

# Documentación

## Entrega No. 4

---

### Integrantes

- Ricardo Jose Garzon Arias C.c. 1.017.270.496
- Sebastián Rendón Arteaga C.c. 1.001.366.265

### Primer Punto

Para la realización del primer Punto de opto por utilizar el sql\*loader, y para realizar la carga de estos archivos se requiere de gestionar unas direcciones de archivos y configurar unos archivos como en el siguiente ejemplo:

```
load data
infile 'C:\Users\ricardo\Desktop\Trabajo4\Muestraliquies.csv'
append
into table BLOQUE
FIELDS TERMINATED BY "," TRAILING NULLCOLS
(
  id,hash,time,timestamp,miner,extra_data,fee,difficult,gas_used,gas_limit,logs_block,min,hash,nonce,receipts_root,sha3_uncles,state_root,total_diff,
  uncle_count,transaction_count,synthetic_transaction_count,call_count,synthetic_call_count,value_total,value_total_and_interval_value_total,inter
  generation,generation_usd,uncle_generation,uncle_generation_usd,fee_total,fee_total_usd,reward,reward_usd)

```

también se requiere tener la tabla creada y vacía previamente en sql, para luego desde la consola ejecutar el siguiente comando por cada archivo a ejecutar(ojo, por cada archivo se hace un archivo especial de sql\*loader y una tabla en sql): `$sqlldr control =`

```
C:\Users\ricardo\Desktop\Trabajo4\Transacciones.ctl log =
```

```
C:\Users\ricardo\Desktop\Trabajo4\Hola.log userid = ricardo/123
```

y así se cargan los datos a sql.

### Segundo Punto

Este punto fue el más jodido de todos, y debido a que 5 paginas no las consideramos suficientes para poder explicar a la perfección este punto, haremos un breve resumen así:

---

```
try {
    resultado = sentencia
        .executeQuery("SELECT block_id, x, y, sender, recipient, value_usd, fee_usd, time FROM TRANSACCION");
    while (resultado.next()) {
        SimpleDateFormat format = new SimpleDateFormat("HH:mm");
        Date parsed = null;
        Date fechainicio = null;
        Date fechainal = null;
        String fecha = resultado.getString(8).toString();
        // System.out.println(resultado.getString(6).toString());
        try {
            if (fecha.split(" ").length > 1) {
                Integer x = 6 * (Integer.parseInt(resultado.getString(2).toString()));
                Integer y = 6 * (Integer.parseInt(resultado.getString(3).toString()));
                fechainicio = format.parse(resultado.getString(1).toString());
                fechainal = format.parse(resultado.getString(1).toString());
                parsed = format.parse(resultado.getString(1).toString());
                Pair<Integer, Integer> coordenadas = new Pair<Integer, Integer>(x, y);
                ArrayList<String> elementos = new ArrayList<String>();
                if (fechainicio.compareTo(parsed) == 0) {
                    // System.out.println("Inicio igual");
                    elementos.add(resultado.getString(1).toString());
                    elementos.add(resultado.getString(4).toString());
                    elementos.add(resultado.getString(5).toString());
                    elementos.add(resultado.getString(6).toString());
                    elementos.add(resultado.getString(7).toString());
                    elementos.add(resultado.getString(8).toString());
                    transacciones.put(coordenadas, elementos);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

primero se cargan los datos de la base de datos a java, y luego se convierte la fecha en formato fecha para luego tomar solo los datos en el rango de fechas que nos indican.

```
Map<ArrayList<Integer>, java.awt.Color> rangocolores = Punto2.rangocolores(Integer.parseInt(color.getText()),
    color1.getText(), color2.getText(), color3.getText(), color4.getText(), color5.getText(),
    color6.getText(), color7.getText(), color8.getText(), color9.getText(), color10.getText());
GraficoPunto2 graficoPunto2 = new GraficoPunto2(rangocolores, Integer.parseInt(tamaño.getText()), transacciones);
DrawWindow.setSize(800, 800);
DrawWindow.setResizable(false);
DrawWindow.setLocation(200, 50);
DrawWindow.setTitle("Creaditos malucos de manejar");
DrawWindow.setVisible(true);
```

luego se llama otra clase para graficar la cuadrícula y se le mandan los colores correspondientes (en orden del 1 al 10 son: red, yellow,gray,pink,orange,cyan,magenta,brown,blue,green)

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        Integer contador = 0;
        Set<Integer> set = new HashSet<Integer>();
        for (Map.Entry<Integer, Integer> entry : transacciones.entrySet()) {
            if ((entry.getKey().getValue() >= i && entry.getKey().getValue() <= j) ||
                (entry.getKey().getValue() <= i && entry.getKey().getValue() >= j)) {
                contador++;
                set.add(Integer.valueOf(entry.getKey().getValue()));
            }
        }
        if (entry.getKey().getValue() <= i && entry.getKey().getValue() >= j) {
            Integer holas = Integer.parseInt(entry.getValue().replace("-", ""));
            Float adios = (Float) (holas / 10 * ("0.00").replace("-", "").length() - 1);
            valor = valor + adios;
        } else {
            Integer holas = Integer.parseInt(entry.getValue().replace("-", ""));
            valor = valor + holas;
        }
        if (entry.getKey().getValue() >= i && entry.getKey().getValue() <= j) {
            Integer holas = Integer.parseInt(entry.getValue().replace("-", ""));
            Float adios = (Float) (holas / 10 * ("0.00").replace("-", "").length() - 1);
            valor1 = valor1 + adios;
        } else {
            Integer holas = Integer.parseInt(entry.getValue().replace("-", ""));
            valor1 = valor1 + holas;
        }
    }
}
```

luego de esto se leen los datos, primero se suman los valores value\_usd y fee\_usd(cabe aclarar que esto pasa una vez el usuario de click al boton de ver los cuadrados), luego de esto se procede a graficar cada recuadro mirando cuántas transacciones se dieron en esa zona y asignándole un color de acuerdo al rango que se seleccionó, las transacciones que ocurrieron en una frontera(lado,esquiza común) pertenecen al primer cuadro que el código considera como suyo(posiblemente el menor).

luego de esto grafica las leyendas y todo lo correspondiente del ejercicio, para luego llamar a otra clase que llama otra ventana para poder ver los datos de cada cuadro.

```

// grafico del total de value_usd
g.setColor(Color.BLACK);
for (Entry<Integer, String> entry : sumtotalvalue.entrySet()) {
    g.drawString(entry.getValue().split(",")[2], (55 + Integer.parseInt(entry.getValue().split(",")[0])),
        (75 + Integer.parseInt(entry.getValue().split(",")[1])));
}

// grafico del total de fee_usd
Integer count = 1;
g.setColor(Color.BLACK);
for (Entry<Integer, String> entry : sumtotalfee.entrySet()) {
    g.drawString(entry.getValue().split(",")[2], (55 + Integer.parseInt(entry.getValue().split(",")[0])),
        (85 + Integer.parseInt(entry.getValue().split(",")[1])));
    g.drawString(count.toString(), (55 + Integer.parseInt(entry.getValue().split(",")[0])),
        (95 + Integer.parseInt(entry.getValue().split(",")[1])));
    count++;
}

g.setColor(Color.green);
for (int i = 50; i < 70; i += salto)
    g.drawLine(50, i, x, i);
for (int i = 50; i < x; i += salto)
    g.drawLine(i, 50, i, y);
g.drawLine(50, 65, x, 65);
g.drawLine(65, 50, 65, y);
Punto2 datos = new Punto2("hola", transaccionesdatos);
datos.setVisible(true);

```

```

ActionListener oyenteDeAccion1 = new ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent e) {
        informacion.setText("");

        Map<Integer, ArrayList<String>> ordenmaius = new HashMap<Integer, ArrayList<String>>();
        TreeMap<Float, ArrayList<ArrayList<String>>> ordenadov = new TreeMap<>();
        ArrayList<String> jua = new ArrayList<String>();
        ArrayList<ArrayList<String>> jue = new ArrayList<ArrayList<String>>();
        Float parsed;

        for (Entry<Integer, ArrayList<String>> entry : transacciones.entrySet()) {
            if (entry.getKey() == Integer.parseInt(entry.getValue().get(0))) {
                for (int j = 5; j < entry.getValue().size() - 1; j = j + 8) {
                    jue = new ArrayList<ArrayList<String>>();
                    jua = new ArrayList<String>();
                    if (entry.getValue().get(j).startsWith("0")) {
                        Integer holas = Integer.parseInt(entry.getValue().get(j).replace(".", ""));
                        parsed = (float) (holas / (10 * ("0.00").replace(".", "").length() - 1));
                    } else {
                        parsed = (float) Integer.parseInt(entry.getValue().get(j).replace(".", ""));
                    }
                    jua.add(entry.getValue().get(j - 5));
                    jue.add(entry.getValue().get(j - 4));
                    jue.add(entry.getValue().get(j - 3));
                    jue.add(entry.getValue().get(j - 2));
                    jue.add(entry.getValue().get(j - 1));
                    jue.add(entry.getValue().get(j));
                    jue.add(entry.getValue().get(j + 1));
                    jue.add(entry.getValue().get(j + 2));
                    if (ordenadov.get(parsed) == null) {
                        jue.add(jua);
                        ordenadov.put(parsed, jue);
                    } else {
                        jue = ordenadov.get(parsed);
                        jue.add(jua);
                        ordenadov.put(parsed, jue);
                    }
                }
            }
        }
    }
}

```

y ya en esta ventana de visualización de los bloques se muestran los datos de acuerdo a cómo el usuario quiera verlos ordenados(fechas de menor a mayor, igual que los números).

## Tercer Punto

```

} catch (SQLException err) {
    JOptionPane.showMessageDialog(null, "No hay conexión con la base de datos.");
    return;
}

try {
    resultado = sentencia
        .executeQuery("SELECT id, time FROM BLOQUE WHERE miner = '" + maincrall.getText() + "'");
    while (resultado.next()) {
        SimpleDateFormat format = new SimpleDateFormat("HH:mm");
        Date parsed = null;
        Date fechaFinal = null;
        Date fechaInicio = null;
        String fecha = resultado.getString(2).toString();
        try {
            parsed = format.parse(fecha.split(" ")[1]);
            fechaInicio = format.parse(maincrall.getText());
            fechaFinal = format.parse(maincrall.getText());
        } catch (ParseException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        if (fechaInicio.compareTo(parsed) == 0) {
            // System.out.println("fecha igual");
            minerall.add(resultado.getString(1));
        } else if (parsed.compareTo(fechaFinal) < 0 && parsed.compareTo(fechaInicio) > 0) {
            // System.out.println("fecha no igual");
            minerall.add(resultado.getString(1));
        } else if (fechaFinal.compareTo(parsed) == 0) {
            // System.out.println("fecha igual");
            minerall.add(resultado.getString(1));
        }
        System.out.println(fecha);
        System.out.println(resultado.getString(1));
        // System.out.println(resultado.getString(6).toString());
        // java.sql.Date sql = new java.sql.Date(parsed.getTime());
        // System.out.println(sql);

        // Intento de traer el url de solo la ciudad ingresada por el usuario en un
        // String
    }
}

```

---

bajo la recomendación dada por el profesor, primero se valida si existen los 2 miners, si no existe alguno de ellos(o ambos) no se grafica.

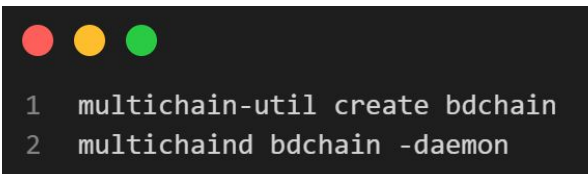
luego de esta validación se obtiene de la base de datos todos los bloques del minero 1 y 2, para luego ser filtrados por el rango de fecha que suministra el usuario, luego se consultan todas las transacciones que tengan asociado al bloque del minero, y se obtiene el value\_usd y se guarda en una lista.

```
public void paint(Graphics g) {  
    int x = 800;  
    int y = 600;  
    // iterando mapa  
    g.setColor(Color.black);  
    g.drawLine(50, 50, 550, 550);  
    g.drawLine(50, 550, 750, 550);  
    System.out.println(ventas);  
    Set<Float> quipu = new HashSet<Float>(ventas);  
    for (Float key : quipu) {  
        int a = Math.round(key / 3500);  
        if (Collections.frequency(ventas, key) > 1) {  
            g.setColor(Color.red);  
            g.drawOval(750 - a-100, 550 - a-100, 25, 25);  
            g.drawString("(" + key + ", " + Collections.frequency(ventas, key)+ ")", 750 - a-100, 450 - a-100);  
        } else {  
            g.setColor(Color.blue);  
            g.drawOval(750 - a-100, 550 - a-100, 25, 25);  
            g.drawString("(" + key + ")", 750 - a-100, 450 - a-100);  
        }  
    }  
}
```

luego de esto se llama otra clase para graficar, donde dividimos los precios por 3500 y los redondeamos para así poder graficar mejor(pues los valores SON MUY GRANDES!), y contamos también los repetidos dándoles un color especial(azulito)

## Cuarto Punto

Para la creación de nuestra Blockchain, se ejecutaron los siguientes prompts o comandos dentro del cmd de Windows:



```
1 multichain-util create bdchain  
2 multichaind bdchain -daemon
```

Esto dentro de una consola cmd, que servirá como daemon para esta blockchain.

Luego, tomamos información del daemon como el puerto, y abrimos una segunda consola cmd.

Aquí antes de crear el asset, preguntamos por la dirección del nodo en el que estamos, pues lo necesitaremos para darnos los permisos necesarios; luego de tener la dirección del

---

nodo, podemos darle al nodo los permisos que queramos, y por último creamos el asset bdcoin y lo depositamos en la misma dirección:

```
1 multichain-cli bdchain getaddresses
2 multichain-cli bdchain grant [Address] connect,send,receive
3 multichain-cli bdchain issue [Address] bdcoin 1000000 0.01
```

Se crearon un millón de unidades, y este asset es divisible por 100, osea, se trabajará con dos decimales.

## Quinto Punto

Para el Login (Inicio de sesión) lo que hicimos fue hacer una consulta en Oracle, para captar la contraseña, y dirección en la blockchain del usuario que ingresaron por la interfaz gráfica, si las contraseñas coinciden, se hace un llamado a un método estático en la interfaz padre, donde se almacenarán las variables globales, nombreUsuario y la dirección

```
1 public static void setCurrent(String nom, String dir) {
2     userName = nom;
3     hash = dir;
4 }
```

Esto nos permitirá validar si hay una sesión activa, desde otras interfaces más rápido, y con todos los datos que podríamos necesitar.

Y para el Sign-In (Registro) lo que hacemos después de ser accionado el botón Registrarse, se valida en la base de datos en Oracle, si el nombre de usuario no exista, luego de

```
1 addressResult = commandManager.invoke(CommandElt.GETNEWADDRESS);
2 newAddress = (String) addressResult;
```

validado este aspecto, se procede a generar una nueva Address en la Blockchain, y

cuando sea generada, el nuevo usuario será guardado en la tabla usuario junto con el nombre de usuario y su contraseña. luego del guardado, se procede a darle los permisos necesarios al nuevo usuario creado, en específico permiso de conexión, envío y recepción.

---

## Sexto Punto

Para el menú del sexto punto, en la parte de visualizar el saldo de la dirección del usuario con el que ingresamos al sistema, hacemos una consulta a la Blockchain para obtener el balance de bitcoin de esa dirección, y lo mostramos en una ventana emergente.

```
1 result = (List<BalanceAssetGeneral>) commandManager.invoke(CommandElt.GETADDRESSBALANCES, current.hash);
```

Para el apartado Pagar, se le pide al usuario que ingrese el usuario al que desea enviar bdcoins, luego de ser accionado el botón Pagar, se valida primero que la cantidad sea

```
1 Map<String, Double> basket = new HashMap<>();
2 basket.put("bdcoin", coins);
3 result = commandManager.invoke(CommandElt.SENDFROM, current.hash, ToHash, basket);
```

correcta (no contenga letras, ni caracteres y si permite decimales), y

segundo, validamos que el usuario destino si exista en la Blockchain, luego de la validación, se procede a hacer el envío del Asset a la cuenta destino, y si los fondos eran suficientes, se mostrará un mensaje de pago exitoso, sino, se mostrará al usuario el mensaje "Insufficient funds".

Y por último para cerrar sesión, lo que hacemos es llamar un método estático global que resetee los campos nombre Usuario y la dirección , y luego se cierra la ventana y se redirige al usuario a la ventana del punto 2.

```
1 MenuPrincipal.setCurrent("", "");
2 Punto2 puntos = new Punto2("Segundo punto");
3 puntos.setVisible(true);
```