Apéndice E: Manual para el desarrollador

1. Introducción

Este manual explica detalladamente los pasos que se deben seguir para llevar a cabo la migración del sistema del CSLR.

La migración se lleva a cabo tomando los siguientes pasos: migrar la base de datos de Postgres a Mysql; hacer las modificaciones correspondientes a los archivos PHP para que trabajen con la nueva base de datos en Mysql; y por último se debe de modificar los archivos de los tutores que se desean migrar.

2. Migración de la base de datos

En esta sección se describirá paso a paso como se lleva a cabo la migración de la base de datos. Es necesario señalar que no se conoce la estructura de la base de datos; es decir, no se cuenta con un diagrama entidad-relación. Sin embargo, es posible obtener las instrucciones que originalmente fueron utilizadas para su creación mediante la ayuda de un comando llamado pg_dump, que nos sirve 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, triggers, funciones, llaves, índices y vistas.

Este comando se utiliza de la siguiente manera:

```
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, el cual en este caso es “test”.

- isystemDB: es el nombre de la base de datos

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

El archivo generado como ya se mencionó contiene comandos SQL capaces de recrear la base de datos; sin embargo dichos comandos tienen la sintaxis de Postgres. Debido a esto, una vez generado el archivo, es necesario modificar la sintaxis de cada una de los comandos y cambiarla a la equivalente en Mysql; también es necesario revisar los tipos de datos manejados en postgres y encontrar su equivalente en Mysql.

2.1 Migración de las tablas

Como primer paso para la migración de las tablas, se deben cambiar las instrucciones correspondientes a la creación de éstas; a continuación se muestra un ejemplo de dicha transformación:

A continuación se muestra 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 2.1.1 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. Este tipo de cambios se deben realizar para todas las tablas.

Ahora ya tenemos las tablas con la sintaxis de Mysql; el siguiente paso que se explicará es el cambiar la sintaxis de las vistas.

2.2 Migración de las vistas

Para explicar el proceso de migración de 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 >	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 '%'));

(0)::oid));	

Tabla 2.2.1 Instrucciones para la generación de las 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

2.3 Migración de los índices

Ahora se mostrará un ejemplo de cómo migrar los índices:. "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 btrees de alta concurrencia de Lehman-Yao.

RTREE: Implementa rtrees estándar utilizando el algoritmo de partición cuadrática de Guttman.

HASH: Una implementación de las dispersiones lineales de Litwin.

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. La sintaxis de los cambios se pueden ver en la tabla 2.2.1

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 2.3.1 Instrucciones para la generación de índices

2.4 Migración de las llaves

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

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

Tabla 2.4.1 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"

2.5 Migración de funciones y triggers

Funciones y triggers: Para llevar a cabo la migración de los triggers se muestra 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 ;
```

2.5.1 Instrucciones para la generación de funciones y triggers

Para migrar las funciones y los triggers fue necesario realizar cambios mas 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. 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.

Ya que se realizó 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.

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.

Para la migración de los sonidos se utiliza el archivo llamado ObjetoLargo.java ejecutando la siguiente instrucción:

```
javac C:\\RUTA\ObjetoLargo.java.
```

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
```

2.6 Migración de los archivos PHP

Ahora que se tiene 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; en este manual solo se muestra un ejemplo de cómo fue modificada una función. 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 2.51 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.

2.7 Migración de los tutores

Los códigos de los tutores se encuentran almacenados en archivos JAR, y su funcionamiento también depende de otros archivos JAR que se encargan de manejar el sintetizador de voz y la base de datos. Es por eso que se deben de realizar modificaciones a los mismos. Para recuperar los códigos es necesario recurrir a la ingenieria inversa, por medio de un decompilardor.

El primer jar al que se debe decompilar es el CSLR.jar , el cual contiene el archivo llamado JSystem, el cual a su vez, contiene la clase que conecta al sistema con Postgres, clase que debe ser modificada para que funcione con MySQL. Dicho archivo

es utilizado por la mayoría de los tutores para llevar a cabo la conexión con la base de datos.

En JSystem encuentra la función llamada connectSystem(String dbServer) , funcion que se debe modificar para que haga 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.

```
/*********************************************************************/
        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);
          }
/*********************************************************************/
```

Es necesario aclarar que no todos los tutores utilizan a JSystem.java para hacer la conexión a la base de datos, algunos lo hacen por su propia cuenta dentro de su código, así en esos casos si es necesario realizar los cambios correspondientes a la conexión de la base de datos dentro del código del tutor.

Para lograr que el sistema busque los sonidos en la base de datos de MySQL es 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 tutor, así que como el sonido ya se encuentra en la base de datos de MySQL solo es necesario recuperar los datos y mandarlos al tutor, a continuación se muestra la parte del código modificada.

Los tutores mandan 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 que se deben a las funciones anteriores son 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 encuentra 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 debe modificar ya que como el sonido ya esta almacenado directamente en la base de datos de MySQL solo es necesario obtener dicho sonido en forma de bytes y almacenarlo en byte0, la instrucción queda asi: abyte0 = resultset.getBytes("W_M_STREAM");

La misma modificación se debe realizar 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 es necesario compilar nuevamente dicho archivo para generar el .class, y así poder volver a generar el JAR y reemplazar el anterior con la nueva versión.

Una vez que se compiló el archivo JSystem, el siguiente paso es generar el CSLR.jar nuevamente con el archivo modificado y compilado; para esto 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. 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 importer 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 que utilizan los tutores no es la misma, por lo que es necesario decompilar los códigos 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, para que funcionen con la sintaxis de Mysql.

3. Tabla de equivalencias de tipos de datos de Postgres y Mysql

Tipos de datos de Postgres y MySQL		
Tipo en Postgres	MySQL	Descripción
bool	boolean	valor lógico o booleano (true/false)
char(n)	CHAR(N)	cadena de carácteres de tamaño fijo
date	DATE	fecha (sin hora)
float4	FLOAT	número de punto flotante con precisión 86#86
float8	DOUBLE, REAL	número de punto flotante de doble precisión
int2	SMALLINT	entero de dos bytes con signo
int4	INT, INTEGER	entero de cuatro bytes con signo
int8	BIGINT	
varchar(n)	VARCHAR(N)	cadena de carácteres de tamaño variable
text	TEXT	