Capítulo 3: Migración de la base de datos del sistema del CSLR

En este capítulo se explica el proceso que se siguió de la base de datos del sistema de tutores y libros interactivos del CSLR.

Como ya se mencionó, la base de datos es manejada por Postgres y accedemos a éste mediante una terminal de Cygwin. Postgres es un manejador de base de datos muy poderoso que es capaz de manejar una gran variedad de tipos de datos.

En este capítulo se explicará paso a paso como se llevó a cabo la migración de la base de datos. Dicha migración realizó en varias partes, en primer lugar se migró la base de datos de Postgres a MySQL. Para lo cual fue necesario hacer una modificación a los archivos que se encargaban de realizar la autentificación del usuario; para que los datos obtenidos por parte de éste fueran comparados con aquellos guardados en la base de datos de MySQL y no la de Postgres. Finalmente se realizó la modificación de los códigos de los tutores para que trabajara ahora con la base de datos ya migrada y dentro de éstos se utilizara la sintaxis de MySQL y no la de Postgres.

Uno de los principales problemas que se tuvo es que la base de datos del sistema del CSLR además de trabajar con índices y llaves, también hace uso de triggers, funciones y vistas. Pero MySQL sólo maneja triggers, funciones y vistas de una manera rudimentaria; asi que fue necesario adaptar dichos triggers y funciones para que se lograran migrar a MySQL.

3.1 Migración de la base de datos de Postgres a MySQL

Ya que se vio el funcionamiento del sistema en general así como de los archivos involucrados en el mismo, se procederá a explicar el primer paso que se llevó a cabo para realizar la migración. El primer paso fue migrar la estructura de la base de datos de Postgres a MySQL.

Para poder obtener la estructura de la base de datos, se utilizó el comando pg_dump, para generar un archivo de texto con comandos SQL que permiten recrear la base de datos tal y como se encuentra en ese momento, es decir con las instrucciones que nos permiten recrear sus tablas,llaves,indices, triggers, funciones y vistas.

Para poder hacer uso de este comando, se debe de abrir una terminal de Cygwin, y teclear el comando siguiente:

```
pg_dump -s -U test isystemDB > dumpnodata.txt
```

–s: Esta opción nos permite obtener solo el esquema de la base de datos.

-U: Nos permite especificar el usuario y password de la base de datos.

-dumpnodata.txt: Es el archivo en donde se vaciará el resultado de la instrucción pg_dump.

Se pedirá un password para el usuario test, el cual es “test”. Después Postgres generará el archivo dumpnodata.txt con el esquema de la base de datos isystemdb.

El archivo generado contiene comandos SQL capaces de recrear la base de datos sin embargo dichos comandos tienen la sintaxis de Postgres, así que una vez generado el archivo, fue necesario modificar la sintaxis de cada uno de los comandos y cambiarla a la equivalente en MySQL; también fue necesario revisar los tipos de datos manejados en postgres y encontrar su equivalente en MySQL.

A continuación se muestran ejemplos de los cambios hechos a la sintaxis de dichas instrucciones:

Postgres	MySQL
`CREATE UNIQUE INDEX assess_sound_default_pk ON assess_sound_default USING btree (s_text);`	`CREATE UNIQUE INDEX assess_sound_default_pk ON assess_sound_default (s_text(767));`

Tabla 3.1.1 Instrucciones para la generación de índices.

En la columna izquierda se muestra la instrucción para Postgres y en la derecha la misma para MySQL.

La tabla 3.1.1 muestra las instrucciones para la creación de índices. En Postgres es necesario indicar el método de indexado, con la palabra “Btree”.

“Btree” es el método de acceso que por defecto utiliza Postgres, y es utilizado en todos los índices de la base de datos. Postgres cuenta con tres métodos de acceso para el índice, a continuación se explican muy brevemente:

BTREE: Una implementación de los árboles B de alta concurrencia de Lehman-Yao.[PostgreSQLDoc, 05]

RTREE:Implementa rtrees estándar utilizando el algoritmo de partición cuadrática de Guttman. [PostgreSQLDoc, 05]

HASH:Una implementación de las dispersiones lineales de Litwin[PostgreSQLDoc, 05]

En la creación de tablas de MySQL se usará el tipo de tabla INNODB, el cual utiliza como método de indexamiento el Btree, por lo que, la parte de la instrucción "USING btree" es omitida al hacer la transformación a la sintaxis de MySQL.

Se realizaron los cambios a 200 índices.

Para mostrar la modificación de la sintaxis de las llaves se utiliza el siguiente ejemplo:

Postgres	MySQL
`ALTER TABLE ONLY school_info ADD CONSTRAINT pk_school_info PRIMARY KEY (sch_abbr);`	`ALTER TABLE school_info ADD CONSTRAINT pk_school_info PRIMARY KEY (sch_abbr);`

Tabla 3.1.2 Instrucciones para la generación de llaves.

La instrucción "ALTER TABLE ONLY" nos permite crear un índice en una tabla ya existente. Para MySQL utilizamos "ALTER TABLE" Se realizaron los cambios a 104 llaves.

Como ejemplo para mostrar los cambios hechos a la sintaxis de las funciones y triggers se usó el siguiente ejemplo:

```
CREATE FUNCTION
delete_obj_compre_stream()
RETURNS "trigger"
    AS '
DECLARE
      a_out  VARCHAR;
      flag BOOLEAN;
BEGIN
      flag := TRUE;
      IF TG_OP = ''UPDATE''
THEN
            IF
NEW.OBJ_COMPRE_STREAM =
OLD.OBJ_COMPRE_STREAM THEN
                  flag :=
FALSE;
            END IF;
      END IF;

      IF OLD.OBJ_COMPRE_STREAM
IS NOT NULL AND
OLD.OBJ_COMPRE_STREAM > 0 AND
flag THEN
            SELECT
lo_unlink(OLD.OBJ_COMPRE_STREA
M) INTO  a_out;
      END IF;
      RETURN OLD;

END;
```

```
CREATE TRIGGER
trg_obj_compre_stream
    AFTER DELETE OR UPDATE ON
object_comprehensive_dialog
    FOR EACH ROW
    EXECUTE PROCEDURE
delete_obj_compre_stream();
```

```
DELIMITER |
CREATE TRIGGER
trg_obj_compre_stream
       AFTER DELETE ON
object_comprehensive_dialog
      FOR EACH ROW
            BEGIN
                  DECLARE  a_out
VARCHAR(200);
                  DECLARE  flag
BOOLEAN;

            SET flag = TRUE;

IF OLD.OBJ_COMPRE_STREAM  IS NOT
NULL AND OLD.OBJ_COMPRE_STREAM > 0
AND flag THEN
                  SELECT
lo_unlink(OLD.OBJ_COMPRE_STREAM)
INTO  a_out;
            END IF;

END;
|
DELIMITER ;

DELIMITER |
CREATE TRIGGER
trg_obj_compre_streamB
      AFTER UPDATE ON
object_comprehensive_dialog
      FOR EACH ROW
            BEGIN
       DECLARE  a_out
VARCHAR(200);
            DECLARE  flag BOOLEAN;

            SET flag = TRUE;

            IF
NEW.OBJ_COMPRE_STREAM =
OLD.OBJ_COMPRE_STREAM THEN
                  SET flag =
FALSE;
            END IF;
IF OLD.OBJ_COMPRE_STREAM  IS NOT
NULL AND OLD.OBJ_COMPRE_STREAM > 0
AND flag THEN
                  SELECT
lo_unlink(OLD.OBJ_COMPRE_STREAM)
INTO  a_out;
            END IF;
       END;
|
DELIMITER ;
```

Tabla 3.1.3 Instrucciones para la generación de funciones y triggers:

Para migrar las funciones y los triggers fue necesario realizar cambios más complejos; debido a que MySQL no maneja el llamado a funciones como Postgres y el manejo de triggers de MySQL es muy rudimentario fue necesario integrar ambos. Recordemos que un trigger o disparador es un evento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción (INSERT), actualización (UPDATE) o borrado (DELETE). En la tabla se muestra el trigger "trg_obj_compre_stream" y su condición para dispararse es "AFTER DELETE OR UPDATE" sin embargo MySQL no soporta esa instrucción; entonces es necesario realizar dos triggers por separados, uno que contiene la instrucción "AFTER DELETE" y otro con la instrucción "AFTER UPDATE". La otra problemática que se presenta, es que el trigger de Postgres hace un llamado a la funcion "delete_obj_compre_stream" mediante la instrucción "EXECUTE PROCEDURE", sin embargo una vez mas, MySQL no soporta dicha instrucción, por lo que es necesario incluirla directamente en los triggers.Se realizaron los cambios a 28 funciones y 27 triggers.

A continuación se muestra un ejemplo del cambio en la sintaxis de una tabla con los tipos de datos en Postgres y su equivalente en MySQL:

Postgres	MySQL

`CREATE TABLE student_grade (`
`    gd_id character varying(3) NOT NULL,`
`    gd_name character varying(100),`
`    gd_desc text,`
`    gd_order integer`
`);`
	`CREATE TABLE student_grade (`
	`    gd_id  varchar(3) NOT NULL,`
	`    gd_name  varchar(100),`
	`    gd_desc text,`
	`    gd_order integer`
	`) TYPE=INNODB;`

Tabla 3.1.4 Instrucciones para la generación de tablas.

En este ejemplo podemos ver que en la tabla que tiene la sintaxis de Postgres, se tiene los tipos de datos varying, text e integer; así que para poder transformar dicha instrucción a la sintaxis de MySQL, buscamos su equivalente en la tabla y podemos notar que el tipo de dato varying( ) de Postgres corresponde a varchar( ) de MySQL, y que text es igual para ambos; así como integer.

MySQL tiene diferentes tipos de tabla; los cuales se describen brevemente a continuación:

- ISAM.- es el formato de almacenaje mas antiguo, y posiblemente pronto desaparecerá. Presentaba limitaciones (los archivos no eran transportables entre máquinas con distinta arquitectura, no podía manejar archivos de tablas superiores a 4 gigas) [MySQLDoc,05].
- MYISAM: es el tipo de tabla por defecto en MySQL desde la versión 3.23. Permite archivos de mayor tamaño que ISAM. Ofrece la posibilidad de indexar campos BLOB y TEXT [MySQLDoc,05].
- HEAP: Crea tablas en memoria. Son temporales y desaparecen cuando el servidor se cierra; a diferencia de una tabla TEMPORARY, que solo puede ser accedida por el usuario que la crea, una tabla HEAP puede ser utilizada por diversos usuarios [MySQLDoc,05].
- MERGE : Una colección de tablas MyISAM usadas como una tabla [MySQLDoc, 05].
- BDB: Tablas transaccionales que soportan AUTOCOMMIT Y ROLLBACK [MySQLDoc, 05]

- INNODB: InnoDB provee a MySQL con el soporte para trabajar con transacciones, sus principales características son :

- Recuperación automática ante fallas. Si MySQL se interrumpe su funcionamiento de forma anormal, InnoDB automáticamente completará las transacciones que quedaron incompletas. [MySQLDoc,05].

- Integridad referencial. Ahora se pueden definir llaves foráneas entre tablas InnoDB relacionadas para asegurarse de que un registro no puede ser eliminado de una tabla si aún está siendo referenciado por otra tabla. [MySQLDoc,05].

- SELECTs sin bloqueo. El motor InnoDB usa una técnica conocida como multi-versioning (similar a PostgreSQL) que elimina la necesidad de hacer bloqueos en consultas SELECT muy simples. Ya no será necesario molestarse porque una simple consulta de sólo lectura está siendo bloqueada por otra consulta que está haciendo cambios en una misma tabla[MySQLDoc,05].

Para la migración se decidió usar el tipo de tabla INNODB por todas las ventajas que ofrece y que ya fueron descritas. Todos estos tipos de cambios se deben realizar para todas las tablas.

Se realizaron los cambios a 101 tablas.

Para explicar el proceso de cambio de sintaxis a las vistas se utilizará el siguiente ejemplo:

Postgres	MySQL
CREATE VIEW all_stream AS SELECT object_book.obj_stream AS sid, 'OBJECT_BOOK' AS tname FROM object_book WHERE (((object_book.obj_type)::text = 'JBookImage'::text) AND (object_book.obj_stream > (0)::oid));	CREATE VIEW all_stream AS SELECT obj_stream AS sid, 'OBJECT_BOOK' AS tname FROM object_book WHERE ((obj_type LIKE 'JBookImage') AND (obj_stream LIKE '%'));

Tabla 3.1.5 Instrucciones para la generación de vistas

En la tabla anterior se pueden ver varias diferencias entre ambas sintaxis; la primera y mas notable es que en Postgres se debe especificar en la instrucción WHERE la tabla de donde proviene el campo referido así como el tipo de dato al que se esta refiriendo mediante "**::**" ; por ejemplo, en la instrucción "(object_book.obj_type)::text" : "object_book" es el nombre de la tabla, "obj" es el campo y ambos se encuentran separados por un "."; y "::text" nos indica que obj es de tipo text. La instrucción completa es: (object_book.obj_type)::text = 'JBookImage'::text. Sin embargo para la sintaxis de MySQL no es necesario incluir en la instrucción WHERE la tabla a la que pertenece, ni el tipo de dato, por lo que en MySQL quedaría: WHERE (obj_type LIKE 'JBookImage') . Hay otra diferencia evidente en esta instrucción que corresponde al operador lógico de comparación; el cual para Postgres es "=" y para MySQL es "LIKE". Se realizaron los cambios a 167 vistas.

Ya que se tuvo el dump de postgres cambiado en su totalidad a la sintaxis de MySQL el siguiente paso es a crear una base de datos en MySQL y vaciar el esquema contenido en el dump, utilizando la instrucción source. Con esta intrucción MySQL ejecuta todas las instrucciones que se encuentren en el archivo especificado en el source.

Para llevar a cabo este vaciado se procedió de la siguiente manera:

- Se creo una base de datos en MySQL con el nombre de isystemdb, para esto se teclea desde una terminal de MySQL la siguiente instrucción:

```
create database isystemdb;
```

- Ahora es necesario crear primero las tablas, por lo que se debe separar en un archivo de texto todas los comandos SQL del archivo anterior encargados de la generación de tablas.

- En la Terminal de MySQL se teclea la siguiente instrucción:

```
source C:\\RUTA_DEL_ARCHIVO\NOMBRE.TXT
```

Donde RUTA_DEL_ARCHIVO es la ruta donde se encuentra el archivo al que se le hicieron las modificaciones, y NOMBRE.TXT es el nombre del archivo y su extensión. Una vez hecho esto, ya tenemos adentro de la base de datos las tablas, sin embargo todavía no estan los datos dentro de la base de datos;

- El siguiente paso es llenar la base de datos, paso que se lleva a cabo con la ayuda del programa llamado DBConnection.java el cual se incluye en el anexo. Se ejecuta el programa con la instrucción: javac C:\\RUTA\DBConnection.java. Cabe mencionar que el proceso de llenado es tardado.

- Una vez llena la base de datos se pueden introducir las vistas, llaves, triggers, indices y funciones con un source de los archivos de texto que contienen los comandos modificados a la sintaxis de MySQL:

Desde la terminal de MySQL se teclea la instrucción

```
source    C:\\RUTA\\ARCHIVO.txt
```

El programa encargado del vaciado de la base de datos de Postgres a MySQL se llama DBConnection.java y tiene la siguiente estructura:

DBConnection.java
- connection() - guarda_info() - guarda_col()

Tabla 3.1.6 Funciones del Programa DBConnecrtion.java

Dicho programa está compuesto por tres que a continuación se explican:

- connection(): Esta clase es la encargada de establecer la conección tanto de la base de datos de postgres (isystemDB) , como la de MySQL (isystemdb).

-guarda_info(): Se encarga de extraer los datos de la tabla en turno y mandarlos a guarda_col

-guarda_col(): Función encargada de construir un query para insertar los datos de la tabla de Postgres a su respectiva tabla de MySQL.

Al ejecutar dicho programa se logró introducir todos los datos de la base de datos utilizada anteriormente, a la nueva base de datos en MySQL. El código del programa anterior se incluye en el anexo de éste documento.

Al revisar la base de datos ya una vez migrada (utilizando MySQL Query Browser visor de resultados obtenidos al ejecutar una sentencia SQL) se pudo notar que varios de los campos existentes no eran utilizados, sin embargo no se creyó conveniente borrarlos ya que no se sabia si alguno de los libros interactivos o tutores no analizados hacia uso de éstos.

También fueron eliminados registros repetidos del las tablas, los cuales solo ocupaban espacio extra dentro de la base de datos.

En la base de datos también son almacenadas algunas imágenes las cuales son utilizadas por los libros interactivos. Con el programa anterior tanto las imágenes como los sonidos no fueron migrados correctamente, debido a la manera en que se encuentran

almacenados en la base de datos de Postgres. Postgres almacena los datos de forma binaria en una tabla aparte y se refiere a éstos mediante un identificador OID (Object Identifier) que se encuentra en la tabla original.

Para poder llevar a cabo la migración de los sonidos y las imágenes, fue necesario recuperar los datos a través del OID y almacenarlos en un array de bytes de manera temporal, para después insertarlos en su tabla equivalente de la base de datos de Mysql. DBConnection.java no realizaba correctamente la migración de los sonidos ni imágenes por lo que se creó otro programa para realizar esta tarea. El programa encargado de realizar ésta tarea se llama ObjetoLargo.java y se encuentra en el anexo de éste documento.

Una vez teniendo la información ya en la base de datos de MySQL el siguiente paso fue comenzar la conexión del sistema con MySQL.

3.2 Conexión del sistema con MySQL para la autentificación del usuario

Una vez que se tuvo la base de datos en MySQL se debe a modificar los archivos que utiliza la pagina index2.php para llevar a cabo la autentificación del usuario, y tambien aquellos que contienen funciones que son invocadas por otros archivos PHP.

Dichos archivo se encuentran en C:\Archivos de programa\Apache Group\Apache\htdocs\beginweb\lib\inc\ , y son sql.inc, forms.inc y system.inc. Estos archivos contiene las funciones necesarias para realizar diferentes acciones con la base

de datos, estas funciones se encuetran diseñadas para trabajar con PHP y postgres, por lo que es necesario realizar su modificación para que trabaje con PHP y MySQL.

La función que se tomará como ejemplo se muestra a continuación:

PHP,Postgres	PHP,MySQL

function getGroupStoryName($database , $user_id , $group_story_id)
{
$sql = "SELECT group_name FROM GROUP_STORY WHERE user_id = " . Quote($user_id) . " AND group_story_id = " . Quote($group_story_id);
$result = pg_exec($database,$sql);
if(($row = pg_fetch_array($result))) {
return $row['group_name'];
} else {
return "";
}
}
	function getGroupStoryName($database, $user_id , $group_story_id)
	{
	$sql = "SELECT group_name FROM GROUP_STORY WHERE user_id = " . Quote($user_id) . " AND group_story_id = " . Quote($group_story_id);
	$result = mysql_query($sql,$database);
	if(($row = mysql_fetch_array($result))) {
	return $row['group_name'];
	} else {
	return "";
	}
	}

Tabla 3.2.1 Ejemplo de función contenida en el archivo index2.php

En la función que trabaja con Postgres podemos notar dos instrucciones importantes, pg_exec y pg_fetch array, cuyos equivalentes para MySQL son MySQL_query y MySQL_fetch_array. Tanto pg_execute como MySQL_query son las instrucciones que ejecutan el query que es mandando mediante la variable $sql; y pg_fetch_array y MySQL_fetch array son las intrucciones que obtiene un array de lo que la variable $result regresa.

Es necesario buscar las equivalencias que se tienen para cada una de las instrucciones que no son compatibles en MySQL; una vez hecho esto se guardan las

modificaciones y se reemplaza el archivo anterior. Este procedimiento se debe seguir con los otros dos archivos con extensión inc, así como los archivos PHP en que sea necesario.

Una vez realizado este paso, el siguiente paso consistió en modificar los archivos JAR que cada tutor maneja para que funcionen con MySQL.

3.3 Conexión de los tutores, libros interactivos con MySQL

Para poder llevar a cabo este paso, se recurrió al uso de la ingeniería inversa. Al ser decompilados los archivos JAR fue posible recuperar los archivos .class y mediante ingeniería inversa se obtuvieron los .java de los mismos.

El primer jar al que se le aplicó ingeniería inversa fue el CSLR.jar. Para extraer todos los archivos del CSLR.jar se teclea en una terminal del sistema la siguiente instrucción:

```
Jar xf  CSLR.jar
```

- La opción x indica que se desea extraer los archivos de JAR
- La opción f indica que el nombre del archivo JAR es especificado en esa linea de comando.
- CSLR.jar el nombre del JAR del cual se desean extraer los archivos. En caso de que el JAR se encuentre en una ruta diferente, es necesario especificarla.

Este commando extraerá todos los archivos contenidos en el jar, poniendolos dentro de los directorios en los cuales fueron guardados originalmente, para el CSLR.jar

el directorio principal es “edu”, y dicho directorio contiene los subdirectorios listados en la Tabla 3.3.1.

JAudioServer
JCSLRLayoutManager
JCSLRMath
JCSLRSwing
JCUAnimate
JIO
JListener
JNative
JNet
JRecog
JSQL
JSystem
JTracker
JTTS
JTutorUtil
JUtil
parameters
Sonic

Tabla 3.3.1 Contenido del CSLR.jar

En los directorios JAudioServer, Sonic, JSQL y JTutorUtil se encuentran los códigos necesarios para la grabación y recuperación de los sonidos de la base de datos del sistema del CSLR.

JCUAnimate con los códigos encargados de la animación de los agentes animados.

JTTS contiene los códigos necesarios para el funcionamiento del sintetizador de voz.

El directorio JSystem contiene el archivo llamado JSystem.class, en dicho archivo se encuentra la función llamada connectSystem(String dbServer) , funcion que se modificó para que hiciera la conexión a la base de datos de MySQL mediante la instrucción “DriverManager.getConnection (url,username, password)” especificando el nombre de la base de datos en “url”, el nombre de usuario para accesar a esa base de datos “username” y el password.

La función modificada se muestra a continuación:

```
/*********************************************************************/

        public static Connection connectSystem(String dbServer)
              throws SQLException
           {
                String url="jdbc:mysql://localhost:3306/isystemdb";
                String username = "root";
                String password = "saviaga";

                return DriverManager.getConnection(url, username, password);
           }
/*********************************************************************/
```

Para lograr que el sistema buscara los sonidos en la base de datos de MySQL fue necesario modificar el archivo JTutorSound que se encuentra en el directorio JTutorUtil del CSLR.jar;

JTutorSound originalmente buscaba el se sonido a través de su OID, lo recuperaba y lo regresaba al JFourSquare, así que cuando el sonido ya se encontraba en la base de datos de MySQL solo era necesario recuperar los datos y mandarlos a JFourSquare, a continuación se muestra la parte del código modificada.

JFourSquare manda a la función getJCSLRSOBStream de JTutorSound la conexión a la base de datos, el texto correspondiente al sonido que desea buscar en la base de datos, una bandera y una cadena con el lenguaje en el que se encuentra dicho sonido;

Dependiendo de las opciones enviadas ésta función manda a llamar a las funciones:
getDefaultStreamByWord
getMXStreamBySentence,
getMXDefaultStreamByWord,
getDefaultStreamBySentence
getDefaultStreamByNoneWord.

Las cuales se encargan de buscar el sonido en las diferentes tablas de sonidos que tiene la base de datos.

Los cambios que se realizaron a las funciones anteriores fueron los referentes a la recuperación de los sonidos, y a continuación se explican tomando como ejemplo la función getDefaultStreamByWord.

En getDefaultStreamByWord antes de migrar la base de datos el array abyte0 mandaba a llamar a la función getBytes que se encontraba en el archivo JSQL, ésta funcion recibia como parámetros , la conexión de la base de datos de Postgres y un OID que se encontraba almacenado en el campo W_M_STREAM de la base de datos WORD_SOUND_DEFAULT. Este OID era obtenido por medio de la función getInt que se le aplicaba al resultset obtenido en esa misma función. La función original era :

abyte0 = JSQL.getBytes(connection, resultset.getInt("W_M_STREAM"));

Para poder adaptar la función anterior a la base de datos se modifico ya que como el sonido ya esta almacenado directamente en la base de datos de MySQL solo fue necesario obtener dicho sonido en forma de bytes y almacenarlo en byte0, la instrucción quedó asi: abyte0 = resultset.getBytes("W_M_STREAM");

La misma modificación se realizó para el campo W_F_STREAM, y para las funciones getMXStreamBySentence, getMXDefaultStreamByWord, getDefaultStreamBySentence o getDefaultStreamByNoneWord.

```
/*************************************************************************/

private static JCSLRSOBStream getDefaultStreamByWord(Connection connection, String s, boolean flag)
            throws Exception
        {
            PreparedStatement preparedstatement =
            connection.prepareStatement("SELECT W_M_STREAM , W_F_STREAM
            FROM WORD_SOUND_DEFAULT WHERE UPPER(TRIM(WORD)) = ?
            ");
            preparedstatement.setString(1, s.trim().toUpperCase());
            ResultSet resultset = preparedstatement.executeQuery();
            if(resultset.next())
              {
                 JCSLRSOBStream jcslrsobstream = new JCSLRSOBStream();
                byte abyte0[];
                 if(flag)
                    abyte0 = resultset.getBytes("W_M_STREAM");
                 else
                    abyte0 = resultset.getBytes("W_F_STREAM");
                 if(abyte0 != null)
              {
                      jcslrsobstream.openStream(abyte0);
                      return jcslrsobstream;
              }
            }
              return null;
        }

/*************************************************************************************/
```

Una vez hecho esto fue necesario compilar los códigos y volver a generar el CSLR.JAR, firmarlo y reemplazar el anterior con la nueva versión.

Una vez que se compiló el archivo JSystem, el siguiente paso fue generar el CSLR.jar nuevamente con el archivo modificado y compilado.

3.4 Generación de un archivo JAR y proceso para firmar el mismo.

Para generar un JAR desde una terminal de sistema se teclea:

```
Jar cf  CSLR.jar  edu
```

- La opción ***c*** indica que se quiere crear un archive JAR.
- La opción **`f`** indica que se que se desea que el resultado vaya al JAR especificado.
- **`CSLR.jar`** es el nombre que se desea que tenga el JAR a generear.
- **`edu`** es el nombre del directorio en donde se encuentran los demás subdirectos que contienen los archivos a guarder en el JAR.

Este comando crea el JAR y lo coloca en el directorio desde donde se tecleó el comando.Una vez generado el JAR, es necesario firmarlo. Un JAR es firmado para garantizar la seguridad de los archivos que se encuentran contenidos en este; por ejemplo, si se envían electrónicamente a otro usuario a través de un applet , el destinatario necesita estar seguro que la información que esta recibiendo es la que originalmente fue enviada, y que dicha información no fue modificada en el proceso de envío por alguien mas. Una vez que el archivo es verificado, el applet le permitirá ejecutar operaciones que normalmente prohibiría.

Java permite firmado y verificación de archivo JAR usando llaves privadas y llaves públicas, las cuales son complementarias. Las llaves privadas se utilizan para firmar el archivo, dicha llave es conocida por quien firma el JAR. Un JAR que es firmado con una llave privada puede ser verificado solamente con la llave pública correspondiente.

Ambas llaves no son suficientes para verificar un applet firmado; para esto, todavía es necesario confirmar que la llave pública realmente proviene de quien firmó el applet. Para esto se utilizan los certificados, que quien firma el applet lo incluye en el archivo JAR. Este certificado es una declaración firmada por una autoridad de certificación (firmas especialistas en seguridad digital) que indica a quien pertenece cierta llave pública. En este caso a quien pertenece la lleve pública contenida en el archivo JAR.

Cuando un archivo JAR es firmado, su llave pública es almacenada en el archivo junto con un certificado asociado, para que pueda ser utilizado por cualquiera que desee verificar esa firma.

Para poder firmar un archive JAR primero se debe tener una llave privada. Las llaves privadas y su respectiva llave pública son almacenadas en bases de datos protegidas con password llamadas keystores. Cada llave que se encuentra en las keystores pueden ser identificadas por un alias que normalmente es el nombre de quien posee dicha llave. [TUTORIAL JAVA, 04].

El primer paso para poder firmar el CSLR.jar es generar un par de llaves, mediante la herramienta keytool, esto se logra tecleando la siguiente instrucción desde una terminal del sistema:

```
keytool -genkey -alias firmadigital -keypass
passllave  -keystore sav158 -   storepass store158
```

Donde las opciones son:

- **`-genkey`** es el commando para generar las llaves.
- **`firmadigital`** es el alias que se utilizará para referirse a las llaves contenidas en el keystore.
- **`passllave`** es el password para la llave privada.
- **`sav158`**s es el nombre del keystore
- **`store158`** es el password del keystore.

Ahora que ya se tienen las llaves, el siguiente paso es firmar el JAR, esto se logra con la siguiente instrucción:

```
jarsigner -keystore sav158 CSLR.jar Count.jar
firmadigital
```

Donde:

- **`jarsigner`** es la instrucción para firmar el jar
- **`keystore`** indica que las llaves se encuentran el archivo de almacenamiento sav158
- **`CSLR.jar`** es el archivo a firmar
- **`firmadigital`** es el alias utilizado para referirse a las llaves

Una vez que se firmó el archivo el siguiente paso es exportar un certificado de seguridad:

```
keytool -export -keystore sav158 -alias firmadigital
-file certificado.cer
```

Una vez hecho esto, es necesario importar dicho certificado para que el JAR pueda ser ejecutado:

```
keytool -import -alias firma digital -file
certificado.cer
```

Ahora el JAR ya se encuentra firmado y puede ser utilizado por el sistema. Y éste ya puede hacer la conexión a la base de datos de MySQL.

Sin embargo es importante mencionar que el manejador de bases de datos que se utilizaba originalmente era Postgres y que la sintaxis de postgres y MySQL para los queries no es la misma, por lo que fue necesario decompilar los codigos de los tutores y libros interactivos y hacer las modificaciones necesarias al código con respecto a los queries que se encuentran especificados en los mismos.