PROGRAMA INGENIERÍA DE SISTEMAS
(NUEVO PLAN DE ESTUDIOS)

GUIA DE LABORATORIO

CURSO: SISTEMAS TRANSACCIONALES AREA: Análisis de Datos

Nro. DE LA PRÁCTICA: ____ H. VIRTUALES: __2_ H. INDIVIDUALES: 3

NOMBRE DE LA PRÁCTICA: Diseñar y construir una Base de Datos Distribuida para realizar Transacciones.

OBJETIVO:

Integrar el concepto de transacciones implícitas y explicitas mediante una base de datos en la nube, con el uso del sistema cliente servidor.

OBJETIVOS ESPECÍFICOS:

- Comprender la diferencia entre una transacción implícita y explícita.
- Adecuar la sintáxis usada en las bases de datos a un lenguaje de programación externo.
- Establecer una conexión a la base de datos en la nube.
- Realizar los procesos adecuados y esencialmente requeridos en el lado del Servidor.
- Lograr una conexión entre Cliente Servidor funcional.

TEMÁTICAS:

1. **ADMINISTRATAR TRANSACCIONES EN BASES DE DATOS**
2. **Transacciones Implícitas y Explicitas**
3. **Transacciones y recuentos de referencias**
4. **Iniciar una transacción**
5. **Confirmar una transacción**
6. **Revertir una transacción**

COMPETENCIAS A DESARROLLAR:

1. Comprensión de transacciones implícitas y explícitas.

2. Manejo de sentencias necesarias para ejecución de los dos tipos de transacciones.

3. Habilidad para establecer una correcta conexión a una base de datos en la nube.

CONOCIMIENTOS PREVIOS REQUERIDOS:

1. Declaración y funcionamiento de socket por parte del cliente.
2. Declaración y funcionamiento de socket por parte del servidor.
3. Enviar y recibir datos entre cliente y servidor.
4. Operaciones con datos de cliente en la parte del servidor.
5. Manejo básico de bases de datos.

RECURSOS A NIVEL HARDWARE Y SOFTWARE:

1. Equipo con IDE Java, en este caso.

2. Librerías mysql-connector-java-8.0.19.jar.

3. Conexión a Internet.

DESCRIPCIÓN DE LA PRÁCTICA:

Esencialmente se requería un proveedor que proporcionara una base de datos en la nube. Por lo tanto se seleccionaron los servicios prestados por www.clever-cloud.com.

Posteriormente, la conexión se estableció en el lado del Servidor, por lo que se usó la librería mysql-connector para este fin. También, se necesitó de las credenciales otorgadas por el servidor cloud de mysql (el administrador de bases de datos seleccionado).

Una parte crucial de un servidor es contar con un puerto específico y un socket establecido, por lo que se establecieron estos valores. En este punto la conexión con la base de datos en la nube y la creación del servidor eran completamente funcionales. No obstante, no se estaba modificando ni obteniendo datos de la base de datos, tampoco se establecía una conexión por parte del cliente.

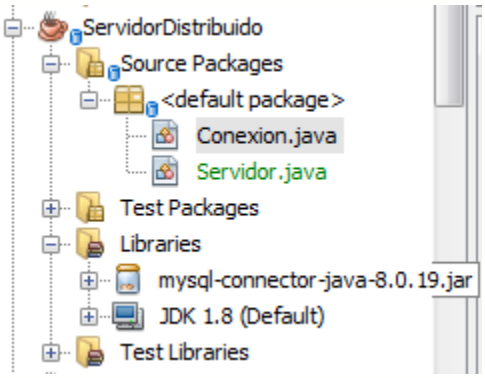
El siguiente paso fue, instaurar los parámetros necesarios para que un cliente pudiese tener una conexión con el Servidor, por lo que se optó por delegar este proceso a un proyecto Java diferente al del Servidor, implantado el puerto, el host y el socket requeridos.

A continuación, se realizaron las sentencias para lograr enviar una petición al Servidor. Adicionalmente, se consiguió obtener respuesta del servidor, por lo que el paso siguiente fue crear funciones que trabajasen para enviar y recibir datos de manera correcta.

Introduciendo este concepto al proyecto, fue necesario crear funciones por parte del Servidor también.

PROCEDIMIENTO O DESARROLLO INGENIERÍL:

Por primera parte, se optó por desarrollar la interfaz del servidor, que cuenta con la conexión a la base de datos en la nube, y con la librería que permite dicha conexión:



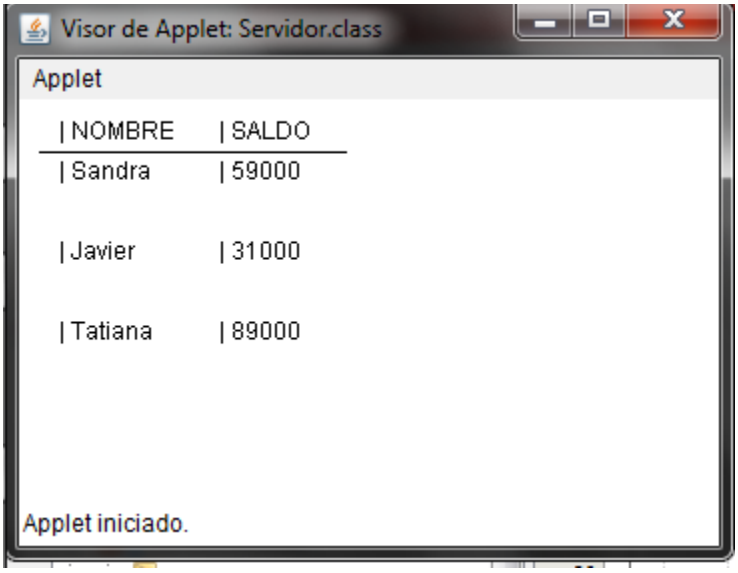
El puerto del Servidor se estableció y se creó el socket servidor que permitirá realizar comunicación con el cliente

```
public class Servidor extends Applet {

    final static int PORT = 5123;
    ServerSocket skServidor;
    Conexion c = new Conexion();

     class Server
     {...152 lines }
}
```

Dentro de la conexión, debió establecerse el parámetro *setAutocommit()* como *false*, pues esta instrucción permite realizar las operaciones explícitas de manera adecuada:

```
public Conexion()
{
    try
    {
        con = DriverManager.getConnection(url, user, password);
        con.setAutoCommit(false);
    } catch (Exception ex) {System.out.println("Error conexión; " + ex);}
}
```

Posteriormente, se programó la instrucción para listar los valores en la base de datos en la nube dentro del Applet del Servidor:



Haciendo uso de la clase Conexión mostrada anteriormente, se muestran los valores con el ciclo *while* contenido en una función:

```
while(c.rs.next())
{
    g.drawString("| "+c.rs.getString("nombre"), x, 2*y);
    g.drawString("| "+c.rs.getString("saldo"), x*5, 2*y);
    y += 20;
    x = 20;
}
```

Luego, se creó la clase Cliente con el mismo puerto de comunicación del Servidor y con su respectivo socket de transferencia de información:

```
public class Cliente extends Applet
{
    Client cliente = new Client();
    Socket skCliente;
    final static String HOST = "localhost";
    final static int PORT = 5123;

    class Client
    {...61 lines }
}
```

Para la visualización inicial del Applet Cliente se usó el siguiente apartado:

```
Label l1 = new Label("-----------------------------------------------
Label labelResultado = new Label("..................................
TextField nameField = new TextField();
TextField valueField = new TextField();
Button b1 = new Button("Realizar transferencia");

public void init()
{
    cliente.setValues();
    add(l1);
    add(nameField);
    add(valueField);
    add(b1);
    add(labelResultado);
}

public void paint(Graphics g)
{
    g.drawString("Usuario: "+cliente.name, x, y);
    g.drawString("Saldo: $"+cliente.balance, x*20, y);
    l1.setLocation(x, y+5);
    g.drawString("Digite nombre de persona a transferir: ",x, y*3);
    nameField.setBounds(x*23, y+28, 57, 21);
    g.drawString("Digite cantidad a trasnferir: ",x, y*4);
    valueField.setBounds(x*23, y*2+28, 57, 21);
    b1.setLocation(x*15, y*6);
    labelResultado.setLocation(x*6, y*8);
}
```

Donde se establece comunicación con el Servidor para obtener los datos iniciales y que más tarde, permitirán realizar los asuntos operacionales de la transacción. La función *setValues()* instaura el socket necesario para la transferencia de datos, recibiendo mediante las sentencias *InputStream* los valores listados a continuación:

```
class Client
{
    String name;
    int balance;

    Client() {}

    /**
     * Obtiene el nombre y el saldo por parte del servidor y los muestra.
     */
    public void setValues()
    {
        try
        {
            skCliente = new Socket(HOST, PORT);
            DataInputStream name = new DataInputStream ( skCliente.getInputStream() );
            this.name = name.readUTF();
            DataInputStream balance = new DataInputStream ( skCliente.getInputStream() );
            this.balance = Integer.parseInt(balance.readUTF());
            skCliente.close();
        } catch (Exception ex) {System.out.println("Error setValues: "+ex);}
    }
```

Donde simultáneamente, el Servidor hace uso del socket, enviando los datos que está esperando el Cliente con las sentencias *OutputStream* y ejecuta la siguiente función:

```
public void sendCurrentValues ()
{
    try
    {
        Socket skCliente = skServidor.accept();
        String currentName = getName(1);
        DataOutputStream name = new DataOutputStream ( skCliente.getOutputStream() );
        name.writeUTF(currentName);
        DataOutputStream balance = new DataOutputStream ( skCliente.getOutputStream() );
        balance.writeUTF(Integer.toString(getBalance(currentName)));
        skCliente.close();
    } catch (Exception ex) {System.out.println("Error sendValues: "+ex);}
}
```

Obteniendo los valores *name* y *balance* haciendo uso de la clase Conexión, ejecutando sentencias de mysql para realizar las consultas necesarias:

```
public int getBalance(String name) throws SQLException
{
    try
    {
        c.ps = c.con.prepareStatement("SELECT * FROM cuen
        c.rs = c.ps.executeQuery();
        c.rs.next();
        this.successfulTransaction = true;
        return c.rs.getInt("saldo");
    } catch (Exception ex) {
        this.successfulTransaction = false;
        c.con.rollback();
        System.out.println("Error getBalance: " + ex);
        return -1;
    }
}

  public String getName(int id) throws SQLException
  {
      try
      {
          c.ps = c.con.prepareStatement("SELECT * FROM
          c.rs = c.ps.executeQuery();
          c.rs.next();
          this.successfulTransaction = true;
          return c.rs.getString("nombre");
      } catch (Exception ex) {
          this.successfulTransaction = false;
          System.out.println("Error getName: " + ex);
      }
      return null;
  }
```

El booleano *successfulTransaction* funciona como indicador de si la transacción tuvo éxito o no, que posteriormente se usará para indicarle al cliente el estado de la operación. A su vez, se usa el *rollback()* porque pueden existir fallos realizando la transacción de una cuenta a otra, en la siguiente función:

```
public void executeTransaction() throws SQLException, IOException
{
    Socket skCliente = skServidor.accept();
    try
    {
        DataInputStream name = new DataInputStream ( skCliente.getInputStream() );
        String currentName = name.readUTF();
        DataInputStream name1 = new DataInputStream ( skCliente.getInputStream() );
        String destinationName = name1.readUTF();
        DataInputStream value = new DataInputStream ( skCliente.getInputStream() );
        int amount = Integer.parseInt(value.readUTF());
        updateBalance(currentName, amount*-1);
        updateBalance(destinationName, amount);
        c.con.commit();
    } catch(Exception ex) {
        this.successfulTransaction = false;
        c.con.rollback();
        System.out.println("Error executeTransaction: "+ex);
    } finally {
        sendStatusTransaction(skCliente,this.successfulTransaction);
        skCliente.close();
        c.ps.close();
    }
}
```

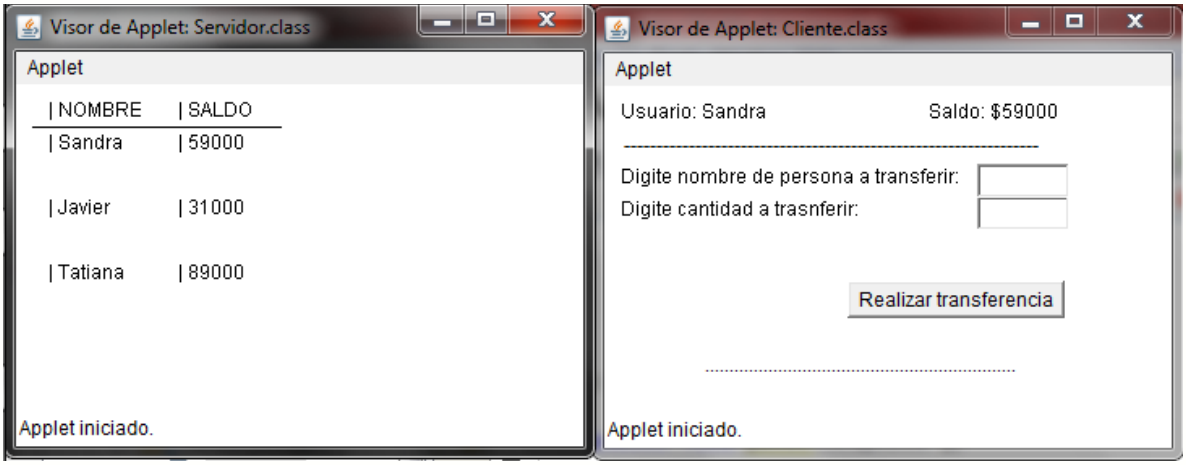
Donde *updateBalance()* permite modificar el valor de saldo en la base de datos:

```
lic void updateBalance(String name, int value) throws SQLException

 try
 {
    c.ps = c.con.prepareStatement("UPDATE cuenta SET saldo="+(getBalance(name)+value)+" WHERE nombre='"+name+"'")
    c.ps.executeUpdate();
 } catch (Exception ex) {
    c.con.rollback();
    System.out.println("Error updateBalance: "+ex);
 }
```

Y a su vez, el resultado de si la transacción fue exitosa o no, es enviado por *sendStatusTransacion()*:

```
public void sendStatusTransaction(Socket skCliente, boolean check)
{
    try
    {
        DataOutputStream checker = new DataOutputStream ( skCliente.getOutputStream() );
        checker.writeBoolean(check);
    } catch (Exception ex) {System.out.println("Error sendStatusTransaction: "+ex);}
}
```

Mientras que por el lado del Cliente, se muestra la siguiente interfaz cuando es ejecutado el Servidor simultáneamente:



Las órdenes son ejecutadas al momento de que sucede el evento de click en el botón *Realizar transferencia*:

```
public boolean action (Event e, Object obj)
{
    if(e.target == b1)
    {
        if (cliente.dataCheck())
        {
            cliente.transaction();
            cliente.setValues();
            repaint();
        }
        else
        {
            labelResultado.setText("Verifique los valores ingresados.");
            repaint();
        }
    }
    return false;
}
```

La sentencia condicional que define si se realizará la ejecución de la transacción o no, está relacionada a la función *dataCheck()*¸que verifica que los datos ingresados no correspondan al mismo nombre de la persona que está realizando la transferencia o que el valor a transferir no tenga un valor que no corresponda a un número entero, o un número negativo.

```
public boolean dataCheck()
{
    try{ return (nameField.getText().equalsIgnoreCase(this.name) || Integer.parseInt(valueField.getText()) <= 0)
    catch (Exception ex){ return false; }
}
```

Una vez los valores son verificados, se procede con el proceso transaccional, que básicamente envía todos los datos correspondientes al Servidor, para después volver a recibirlos y actualizar los datos en la interfaz ejecutando nuevamente la función *setValues()*¸que se enseñó previamente:

```
public void transaction()
{
    boolean successfulTransaction = false;
    try
    {
        skCliente = new Socket(HOST, PORT);
        DataOutputStream currentName = new DataOutputStream ( skCliente.getOutputStream() );
        currentName.writeUTF(this.name);
        DataOutputStream name = new DataOutputStream ( skCliente.getOutputStream() );
        name.writeUTF(nameField.getText());
        DataOutputStream value = new DataOutputStream ( skCliente.getOutputStream() );
        value.writeUTF(valueField.getText());

        DataInputStream check = new DataInputStream ( skCliente.getInputStream() );
        successfulTransaction = check.readBoolean();
        skCliente.close();
    } catch (Exception ex) {System.out.println("Error transaction: "+ex);}
    finally
    {
        if(successfulTransaction)
            labelResultado.setText("Operación extiosa.");
        else
            labelResultado.setText("Verifique los valores ingresados.");
    }
}
```

METODOLOGÍA:

Las bases fundamentales para este proyecto, fueron la realización de la investigación a fondo del concepto de transacciones implícitas y explícitas. Luego, el retó fue aterrizar los conocimientos adquiridos a la sintáxis de la base de datos elegida (mysql). Para esto, se realizaron las investigaciones que se consideraron necesarias para dicho fin, haciendo uso de documentos en línea que demostrasen una capacidad robusta para el estudio del área a trabajar.

Debido a que el entorno de desarrollo incluía una base de datos en la nube, se optó por elegir un proveedor de dicho servicio gratuito en línea, como lo fue el caso de www.clever-cloud.com. Posterior a tener presente la interfaz web a usar, se consultó la documentación imprescindible para un correcto funcionamiento y acoplamiento.

Finalmente, se fabricó el sistema cliente servidor con el IDE de Java de NetBeans, complementando el desarrollo con el conocimiento adquirido en clase.

PRODUCTOS Y RESULTADOS ESPERADOS:

En conclusión, se logró establecer un sistema cliente servidor de tres capas; donde la capa número corresponde al cliente, la segunda al servidor y la tercera a la base de datos en la nube.

Se encontró una correcta ejecución del fin del proyecto, que principalmente era el uso de transacciones implícitas y explícitas. Pues, en los momentos que la transacción explícita presentaba error, lograba realizarse un *rollback* para no ejecutar cambios permanentes que no obedecieran a las propiedades *ACID* de una base de datos.

CRITERIOS DE EVALUACIÓN:

1. **Forma parte del 40% de la calificación.**
2. **Se tiene en cuenta el conocimiento de las definiciones, desarrollo de la práctica, análisis de los resultados, conclusiones y las habilidades, actitud, motivación para realizar la actividad de acuerdo a la rúbrica (Ver abajo).**
3. **Terminación del laboratorio**
4. **Cumplimiento de entrega**
5. **Cumplimiento de los objetivos del laboratorio**

RÚBRICAS DE EVALUACIÓN					
CRITERIOS DE EVALUACION	**4.5 - 5.0**	**4.0 - 4.5**	**3.0 - 3.5**	**2.5 - 3.0**	**CALIFICACION**
1.- Definiciones **Conceptos**	No es copia fiel de los textos consultados sino una síntesis de ideas completas y claras del tema.	Excede a cuatro cuartillas o no alcanza a cubrir una. Algunos párrafos son copias fieles de los textos consultados. Algunas ideas del tema están cortadas.	No se presenta de manera clara y completa. La relación con el problema planteado es prácticamente incongruente. El 60% del tema es copiado.	Realiza un 50% ó 60% de los experimentos, mencionando el procedimiento de manera completa incluyendo el material y equipo utilizado. Las respuestas son congruentes con los experimentos realizados.	

2.- Desarrollo de la Práctica - Dedicación	Realiza todos los experimentos mencionando el procedimiento de manera completa, incluyendo el material y equipo utilizado. Las respuestas son congruentes con los experimentos realizados.	Realiza un 80% de los experimentos, mencionando el procedimiento de manera completa incluyendo el material y equipo utilizado. Las respuestas son congruentes con los experimentos realizados.	Realiza un 50% ó 60% de los experimentos, mencionando el procedimiento de manera completa incluyendo el material y equipo utilizado. Las respuestas son congruentes con los experimentos realizados.	Realiza un 40% de los experimentos, mencionando el procedimiento de manera completa incluyendo el material y equipo utilizado. Las respuestas son congruentes con los experimentos realizados.	
3.- Interpretación, Análisis de los Resultados	Recopila y ordena los datos en relación al procedimiento. Se presentan los datos en tablas, gráficas, dibujos, etc. Claramente identificados. Los datos se interpretan y analizan comparativamente con la información bibliográfica consultada.	Presenta datos ordenados en relación al procedimiento. Se presentan en tablas, gráficas, dibujos, etc. claramente identificados, se interpretan y analizan parcialmente en un 80%.	Tiene datos parcialmente ordenados, presenta algunas tablas o gráficas, los resultados se interpretan y analizan en 50% ó 60%	Tiene datos parcialmente ordenados, presenta algunas tablas o gráficas, los resultados se interpretan y analizan en 50%.	
4.- Conclusión	Deduce el comportamiento de la(s) variable(s) estudiada(s) a partir del problema planteado. Rechaza o acepta la hipótesis e incluye propuestas de mejora o genera nuevos problemas.	Deduce el comportamiento de la(s) variable(s) estudiada(s) a partir del problema planteado. Incluye el rechazo o la aceptación de la hipótesis, pero no las propuestas de mejoras.	Deduce el comportamiento de la(s) variable(s) estudiada(s) a partir del problema planteado. No incluye el rechazo o aceptación de la hipótesis ni propone mejoras.	Deduce el comportamiento de la(s) variable(s) estudiada(s) a partir del problema planteado. No incluye el rechazo o aceptación de la hipótesis ni Propone mejoras.	
5.- Actitud, Motivación, Disciplina	Participa propositiva e integralmente en toda la práctica.	Participa ocasionalmente o lo hace constantemente pero sin coordinarse con su	Es un observador pasivo.	No participa en la realización de la práctica.	

		compañero.			

BIBLIOGRAFÍA LIBROS DIGITALES DE LA UNIVERSIDAD:

North, K. (1999). Distributed transactions, SQL, and application servers. Web Techniques, 4(2), 22-26. Retrieved from https://search-proquest-com.ezproxy.uniminuto.edu/docview/274962979?accountid=48797

Celko, J. (2010). Joe celko's sql for smarties : Advanced sql programming. Retrieved from https://ebookcentral.proquest.com/lib/bibliouniminuto-ebooks/detail.action?docID=610555#

BIBLIOWEB:

https://www2.cs.duke.edu/courses/fall16/compsci316/lectures/11-transaction.pdf
http://myy.haaga-helia.fi/~dbms/dbtechnet/download/SQL-Transactions_handbook_EN.pdf
https://www.clever-cloud.com/doc/addons/mysql/

AUTOR:

Nicolás David Espejo Bernal

ID: 637747