Enfoque adoptado a la resolución de la Tarea 4

Para esta práctica tendré que hacer uso de **JDBC** (**J**ava **D**ata**B**ase **C**onnectivity) que no es más que una API que permite la ejecución de operaciones sobre bases de datos desde Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando SQL. De modo que he instalado **XAMPP**, ya que esta herramienta me ofrece sin problemas un ecosistema **Apache** y **MySQL.**

Antes de ponernos a programar me gustaría entrar un poco en contexto ya que me ayudará para los acciones futuras, para conectar Java con una bbdd necesitamos conocer las siguientes cosas.

Lo primero que necesitamos para conectarnos con una base de datos es un **Driver** (o **Connector**). Ese Driver de alguna forma sabe cómo hablar con la base de datos. NetBeans tiene los drivers de **MySQL** por lo tanto este paso no es laborioso, en caso de no usar NetBeans tendríamos que conseguir que java tenga el jar del driver accesible, que sepa dónde encontrarlo.

```
19  <body>
20      <%
21          // Cargamos el driver de conexión para la base de datos MySQL.
22          Class.forName("com.mysql.jdbc.Driver");
23
24          // Credenciales para la conexión (Connection)
25          String cadcon = "jdbc:mysql://localhost:3306/baloncesto";
26          String usuario = "root";
27          String password = "";
28
29          // Una vez que nos hemos asegurado que java tiene el Driver cargado,
30          // simplemente pediremos conexión con la base de datos a la clase DriverManager
31          Connection conexion = DriverManager.getConnection(cadcon, usuario, password); // Conexion
32          Statement s = conexion.createStatement(); // Statement
33          ResultSet resultConsul = s.executeQuery("SELECT * FROM tabla"); // Resultados de las consultas
34      %>
35  </body>
```

DRIVER

Realizamos conexión con DriverManager

Statement lanza las consultas y estas se guardan en ResultSet, para hacer varias consultas debe haber varios Statements y ResulSets

Connection: Sirve para pedir conexión a la base de datos a la clase DriverManager.

Statement: Sirve para procesar una sentencia SQL estática y obtener los resultados producidos por ella. Solo puede haber un ResultSet abierto para cada objeto Statement en un momento dado. Todos los métodos statement que procesan una sentencia SQL cierran implícitamente el ResultSet actual de una sentencia si existe uno abierto.

ResultSet: Las consultas se hacen con **executeQuery()** y nos devolverán un ResultSet.
El ResultSet de alguna forma representa una conexión hacia los datos. En el ResultSet NO están todavía los datos. Según se los vayamos pidiendo, los irá trayendo de la base de datos. Esto quiere decir que si una consulta devuelve muchos resultados, no se nos va a llenar la memoria por el hecho de hacer la consulta.
Para traer el primer resultado, debemos llamar el método **next()** del ResultSet. Para el siguiente otro next() y así sucesivamente hasta que next() devuelva false, indicando que ya no quedaban datos.

Primera etapa:

Para empezar con esta actividad antes debo organizar cómo va a ser realizada, en primera instancia había pensado en hacerla completamente dinámica, desde la creación de la BBDD/tablas (leyendo los XML y obteniendo los valores para averiguar cuál es su tipo de atributo) a la inserción de los datos en cada una de ellas, aunque también he pensado que como tenemos la certeza de que son 16 bloques (y será una tabla por bloque), podría crear las tablas manualmente y luego en nuestro programa **generar los INSERT** donde fuera necesario leyéndolo de los XML.
Como iba diciendo, cada bloque sería una tabla en la cual volcaremos todos los datos (DAT), y como si para un cliente se tratara, debo de evaluar qué y cómo se va a hacer (Si dinámicamente o de forma manual creando las tablas).

Ventajas de hacerlo dinámico

1. Me ahorro tener que crear todas las tablas manualmente.

2. Crearía las tablas (si no existieran) y si en un futuro recibiera más bloques sería capaz de crearse una tabla extra con sus columnas pertinentes y su respectiva inserción.

Desventajas de hacerlo dinámico

1. El problema que le veo es que debo evaluar cada dato de columna previamente para indicarle a la sentencia CREATE TABLE qué tipo de columna es y pueda ser insertado sin problemas.

   Pequeña solución: Esto podría no ser un problema si hiciera que todos los atributos fueran varchar(255) y luego casteados dependiendo de la circunstancia (Descartado).

Ventajas de hacerlo manual

1. Creo yo mismo las tablas con LDD o modo visual vía PHPMyAdmin/SQL Developer/HeidiSQL.

2. El tipo de dato que corresponde al atributo de la tabla lo seleccionaremos nosotros y por lo tanto el programa no sería tan extenso.

3. Al evaluar cada tabla y sus tipos los INSERT se harían con menor margen de error.

Desventajas de hacerlo manual

1. Si en un futuro se quisiera añadir un nuevo bloque tendría que crearse previamente la tabla, por lo tanto quitamos el factor de que sea dinámico pero ganamos en seguridad y estabilidad.

Bitácora 1: Programar una herramienta que me genere las sentencias INSERT INTO de forma dinámica.

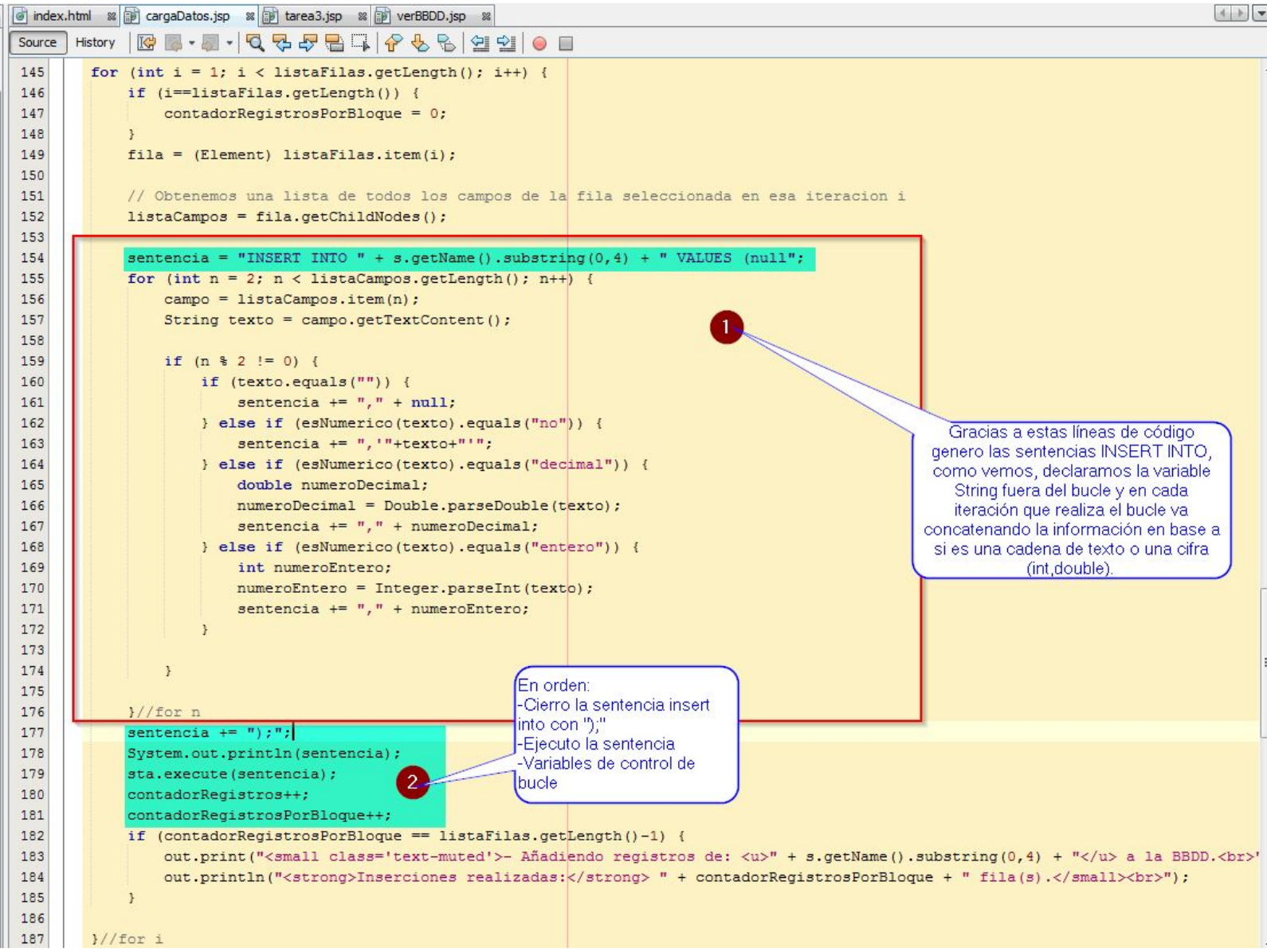
Una vez con nuestras tablas creadas (las cuales han sido creadas mediante LDD), debemos indicar de donde quiero leer los datos que van a procesarse para subirse a la bbdd, en mi caso lo leeré con los archivos XML generados previamente, por lo tanto he tenido que usar **DOM** para moverme por los nodos de la estructura de mis archivos XML, una vez podemos obtener todos los datos, nos queda programar la herramienta que nos va a ayudar a **generar las sentencias INSERT INTO**, me he ayudado de una función para averiguar si en una cadena de texto:

- Averiguar si es un número decimal
- Averiguar si es un número entero
- Averiguar si es una cadena de texto corriente

Dependiendo de si es una cifra o una cadena de texto debemos controlarlo ya que la sentencia **INSERT INTO** y nuestras tablas distinguen y tienen columnas varchar, int, DECIMAL… Aquí muestro la creación de la función y herramienta:

```
<%!
public static String esNumerico(String cadena) {

        String resultado;

        try {
            Integer.parseInt(cadena);
            resultado = "entero";
        } catch (NumberFormatException excepcion) {
            try {
                Double.parseDouble(cadena);
                resultado = "decimal";
            } catch (NumberFormatException excepcionDos) {
                resultado = "no";
            }

            return resultado;
        }
        return resultado;
    }

%>
```

FUNCION



Cabe mencionar que previamente se ejecuta un **TRUNCATE TABLE** a las tablas que detecta que van a ser introducidos los datos, para evitar redundancia y repeticiones ya que estamos trabajando sobre bloques ya creados, esto se realiza antes de iniciar la ejecución de las sentencias INSERT INTO generadas en cada iteración.

Bitácora 2: Visualización de las tablas vía consultas SELECT

Como dice nuestro título, además de cargar los datos debemos crear una pequeña interfaz que nos permita visualizar nuestras tablas, para ello me he servido de parte del código que utilicé en la tarea 3 ya que también consistía en visualizar datos por pantalla, aunque no de una bbdd si no de XML, así que además de adaptarlo a mis necesidades para esta práctica, lo he mejorado.

Me he servido de algunas sentencias curiosas que nunca había utilizado:

- **SHOW TABLES:** Me muestra los nombres de las tablas creadas en mi BBDD, esto lo he utilizado para obtener los nombres y poder integrarlo a la hora de elegir qué tabla queremos ver mediante un: **<select><option>**nombre_tabla**</option></select>**

- **DESCRIBE** nombre_tabla: Usado para obtener las columnas de la tabla que necesite, esto lo he usado para imprimir los nombres de columna de mi bbdd.

Ahora muestro como funciona nuestra herramienta para visualizar los datos:

```
    /**
     * Añado QUERY A UNA LISTA PARA NO VOLVER A ACCEDER AL RESULTSET
     */
    while (listaColumnas.next()) {
        nombreColumnas.add(listaColumnas.getString("Field"));
    }

    int n = 0;
    out.print("<tr>");

    /**
     * IMPRIMO NOMBRE DE LAS COLUMNAS CON TH
     */
    for (int i = 0; i < nombreColumnas.size(); i++) {
        out.print("<th>" + nombreColumnas.get(i) + "</th>");
    }
    out.println("</tr>");
    long inicio = System.currentTimeMillis();
    resultConsul = sta.executeQuery("SELECT * FROM " + opcion + ";");
    String texto = "";

    while (resultConsul.next()) {
        out.print("<tr>");
        for (int i = 0; i < nombreColumnas.size(); i++) {
            texto = resultConsul.getString(nombreColumnas.get(i));
            out.print("<td>" + texto + "</td>");
        }
        out.println("</tr>");
    }
    out.print("</table>");
    long finaliza = System.currentTimeMillis() - inicio;
    out.println("<h4 class='text-center'>Ha tardado " + finaliza + " milisegundos.</h4>");
}//if
```

1 — Imprimimos en negrita las títulos de columnas de forma dinámica.

2 — Imprimimos los datos fila por fila obteniendo nuestros datos de la bbdd con un .getString (nombre_columna) sobre el ResultSet

Bitácora 3: Diseño de la interfaz

Para la realización de la interfaz me he servido de un framework, esto para mi es algo bastante nuevo y aunque en anterioridad había oído hablar de ellos no había sido hasta ahora cuando estoy aprendiendo un poco más en profundidad cómo funcionan, en concreto he utilizado BootStrap 3.

También he mirado algunas herramientas bastante útiles y potentes como **GRID ó FlexBox,** he intentado durante la elaboración de esta práctica profundizar un poco más en estas dos tecnologías y tener una pequeña toma de contacto con ellas.

Diseño index

Diseño cargar datos

Al finalizar la carga de datos me indica además **cuantas** filas han sido añadidas, por tabla y en total, y a parte el tiempo que ha tardado este proceso en realizarse:

Carga de datos

¡Atención! Con motivo de hacer una inserción segura, al realizar esta acción se realizará un **borrado masivo** de todos los datos almacenados en las tablas, y **posteriormente se volverán a introducir** de nuevo. Hazlo si estás seguro pues esta acción puede llevar unos minutos.

Se obtendrán los datos de:
C:\Users\Administrador\Documents\NetBeansProjects\Tarea4\datos

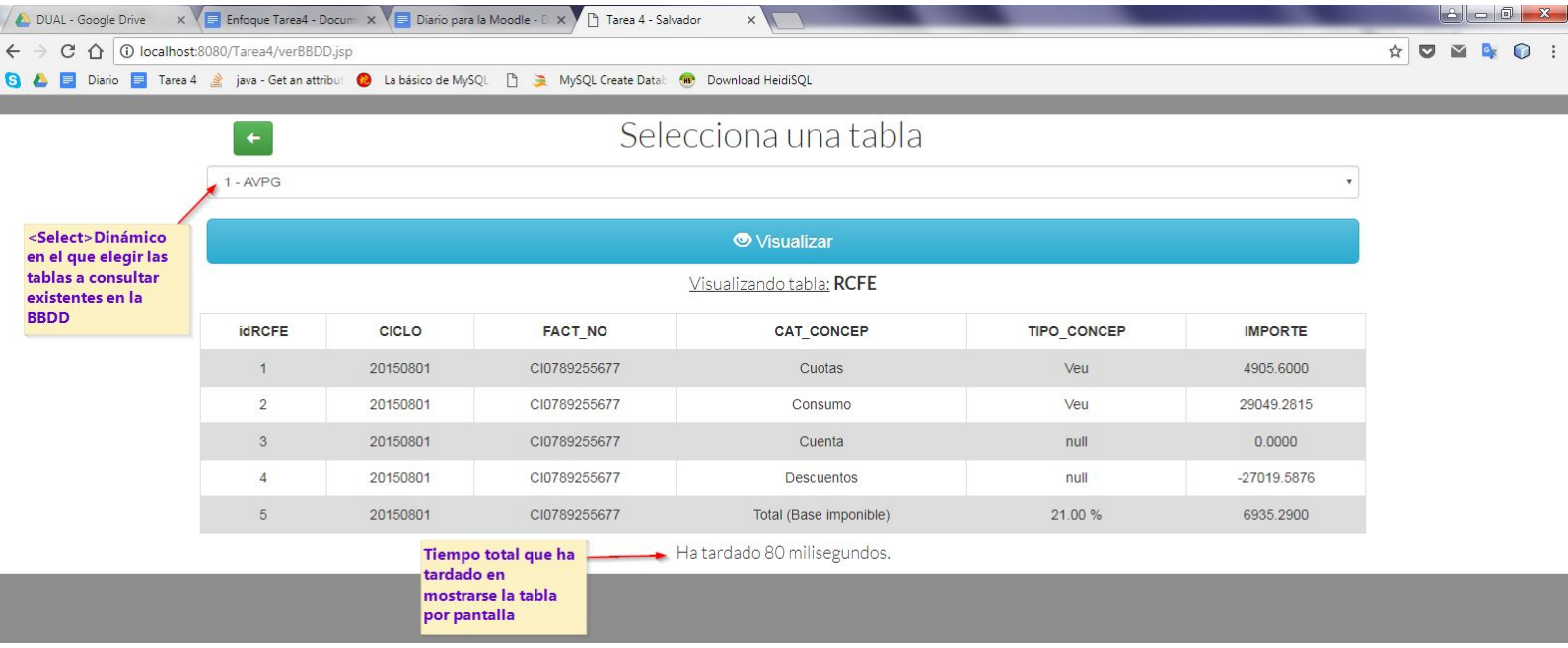
Cargar datos

- Añadiendo registros de: AVPG a la BBDD.
Inserciones realizadas: 1 fila(s).
- Añadiendo registros de: DCFC a la BBDD.
Inserciones realizadas: 1 fila(s).
- Añadiendo registros de: DDNC a la BBDD.
Inserciones realizadas: 15 fila(s).
- Añadiendo registros de: ECTA a la BBDD.
Inserciones realizadas: 1 fila(s).
- Añadiendo registros de: FACT a la BBDD.
Inserciones realizadas: 1 fila(s).
- Añadiendo registros de: IDCL a la BBDD.
Inserciones realizadas: 1 fila(s).
- Añadiendo registros de: RCFE a la BBDD.
Inserciones realizadas: 5 fila(s).
- Añadiendo registros de: RCLL a la BBDD.
Inserciones realizadas: 9 fila(s).
- Añadiendo registros de: RCTL a la BBDD.
Inserciones realizadas: 1 fila(s).
- Añadiendo registros de: RDES a la BBDD.
Inserciones realizadas: 1 fila(s).
- Añadiendo registros de: RIMP a la BBDD.
Inserciones realizadas: 1 fila(s).

Total inserciones realizadas: 37.

Ha tardado 7911 milisegundos.

Diseño ver BBDD



Tarea 4 finalizada