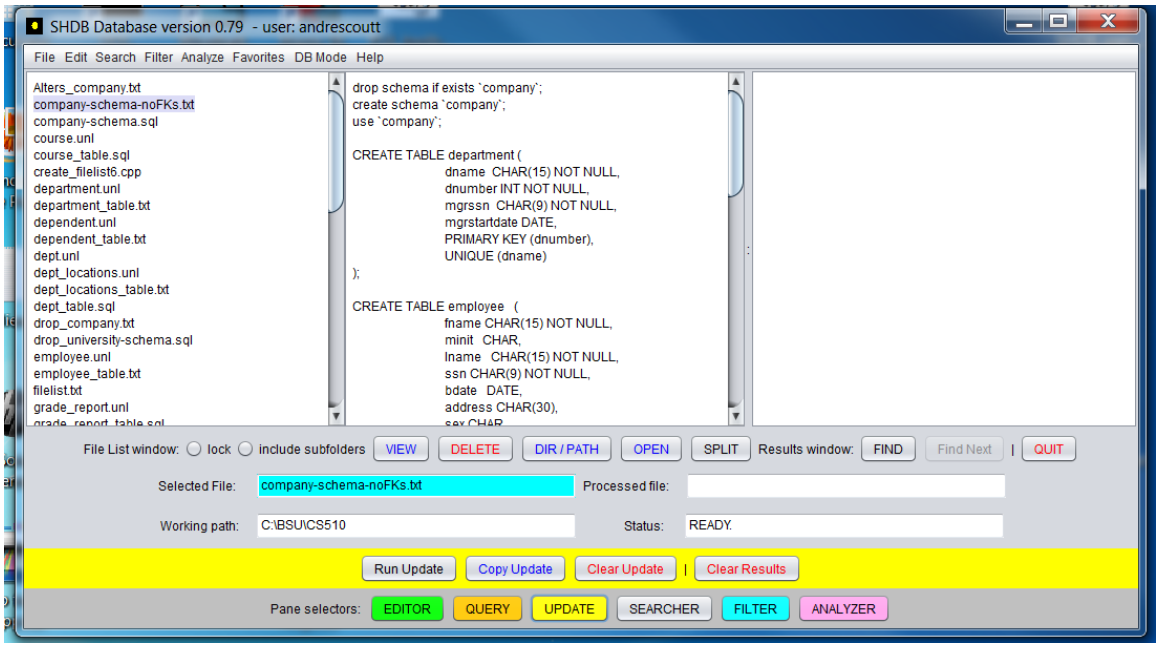

Figure 4.3 Run Update, create schema and tables.

Before the transaction can be completed, the user will get a user dialog for every one of the tables referred in the script. The purpose of these user dialogs are to suggest to the user the encryption types that can be utilized during the Semantic Hiding of the information and also, the user is given the opportunity to change the corresponding selections. Please refer to the next figure.

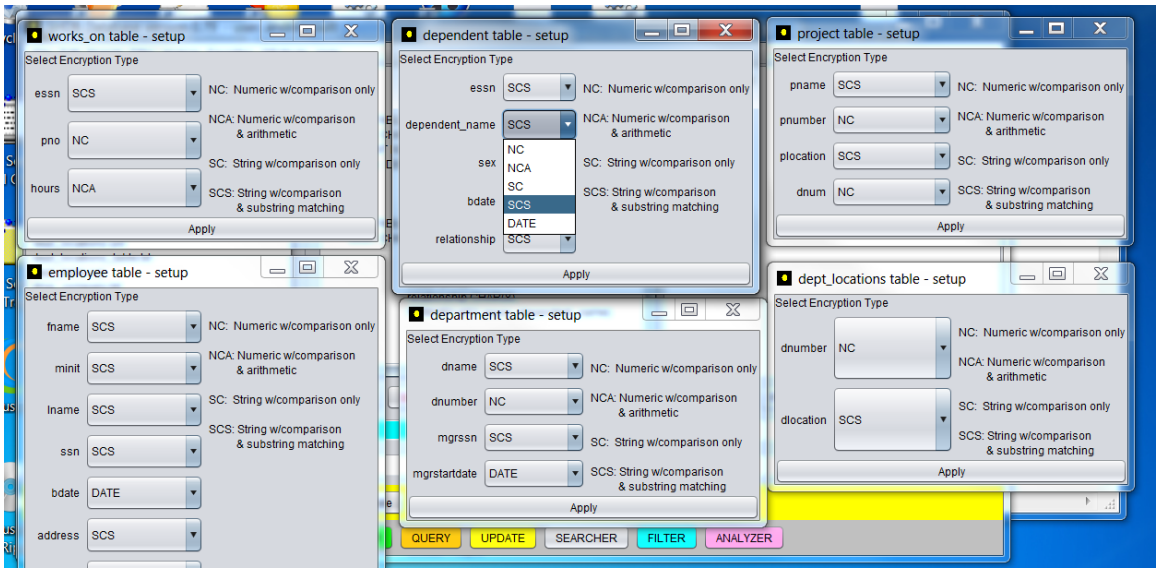

Figure 4.4 Run Update, Encryption Type definitions for tables.

When the user clicks on the 'Apply' button, the corresponding *create table* mySQL expression is translated and sent to the mySQL server for its creation.

The next step for this demonstration is to load the database with the information from the tables. See next figure:

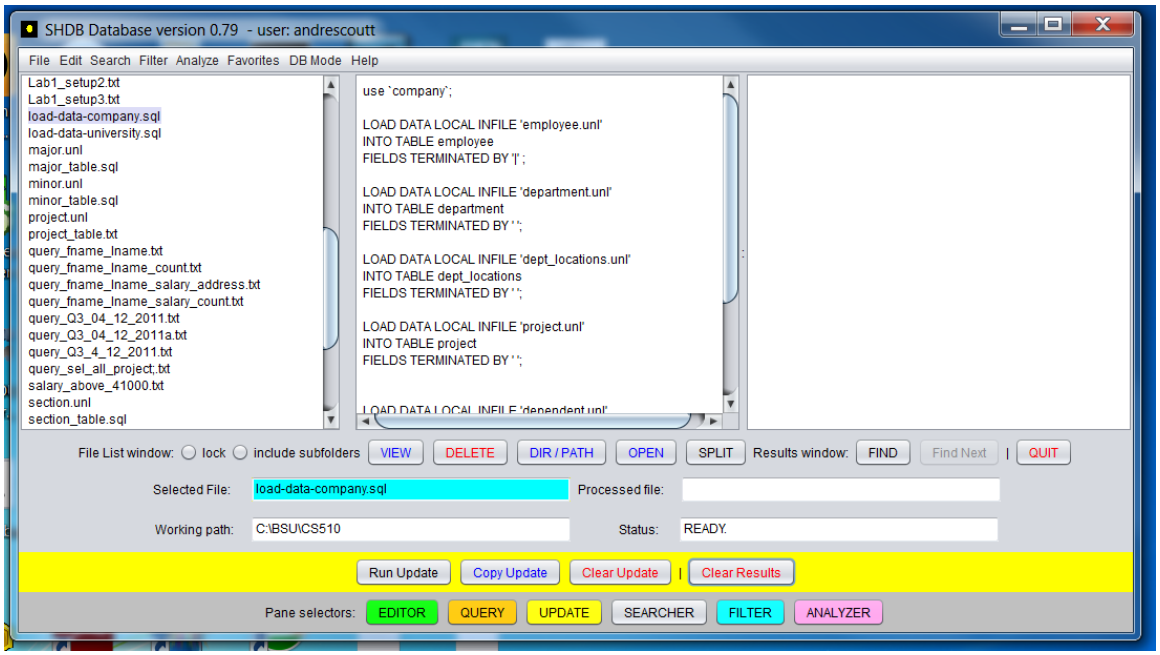

Figure 4.5 Run Update, load database.

When the user clicks on the 'Run Update' button, a lot of things happen. Basically the *load data* expressions are translated, internal and temporal .unl files are created with the encrypted information based on the original .unl files and the information is then sent to the mySQL server.

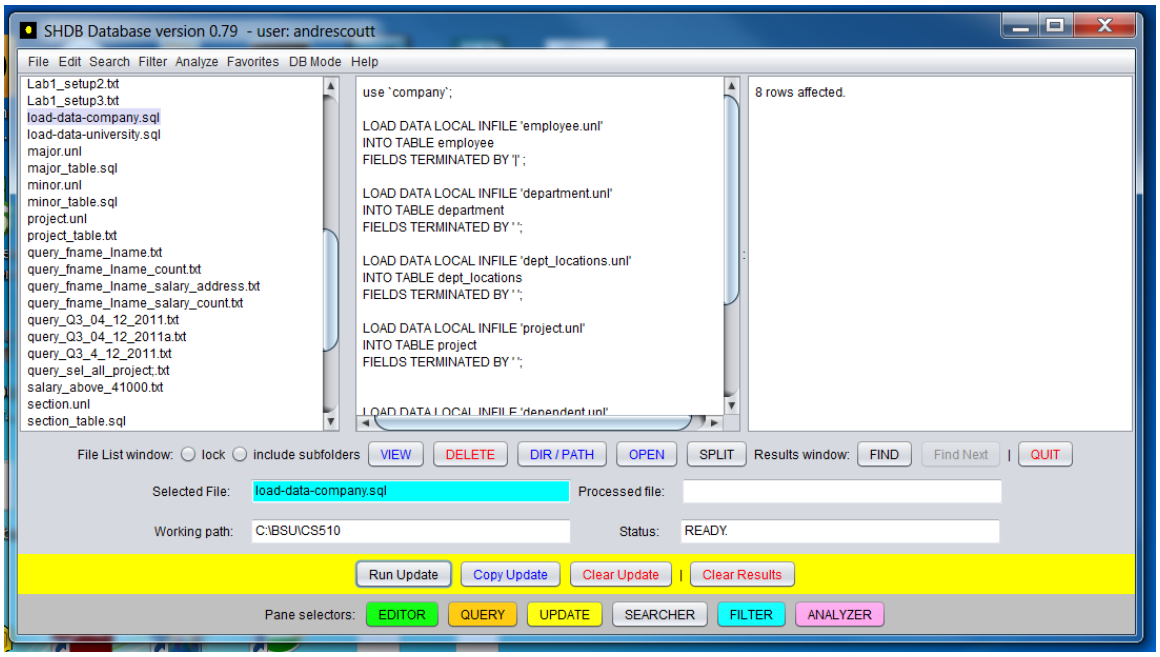

Figure 4.6 Load data completion.

The feedback for the user after clicking on the Run Update button is minimal, in this example, it is "8 rows affected", which is exactly the same message that the user would get by using a mySQL interface that does not handle encryption. This is a good example of the transparency provided to the user by using the SHDB application.

Now let's see some examples of query operations. For the next cases, let's consider that the user has already pressed the 'QUERY' orange button to bring the query pane with the corresponding buttons in the orange bar.

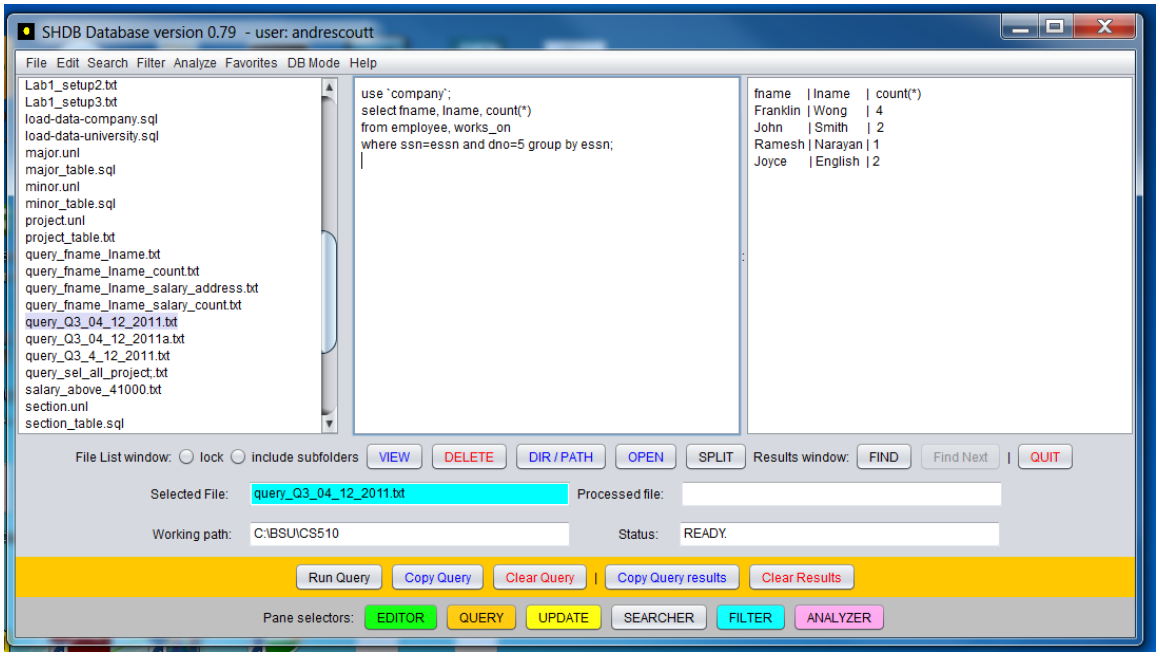

Figure 4.7 Run Query example 1.

When the user clicks on the ‘Run Query’ button, the results are instantly shown on the third text area on the right, that is called the ‘Results window’.

Similarly to the previous example, a lot goes on for an operation like this. As soon as the ‘Run Query’ button is pressed, the syntax of the expression is reviewed and corrected (whenever that it is possible), the query expression is parsed, the internal .cfg files are accessed to find the necessary encryption types to be used and then the translated mySQL expression is sent to the mySQL server that is currently connected to the application. When the mySQL server responds with data results, they are written to an internal temporal file, along with the corresponding headers for each column. These information is then decrypted, again by accessing the .cfg files which are chosen at the time that the query expression was parsed. Once that the results are decrypted, they are sent to the “Results window” so the user can look at them (and/or copy to the clipboard).

Let’s see a couple more examples:

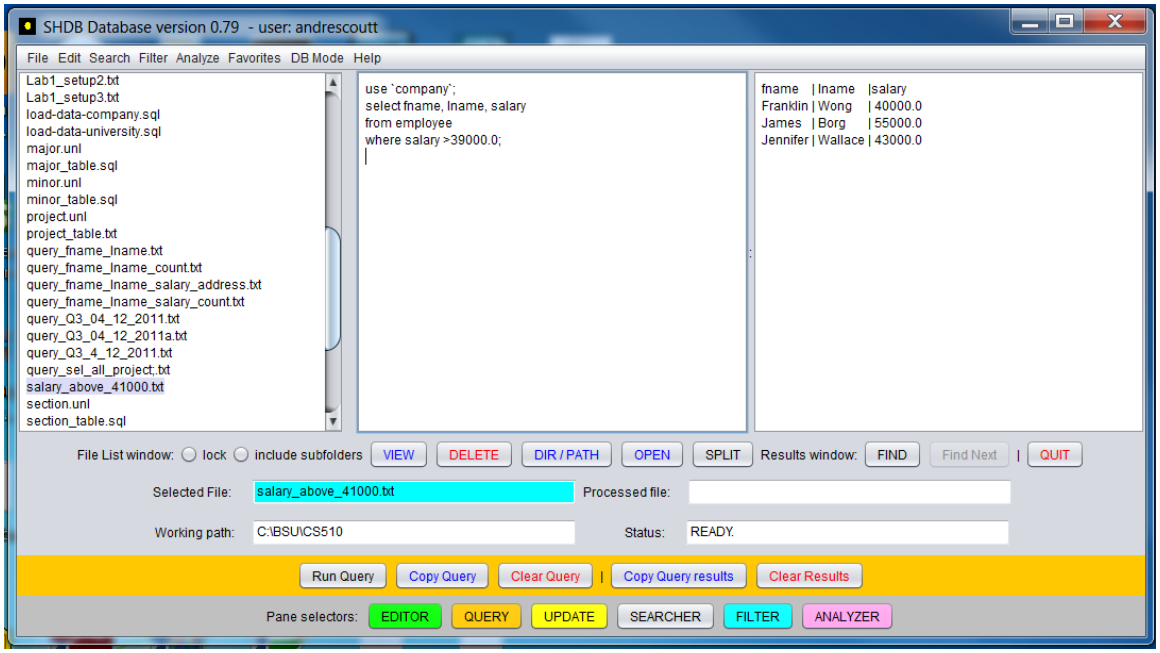

Figure 4.8 Run Query example 2.

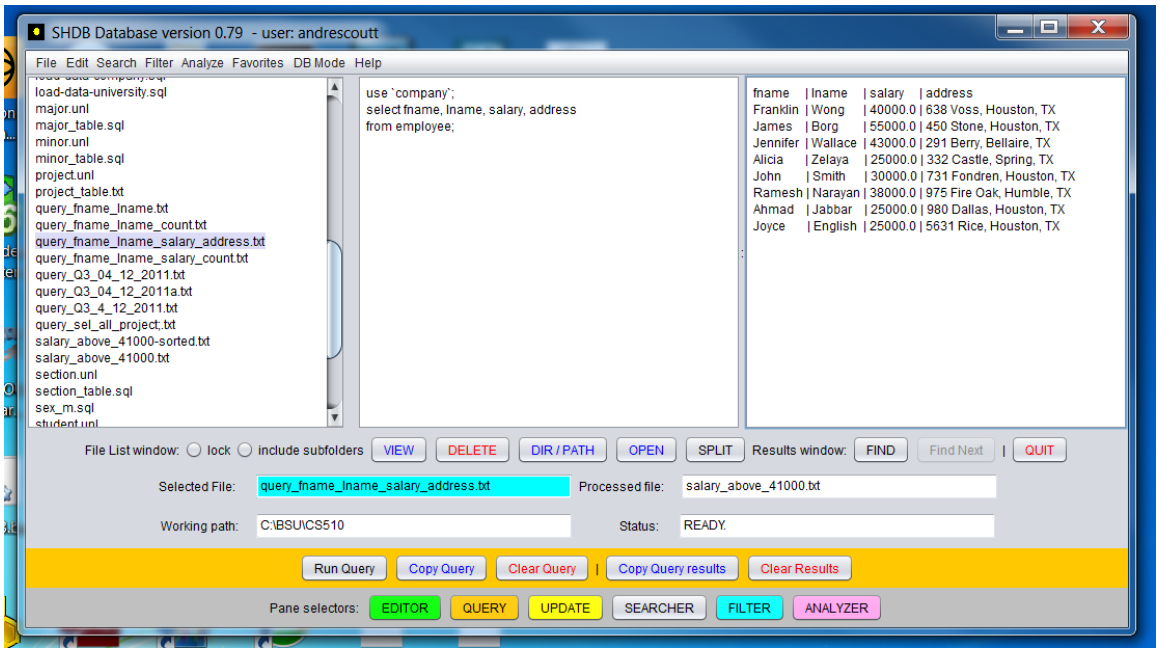

Figure 4.9 Run Query example 3.

As mentioned before, the user will get the same look and feel as if the application was not using encryption at all.

As a bonus feature, for demonstration purposes and also for development, the SHDB application includes a ‘DB Mode’, which can be secure (default) or non secure (selected by removing the check mark from the ‘DB Mode’ menu item, ‘Secure’.

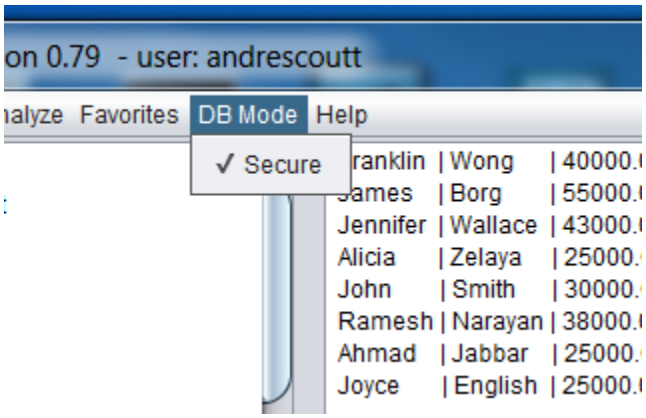

Figure 4.10 Database mode.

The next Figure shows how the non secure mode can be used to have direct access to the encrypted information, this feature is useful for demonstration and development purposes and can be easily removed by commenting a single line of code in the SHDB application development tool.

- In normal mode, we can see the effects of the encryption

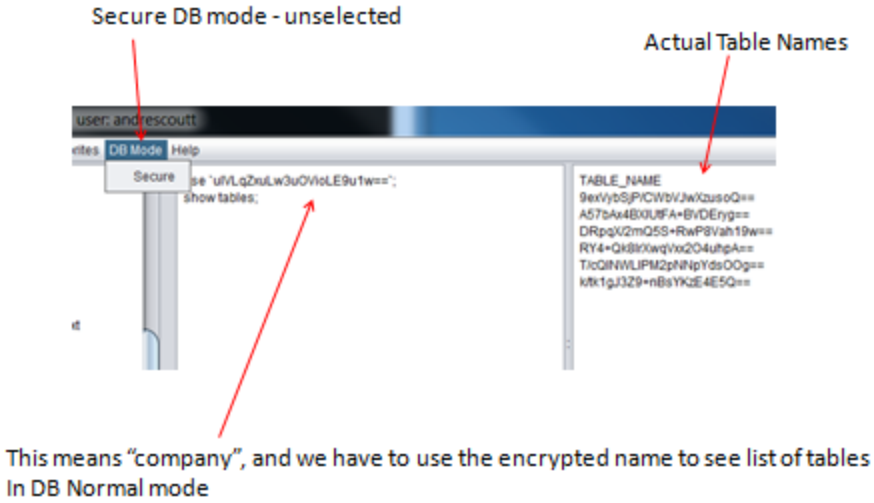

Figure 4.11 Effects of the encryption revealed by using normal mode.

4.5 Future work

As in any other implementation, there is always room for improvement. The following features could be added to enhance the user experience of this application:

- Addition of administration capabilities to the Main GUI (like user creation, deletion and permission control).
- More work is required to provide support to those mySQL commands not included yet in the SHDB application.
- Capability to export data from "Results Window" to excel tables or command separated value (.csv) files.
- Creation of log files with usage of the SHDB application.
- Capability to create automatic version updates of the SHDB application.
- Options to change the look and feel of the GUI windows as well as options to change the layout of the text areas.
- Resizable text areas.

CHAPTER 5. CONCLUSION

The goal of implementing a comprehensive GUI application that could serve as a demonstration tool for the cloud database data privacy concept has been accomplished in this implementation. We did not want to come up with a solution that required a user to run a low level application in a command console and instead we decided to go ahead and implement the whole concept with a convenient Graphical User Interface. This resulted in an application that is easy to use for those users familiar with mySQL.

The implementation was carefully planned and divided in three major areas: authentication and connectivity, Graphical User Interface and Encryption and Decryption (for use in the data privacy feature). The data privacy feature was not easy to implement, and it was perhaps the most challenging part of this project. However, all of the complexity has been conveniently hidden from the user.

After months of hard work, a few thousand lines of C++ and Java code and very intensive testing, we think that what was born as an idea is now an actual implementation with a friendly look and feel but most importantly, robustness and ease of use.

In the future, we would like to add more convenient features to the application, always performing a very thorough testing in order to ensure that this continues to be a robust and flexible solution, applicable to the needs of the cloud database users of today.

REFERENCES

[1] http://www.winmagic.com/solutions

[2] http://aws.amazon.com/console/

[3] https://www.cleardb.com/developers

Author, Imaginary. 2011. *This Is an Example Source.* Boise: Boise State University Publishing.

Doe, John. 1903. *Imaginary Text.* London: Red Herring Press.

<Add book names here>