UNIVERSIDAD VERACRUZANA

FACULTAD DE ESTADÍSTICA E INFORMÁTICA

EXPERIENCIA EDUCATIVA

BASES DE DATOS II

FECHA

27 DE MAYO DE 2014

ESTUDIANTES

OSVALDO CÓRDOVA ABURTO
NORMA ELADIA HERNÁNDEZ SÁNCHEZ
FREDDY ÍÑIGUEZ LÓPEZ
YELENA ASTRÍD RUÍZ PUCHETA

PROYECTO 4 – COMO&COMO

Contenido

DESCRIPCIÓN

El presente documento pretende cubrir los aspectos técnicos de cómo se lleva a cabo el proceso de replicación en las bases de datos entre las interfaces Java desarrolladas en el proyecto pasado. De manera que al final de la configuración, las interfaces Java puedan ser ejecutadas en dos computadoras diferentes y que las actualizaciones a las tablas de la computadora 1 también se vean reflejadas en las tablas de la computadora 2.

Cabe mencionar que los requerimientos del proyecto pedían al menos 2 computadoras físicas replicando la base de datos, pero también podían ser 3 (para lo cual se obtenía una mejor calificación), sin embargo, por cuestiones de tiempo no ha sido posible agregar un tercer equipo al proyecto, por lo que únicamente el desarrollo del presente manual se lleva a cabo con dos computadoras.

CONEXIÓN JAVA – ORACLE

Justo como ocurrió en el proyecto 3 de COMO&COMO, la conexión de la interfaz Java con el manejador de bases de datos de Oracle se llevó a cabo con la ayuda del conector OJDBC que el mismo Oracle pone a disposición en su página Web. De hecho, este actual proyecto (proyecto 4 – Replicación) se ha reutilizado por completo el funcionamiento y recursos del proyecto 3, por lo que para saber más acerca de la conexión entre Java y Oracle puede consultar la documentación del tercer proyecto; de momento nos interesa describir el proceso de la replicación en las bases de datos.

REPLICACIÓN

La replicación consiste en que sin importar el número de computadoras que se encuentren conectadas a una red, todas las bases de datos de estas tengan los mismos datos, es decir, que no suceda que en un sitio existen algunos datos y que en otros los demás. Más específicamente, la replicación es realizar las actualizaciones en el sitio A, y que también se vean reflejados los cambios en el sitio B, en el sitio C,… , en el sitio N.

Por lo anterior, se comenzó a desarrollar una técnica para manejar estas actualizaciones, y la manera en la que todas las interfaces Java conocieran y se dieran cuenta de las actualizaciones que se realizan de manera remota en los demás sitios de la aplicación.

Una solución a esto (y la que se describe en estas páginas) es que exista un 'Servidor' (en bases de datos se le conoce como *gestor de transacciones*[1]) el cual sea el encargado de llevar a cabo el control sobre las actualizaciones a las tablas de la base de datos, y que este a su vez sirviera a un 'Cliente', de manera que cuando el servidor detectara una actualización local le pudiese comunicar sobre esto al cliente remoto, y viceversa, que cuando el Servidor detectara una actualización remota en la interfaz del Cliente, pudiera obtener los datos e implementarlo en las tablas correspondientes en la base de datos local.

Por tanto, para lograr esto, se hizo uso de interfaces de comunicación por red, mejor conocidas como *Sockets*[2], los cuales son capaces de llevar una intercomunicación de procesos sobre una red de computadoras, lo cual es ideal para el propósito del proyecto. Por supuesto que existen otras técnicas mejores/peores que los sockets, pero se han elegido estos últimos debido a su facilidad de uso.

Ahora, los cambios en la clase **ComoyComo.java** son los siguientes:

- Dependiendo del equipo que ejecute la interfaz (se refiere a si es el servidor o el cliente), se define un Socket o un ServerSocket, se referencia con la actual conexión a la base de datos y a la misma interfaz, se crea un hilo para su ejecución y se corre, tal como sigue:

```
s = new Server(conexionBD, interfaz);
Thread t = new Thread(s);
t.start();
```

[1] Para mayor información, consulte http://es.wikipedia.org/wiki/Gestor_transaccional
[2] Encuentre mayor información en http://en.wikipedia.org/wiki/Network_socket

Cabe mencionar que el socket, tanto el Servidor como el Cliente, son hilos (Threads) en java, ya que éstos necesitarán estar buscando por las actualizaciones en las bases de datos.

- La clase **ComoyComo.java** ahora implementa un método muy importante llamado *fillAllTables*, el cuál es el encargado de refrescar todas las tablas en caso de que el gestor de transacciones haya ejecutado una actualización local,

```
public void fillAllTables(){
        fillTableClientes(titClientesAux);
        fillTablePlatillos(titPlatillosAux);
        fillTablePoblacion(titPoblacionAux);
        fillTableCentros(titCentrosAux);
        fillTableEmpleados(titEmpleadosAux);
        fillTableFacturas(titFacturasAux);
}
```

- Y bueno, cada que de manera local se realiza una actualización a la base de datos, se llama al método del gestor de transacciones para que realice este cambio en la base de datos remota, como sigue:

```
s.writeLine(query);
```

Esta línea se escribe cada que se realiza un alta o eliminación de algún registro de cualquier tabla de la base de datos. ‘s’ es el socket Cliente o Servidor.

- Finalmente, para cerrar la aplicación, la clase **ComoyComo.java**, además de cerrar la conexión a la base de datos, ahora también debe preocuparse por la conexión del socket. De manera que en el evento del botón ‘Salir’ se agrega lo siguiente:

```
        try{
                socket.closeConnection();
```

```
                }catch(NullPointerException x){
                        x.printStackTrace();
                }
```

Se define dentro de un try~catch debido a que la intefaz Java se pudo haber ejecutado, pero nunca hubo una conexión con el cliente o el servidor, por lo que el socket puede tener una referencia vacía y detener el programa si se intenta cerrar siendo null.

A continuación se definen el contenido completo de las clases **Cliente** y **Servidor**. Notará que estas dos clases son muy similares, y lo son debido a que su comportamiento es parecido (escribo un mensaje al otro socket cuando detecto una actualización y escucho del otro socket cuando me trae una actualización), sin embargo, existen algunas diferencias que es muy importante definir.

CLASE CLIENTE

```
package comoycomo;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.sql.SQLException;

import javax.swing.JOptionPane;

public class Client implements Runnable{
        private Socket s;
        private ObjectOutputStream oos;
        private ObjectInputStream ois;
        private OracleConnection bdConnection;
        private ComoyComo ventana;

        public Client(OracleConnection bdConnection, ComoyComo ventana){
                this.bdConnection=bdConnection;
                this.ventana=ventana;
        }

        @Override
        public void run() {
                try {
                        s = new Socket(JOptionPane.showInputDialog("Ingrese la direcón IP del servidor"), 9999);
                        oos = new ObjectOutputStream(s.getOutputStream());
                        ois = new ObjectInputStream(s.getInputStream());
                        JOptionPane.showMessageDialog(null, "Conexión exitosa con el servidor", "COMO&COMO
Cliente", JOptionPane.INFORMATION_MESSAGE);
                        this.readLine();
```

```
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        public void readLine(){
                while(true){
                        Object aux;
                        try {
                                aux = ois.readObject();
                                if(aux != null && aux instanceof String){
                                        bdConnection.getStament().executeUpdate(""+aux);
                                        ventana.fillAllTables();
                                }
                                Thread.sleep(30);
                        }catch(InterruptedException | ClassNotFoundException | IOException | SQLException e){
                                e.printStackTrace();
                        }
                }
        }

        public void writeLine(String query){
                try{
                        oos.writeObject(query);
                }catch(IOException e){
                        e.printStackTrace();
                }
        }

        public void closeConenction(){
                try{
                        oos.close();
                        ois.close();
                        s.close();
                }catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```

CLASE SERVIDOR

```
package comoycomo;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.sql.SQLException;

import javax.swing.JOptionPane;

public class Server implements Runnable{
	private ServerSocket ss;		// Socket servidor
	private Socket s;			// Socket cliente
	private ObjectOutputStream oos; // Enviar objetos
	private ObjectInputStream ois;		// Recibir objetos
	private OracleConnection bdConnection;		// Conexión a Oracle
	private ComoyComo ventana;			// Interfaz gráfica

	public Server(OracleConnection bdConnection, ComoyComo ventana){
		this.bdConnection=bdConnection;
		this.ventana=ventana;
	}

	@Override
	public void run() {
		try {
			ss = new ServerSocket(9999); // Crea un nuevo socket servidor
			s = ss.accept();				// Acepta la entrada de un socket cliente
			oos = new ObjectOutputStream(s.getOutputStream());
			ois = new ObjectInputStream(s.getInputStream());
			JOptionPane.showMessageDialog(null, "Conexion exitosa con el cliente", "COMO&COMO
Servidor", JOptionPane.INFORMATION_MESSAGE);
			System.out.println("Servidor~# Conexión exitosa.");
			this.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void closeConnection(){
		try{
			// Se cierran todas las conexiones
			oos.close();
			ois.close();
			s.close();
			ss.close();
		}catch(IOException e){
			e.printStackTrace();
		}
	}

	public void readLine(){
		while(true){
			try{
				Object aux = ois.readObject();
				if(aux != null && aux instanceof String){
```

```
                                        // Realiza el la actualización.
                                        bdConnection.getStament().executeUpdate(""+aux);
                                        // Llama al método que actualiza todas las tablas de la interfaz.
                                        ventana.fillAllTables();
                                }
                                Thread.sleep(30);
                        }catch(InterruptedException | ClassNotFoundException | IOException | SQLException e){
                                e.printStackTrace();
                        }
                }
        }

        public void writeLine(String query){
                try {
                        // Escribe el query en el objeto correspondiente para que
                        // este contenido sea ejecuta posteriormente.
                        oos.writeObject(query);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

Básicamente, las clases contienen lo siguiente:

- Sentencias **imports**, que hacen referencia a las bibliotecas necesarias para la conexión remota del socket o de la entrada y salida de los datos,
- Sentencias **declarativas** de los atributos, sus objetos de lectura y escritura de datos, sockets, la referencia de la base de datos local y de la misma interfaz Java,
- **Constructor**, el cual realiza la referencia de la base de datos local y la interfaz Java que le pasan como parámetros,
- Método **run**, propio del hilo ya que las clases están implementando la interfaz *Runnable*. Es en este método donde se construyen los sockets, la entrada y salida de datos y donde se llama al método *readLine()* que es el encargado de estar buscando por actualizaciones remotas,
- Método **closeConnection**, que es el encargado de realizar la desconexión de los recursos de socket y de él mismo,
- Método **readLine**, encargado de estar revisando si existen actualizaciones remotas; si encuentra una lo que hace es ejecutar la actualización a la base de datos local y de llamar al método de la clase ComoyComo.java que actualiza los datos de las tablas de la interfaz,

- Método **writeLine**, en el cual se define el query que será ejecutado en la base de datos local una vez que se encuentran actualizaciones remotas.

La principal diferencia entre la clase **Cliente** y la clase **Server** es que ésta última, a diferencia de la clase Cliente, define dos tipos de Sockets: uno es el servidor (él mismo) y otro es el cliente (el que se conectará a él y del cual estará consultando por actualizaciones en la base de datos). La clase Cliente solo define un solo socket (el cliente, es decir, él mismo).