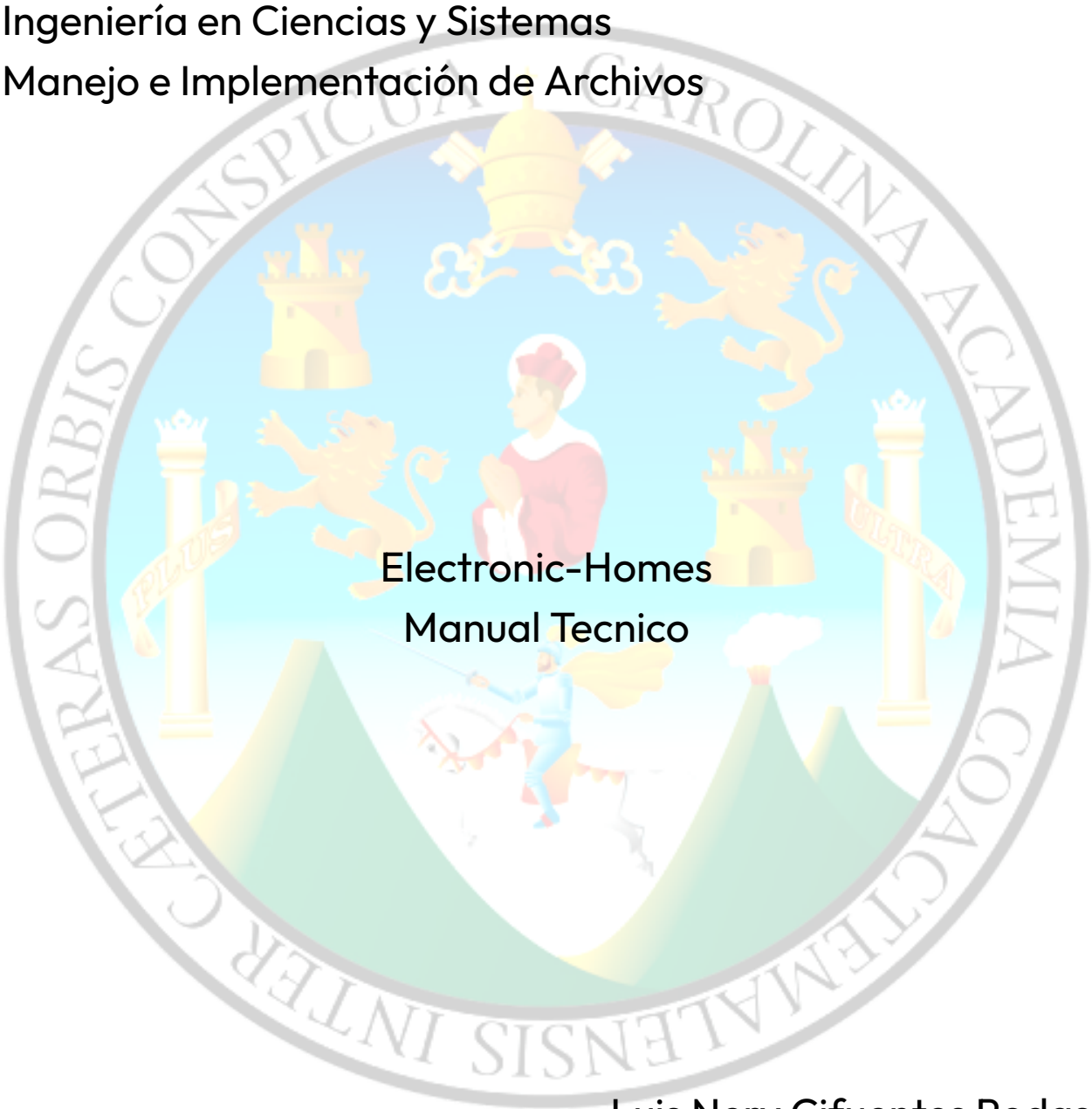
Universidad San Carlos de Guatemala
Centro Universitario de Occidente
División de Ciencias de la Ingeniería
Ingeniería en Ciencias y Sistemas
Manejo e Implementación de Archivos



Electronic-Homes
Manual Tecnico

Luis Nery Cifuentes Rodas
Reg. Academico : 202030482
30/03/2023

Métodos utilizados para las consultas SQL

Para estas consultas se utilizó siempre una variable string que contiene la instrucción SQL para poder realizar las diferentes entidades de nuestro sistema.

1. Cliente

```java
public boolean actualizarCliente(String nombre, String apellido, int nit) { //Metodo que recibe los datos a actualizar del
    boolean correctUpdate = false; //Bandera que marca si se realizo con exito
    String consulta = "UPDATE ControlTienda.Cliente SET nombreCliente = ?, apellidoCliente = ? WHERE nit = ?"; //Se prepar
    try(PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)){ //Se hace la llamada
        //Se sustituyen los valores
        preSt.setString(1, nombre);
        preSt.setString(2, apellido);
        preSt.setInt(3, nit);
        if (preSt.executeUpdate() > 0) { //Vaida que haya sido exitosa
            correctUpdate = true;
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
        //Imprime un mensaje en caso de que no sea asi
    }
    return correctUpdate; //Retorna el valor
}
```

```java
public boolean insertarCliente() {
    boolean correctInsert = false; //Bandera que indica que fue agregado
    String consulta = "INSERT INTO ControlTienda.Cliente(nit, apellidoCliente, nombreCliente) VALUES (?, ?, ?)"; //
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la llamada
        //Se sustituyen los datos
        preSt.setInt(1, this.nitCliente);
        preSt.setString(2, this.apellidoCliente);
        preSt.setString(3, this.nombreCliente);
        if (preSt.executeUpdate() > 0) {
            correctInsert = true; //Si se inserto correctamente se setea a true
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
        //Muestra si hay un error
    }
    return correctInsert; //Devuelve si la insercion fue correcta
}
```

```java
public boolean buscarCliente(int nit) {
    boolean clienteEncontrado = false; //Bandera que indica si el cliente fue encontrado
    String consulta = "SELECT apellidoCliente, nombreCliente FROM ControlTienda.Cliente WHERE nit = ?";
    try(PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la llam
        //Se setean los datos
        preSt.setInt(1, nit);
        ResultSet resultado = preSt.executeQuery(); //Se ejecuta la Query
        if (resultado.next()) { //Valida que se haya encontrado el cliente
            //Se asignan los datos
            this.nitCliente = nit;
            this.apellidoCliente = resultado.getString(1);
            this.nombreCliente = resultado.getString(2);
            clienteEncontrado = true; //Se devulve verdadero
        }
        resultado.close(); //Se cierra para mejor optimizacion
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
        //Muestra si es que hay un error
    }
    return clienteEncontrado; //Devuelve si la operacion fue exitosa
}
```

2. Detalle_Venta

```java
public boolean agregarDtlVenta() {
    boolean detalleAgregado = false; //Bandera que indica si un detalle de vetna fue agregado correctamente
    //Se formula la Query para que inserte datos en la tabla DetalleVenta
    String consulta = "INSERT INTO ControlTienda.DetalleVenta (idVenta, codigoPdt, inventario, descripcion, cantidad, subTotal) VALUES
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) {
        preSt.setInt(1, this.idVenta);
        preSt.setString(2, this.codigoProducto);
        preSt.setInt(3, this.inventario);
        preSt.setString(4, this.descripcionProducto);
        preSt.setInt(5, this.cantidadProducto);
        preSt.setDouble(6, this.subTotal);
        if (preSt.executeUpdate() > 0) {
            detalleAgregado = true;
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, e.getMessage()); //Muestra el mensaje de error
    }
    return detalleAgregado;
}
```

3. Empleado

```java
public boolean comprobarInformacion(long idUsuario, String contraseniap) { //Comprueba si los datos ingresador p
    boolean banderaCorrecta = false; //Bandera que marca si el usuario existe
    String consulta = "SELECT * FROM ControlPersonal.Empleado WHERE idUsuario = ? AND contrasenia = ?"; //Query
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //SE utiliza un try-whith-r
        preSt.setLong(1, idUsuario); //Se sustituye el primer ?
        preSt.setString(2, contraseniap); //Se sustituye el segundo ?
        ResultSet resultado = preSt.executeQuery(); //Se crea un conjunto de resultados
        if (resultado.next()) { //Valida si esta vacio es decir si se encontro el usuario correspondiente
            banderaCorrecta = true; //Se marca la bandero como verdader
            this.idUsuario = idUsuario; //Se llenan los datos del empleado
            this.contrasenia = contraseniap;
            this.sucursal = resultado.getInt("sucursal");
            this.nombreEmpleado = resultado.getString("nombreEmpleado");
            this.rol = resultado.getInt("rol");
            this.salario = resultado.getDouble("salario");
        }
        resultado.close(); //Se cierra el result-set para proteger la memoria
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage());
        //Muestra el error que ocurre del porque no se pudo ejecutar la instrccuion
    }
    return banderaCorrecta; //Se retorna el valor de la bandera
}
```

```java
public boolean agregarEmpleado() { //Metodo que se encarga de insertar un empleado
    boolean banderaAniadido = false; //Bandera que indica que fue aniadido
    String consulta = "INSERT INTO ControlPersonal.Empleado VALUES (?, ?, ?, ?, ?, ?)"; //Orden de consulta
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la consulta
        preSt.setLong(1, this.idUsuario); //Se sustituyen valores
        preSt.setString(2, this.contrasenia);
        preSt.setString(3, this.nombreEmpleado);
        preSt.setInt(4, this.sucursal);
        preSt.setInt(5, this.rol);
        preSt.setDouble(6, this.salario);
        preSt.executeUpdate(); //Se ejecuta la consulta
        banderaAniadido = true; //Tira afirmativo si fue aniadido
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra el error en pantalla
    }
    return banderaAniadido; //Retorna el valor booleano
}
```

4. Producto

```java
public boolean buscarProducto(String codigoProducto, int inventario) { //Metodo que busca el producto
    boolean productoEncontrado = false; //Bandera que indica si fue encontrado
    //Se prepara la consulta
    String consulta = "SELECT descripcion, cantidad, precioUnitario FROM ControlTienda.Producto WHERE codigoPdt = ? AND inventario = ?"
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la llamada
        //Se sustituyen los datos
        preSt.setString(1, codigoProducto);
        preSt.setInt(2, inventario);
        ResultSet resultado = preSt.executeQuery(); //Se hace la consulta
        if (resultado.next()) { //Si existe sustituye los datos y afirma la operacion con exito
            this.codigoProducto = codigoProducto;
            this.inventario = inventario;
            this.descripcionProducto = resultado.getString(1);
            this.cantidad = resultado.getInt(2);
            this.precioUnitario = resultado.getDouble(3);
            productoEncontrado = true;
        }
        resultado.close(); //se cierra el resultado
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Se imprime el error
    }
    return productoEncontrado; //Se retorna el valor
}
```

```java
public boolean insertarProducto () {
    boolean correctInsert = false; //Bandera que indica que el producto fue agregado
    String consulta = "INSERT INTO ControlTienda.Producto VALUES (?, ?, ?, ?, ?)"; //Se formula el Query
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la llamada
        //Se sustituyen los datos
        preSt.setString(1, this.codigoProducto);
        preSt.setInt(2, this.inventario);
        preSt.setString(3, this.descripcionProducto);
        preSt.setInt(4, this.cantidad);
        preSt.setDouble(5, this.precioUnitario);
        if (preSt.executeUpdate() > 0) { //Se ejecuta el Query
            correctInsert = true; //Si setea a true si fue realizada con exito
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un error por si lo hubiera
    }
    return correctInsert; //Devuelve si la operacion se realizo con exito
}
```

```java
public boolean actualizarStock(String codigoProducto, int noInventario, int cantidadActual) {
    boolean stockActualizado = false; //Bandera que indica que la operacion fue realizada con exito
    String consulta = "UPDATE ControlTienda.Producto SET cantidad = ? WHERE codigoPdt = ? AND inventario = ?"; //Se formula la Query
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se preapra la llamadda
        preSt.setInt(1, cantidadActual); //Se sustituyen los datos
        preSt.setString(2, codigoProducto);
        preSt.setInt(3, noInventario);
        if (preSt.executeUpdate() > 0) { //Se actualiza el stock
            stockActualizado = true; //Se cambia el valor a true
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un mensaje de error por si hubiera
    }
    return stockActualizado; //Se retorna el valor de exito de la operacion
}
```

```java
public boolean acutalizatDesc(String nuevaDesc) {
    boolean actualizado = false;
    String consulta = "UPDATE ControlTienda.Producto SET descripcion = ? WHERE codigoPdt = ? AND inventario = ?";
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) {
        preSt.setString(1, nuevaDesc);
        preSt.setString(2, this.codigoProducto);
        preSt.setInt(3, this.inventario);
        if (preSt.executeUpdate() > 0) {
            actualizado = true;
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
    }
    return actualizado;
}
```

```java
public boolean actualizarPrezio(double nuevoPrecio) {
    boolean actualizado = false;
    String consulta = "UPDATE ControlTienda.Producto SET precioUnitario = ? WHERE codigoPdt = ? AND inventario = ?";
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) {
        preSt.setDouble(1, nuevoPrecio);
        preSt.setString(2, this.codigoProducto);
        preSt.setInt(3, this.inventario);
        if (preSt.executeUpdate() > 0) {
            actualizado = true;
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
    }
    return actualizado;
}
```

5. Reportes

```java
public void calcularVentasPorSucursal() {
    this.lb = new Matriz(); // Se crea una nueva instancia de la matriz
    /*
    * Se formula una Query que cuente veces se repite un id de una Sucursal en un registro de venta
    * Indica que lo ordende de mayor a menor por el numero de id de la sucursal y que lo agrupe por el nombre de la misma
    */
    String consulta = "SELECT COUNT(s.sucursal), p.nombreInmueble FROM ControlInmuebles.Instalacion AS p INNER JOIN ControlTienda.Vent
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se realiza la llamada
        ResultSet resultado = preSt.executeQuery(); //Se ejecuta la Query
        while (resultado.next()) { //Recorre el resultset hasta ya no encontrar mas registros
            Lista lista = new Lista(null); //Crea una nueva lista
            lista.agregarNodo(resultado.getInt(1)); //Agrega nodos a la lista
            lista.agregarNodo(resultado.getString(2));
            this.lb.agregarNuevaLista(lista); //Agrega la lista a la matriz
        }
        resultado.close(); //Cierra el resultset
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un error por si lo hubiera
    }
    //return arreglo; //Retorna el arreglo
}
```

```java
public void calcularIngresosPorSucursal() {
    this.lb = new Matriz();
    /*
    * Se formula una Query que cuente veces se repite un id de una Sucursal en un registro de venta
    * Indica que lo ordende de mayor a menor por el numero de id de la sucursal y que lo agrupe por el nombre de la misma
    */
    String consulta = "SELECT SUM(s.total), p.nombreInmueble FROM ControlInmuebles.Instalacion AS p INNER JOIN ControlTienda.Venta AS s
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se realiza la llamada
        ResultSet resultado = preSt.executeQuery(); //Se ejecuta la Query
        while (resultado.next()) { //Recorre el resultset hasta ya no encontrar mas registros
            Lista lista = new Lista(null); //Crea una nueva lista
            lista.agregarNodo(resultado.getDouble(1)); //Agrega nodos a la lista
            lista.agregarNodo(resultado.getString(2));
            this.lb.agregarNuevaLista(lista); //Agrega la lista a la matriz
        }
        resultado.close(); //Cierra el resultset
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un error por si lo hubiera
    }
}
```

```java
public void buscarClientesGanancias() {
    this.lb = new Matriz(); //Se crea una nueva instancia de la matriz
    /*
    * Query formulada para sumar el total de ganancias que genera cada cliente esto sucediendo cada vez que un nit coincida
    * Exigiendo que lo agrupe por el nombre y el nit y que lo ordene de mayor a menor dependiendo del total de su gasto
    */
    String consulta = "SELECT SUM(p.total), p.nitCli, s.nombreCliente, s.apellidoCliente FROM ControlTienda.Venta AS p INNER JOIN Contr
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se realiza la llamada
        ResultSet resultado = preSt.executeQuery(); //Se ejecuta la Query
        while (resultado.next()) { //Recorre el resultset hasta ya no encontrar mas registros
            Lista lista = new Lista(null); //Crea una nueva lista
            lista.agregarNodo(resultado.getDouble(1)); //Agrea nodos a la lista
            lista.agregarNodo(resultado.getInt(2));
            lista.agregarNodo(resultado.getString(3) + " " + resultado.getString(4));
            lb.agregarNuevaLista(lista); //Agrega la lista a la matriz
        }
        resultado.close(); //Se cierra el resultset para optimizar la aplicacion
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un error por si lo hubiera
    }
}
```

```java
public void contarEmpleadosVentas() {
    this.lb = new Matriz(); //Se crea una nueva instancia de la matriz
    /*
    * Se formula la Query para contar cuantas veces vendio un empleado
    * Se ordena que lo agrupe por el id del empleado y su nombre y que lo ordene por la cantidad de manera decendiente
    */
    String consulta = "SELECT COUNT(p.codigoEmpleado), s.idUsuario, s.nombreEmpleado, p.sucursal FROM ControlTienda.Venta AS p INNER JO
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se realiza la llamada
        ResultSet resultado = preSt.executeQuery(); //Se ejecuta la Query
        while (resultado.next()) { //Recorre el resultset hasta ya no encontrar mas registros
            //Llena La matriz
            Lista lista = new Lista(null); //Se crea una nueva lista
            lista.agregarNodo(resultado.getInt(1)); //Se le agrega los datos a esa lista
            lista.agregarNodo(resultado.getLong(2));
            lista.agregarNodo(resultado.getString(3));
            lista.agregarNodo(Conversor.determinarNombreSucursal(resultado.getInt(4)));
            this.lb.agregarNuevaLista(lista); //Se agrega la lista a la matriz
        }
        resultado.close(); //Se cierra el resultset para optimizar la aplicaicon
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un error por si lo hubiera
    }
}
```

```java
public void sumarEmpleadosIngresos() {
    this.lb = new Matriz(); //Se crea una nueva instancia de la matriz
    /*
    * Se formula la Query que recoja los empleados con mas ingresos
    * Se ordenara de manera decendente guidandose por la suma de sus ventas y se agrupara por el id del Empleado y por la sucursal en l
    */
    String consulta = "SELECT p.sucursal, s.idUsuario, s.nombreEmpleado, SUM(p.total) FROM ControlTienda.Venta AS p INNER JOIN ControlT
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la llamada
        ResultSet resultado = preSt.executeQuery(); //Se obtiene un resultset
        while (resultado.next()) { //Valida el tamaño del mismo
            Lista lista = new Lista(null); //Crea una nueva lista
            lista.agregarNodo(Conversor.determinarNombreSucursal(resultado.getInt(1))); //Se agregan nodos a la lista
            lista.agregarNodo(resultado.getLong(2));
            lista.agregarNodo(resultado.getString(3));
            lista.agregarNodo(resultado.getDouble(4));
            this.lb.agregarNuevaLista(lista); //Se agrega la lista a la matriz
        }
        resultado.close(); //Se cierra el result para oprimizar
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Imprime el error en consola
    }
}
```

```java
public void buscarProductosVendidos() {
    this.lb = new Matriz(); //Se crea una nueva instancia de la Matriz
    /*
    * Se selecciona la columna codigoPdt y descripcion de la tabla DetalleVenta
    * Se cre una nueva columna para el resultado que sumara la cantidad de veces que se compro ese producto
    * Se agrupara de acuerdo al codigo del producto y a su descripcion y se ordenara de manera descendente con un limite de 10
    */
    String consulta = "SELECT codigoPdt, descripcion, SUM(cantidad) FROM ControlTienda.DetalleVenta GROUP BY codigoPdt, descripcion O
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la call
        ResultSet resultado = preSt.executeQuery(); //Se obtiene ul resultset
        while (resultado.next()) { //Valida que contiene elementos
            Lista lista = new Lista(null); //Se crea una nueva lista
            lista.agregarNodo(resultado.getString(1)); //Se aniaden nuevos nodos
            lista.agregarNodo(resultado.getString(2));
            lista.agregarNodo(resultado.getInt(3));
            this.lb.agregarNuevaLista(lista); //se agrega la lista a la matriz
        }
        resultado.close(); //Se cierra el resultado para optimizar
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage());//Muestra el error
    }
}
```

```java
public void contarProductosIngresos() {
    this.lb = new Matriz(); //Se crea una nueva instancia de la Matriz
    /*
    * Se selecciona la columna codigoPdt y descripcion de la tabla DetalleVenta
    * Se cre una nueva columna para el resultado que sumara el precio que se pago cuantas veces aparezca el codigo de un producto
    * Se agrupara de acuerdo al codigo del producto y a su descripcion y se ordenara de manera descendente con un limite de 10
    */
    String consulta = "SELECT codigoPdt, descripcion, SUM(subTotal) FROM ControlTienda.DetalleVenta GROUP BY codigoPdt, descripcion OR
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la call
        ResultSet resultado = preSt.executeQuery(); //Se obtiene ul resultset
        while (resultado.next()) { //Valida que contiene elementos
            Lista lista = new Lista(null); //Se crea una nueva lista
            lista.agregarNodo(resultado.getString(1)); //Se aniaden nuevos nodos
            lista.agregarNodo(resultado.getString(2));
            lista.agregarNodo(resultado.getInt(3));
            this.lb.agregarNuevaLista(lista); //se agrega la lista a la matriz
        }
        resultado.close(); //Se cierra el resultado para optimizar
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Se muestra un error en la consola
    }
}
```

```java
public void buscarTop5Productos(int idSucursal) {
    this.lb = new Matriz(); //Crea una nueva instancia para la matriz
    //Se formula una consulta que seleccione los productos por id de inventario y que sume sus cantidades
    String consulta = "SELECT codigoPdt, descripcion, SUM(cantidad) FROM ControlTienda.DetalleVenta WHERE inventario = ? GROUP BY codi
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se realiza la llamada
        preSt.setInt(1, idSucursal);  //Se sustituye los datos
        ResultSet resultado = preSt.executeQuery(); //Se realiza la Query con un resultado
        while (resultado.next()) { //Valida la cantidad de resultados
            Lista lista = new Lista(null); //Se crea una nueva lista y se llena
            lista.agregarNodo(resultado.getString(1));
            lista.agregarNodo(resultado.getString(2));
            lista.agregarNodo(resultado.getInt(3));
            this.lb.agregarNuevaLista(lista); //Se agrega la lista a la matriz
        }
        resultado.close(); //Se cierra el resultset
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Se muestra un error por si lo hubiera
    }
}
```

```java
public void buscarTop5ProductosIng(int idSucursal) {
    this.lb = new Matriz(); //Se crea una nueva instacia de la matriz
    //Se formula una Query que agrupe por el codigo pdt de cierto inventario y que sume sus ganacias ordenandolos de manera descendien
    String consulta = "SELECT codigoPdt, descripcion, SUM(subTotal) FROM ControlTienda.DetalleVenta WHERE inventario = ? GROUP BY codi
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) {
        preSt.setInt(1, idSucursal); //Se sustituye los datos
        ResultSet resultado = preSt.executeQuery(); //Se realiza la Query con un resultado
        while (resultado.next()) { //Valida cuantos resultados dio
            Lista lista = new Lista(null); //Se crea una nueva lista y se llena
            lista.agregarNodo(resultado.getString(1));
            lista.agregarNodo(resultado.getString(2));
            lista.agregarNodo(resultado.getInt(3));
            this.lb.agregarNuevaLista(lista); //Se agrega la lista a la matriz
        }
        resultado.close(); //Se cierra el resultset
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Se muestra un error por si lo hubiera
    }
}
```

```java
public void retornarProductosInventario(int inventario) {
    this.lb = new Matriz(); //Se crea una nueva instancia de la matriz
    //Se formula una Query que recoja los productos por numero de inventario
    String consulta = "SELECT codigoPdt, descripcion, cantidad, precioUnitario FROM ControlTienda.Producto WHERE inventario = ?";
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la call
        preSt.setInt(1, inventario); //Se sustituyen los datos
        ResultSet resultado = preSt.executeQuery(); //Se realiza la Query
        while (resultado.next()) { //Valida los resultados disponibles
            Lista lista = new Lista(null); //Se crea una nueva lista
            if (resultado.getInt(3) == 0 && inventario != 1) {
                resultado.next();
            }
            lista.agregarNodo(resultado.getString(1));
            lista.agregarNodo(resultado.getString(2));
            lista.agregarNodo(resultado.getInt(3));
            lista.agregarNodo(resultado.getDouble(4));
            this.lb.agregarNuevaLista(lista); //Se agrega la lista a la Matriz
        }
        resultado.close(); //Se cierra el resultado
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Se imprime por si hubiera un error
    }
}
```

6. Venta

```java
public boolean insertarCompra() { //Metodo que registra las ventas
    boolean insercionCorrecta = false; //Bandera que Indica que la vetna fue realizada
    //Se formula la Query
    String consulta = "INSERT INTO ControlTienda.Venta (sucursal, fechaVenta, codigoEmpleado, nitcli, cf, total) VALUES (?, ?, ?, ?, ?,
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la call
        preSt.setInt(1, this.sucursal); //Se sutituyen datos
        SimpleDateFormat formato = new SimpleDateFormat("yyyy-MM-dd"); //Se da formato a la fecha
        preSt.setDate(2, java.sql.Date.valueOf(formato.format(this.fechaVenta)));
        preSt.setLong(3, this.codigoEmpleado);
        if (this.nitCliente == null) { //Comprueba si se ingreso un nit
            preSt.setNull(4, Types.BIGINT); //Setea como null
        } else {
            preSt.setInt(4, Integer.parseInt(this.nitCliente)); //Sustituye por el nit
        }
        preSt.setBoolean(5,this.cFinal);
        preSt.setDouble(6, this.total);
        if (preSt.executeUpdate() > 0) { //Se comprueba que la insercion hubiera sido exitosa
            insercionCorrecta = true; //La operacion es exitosa
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un error
    }
    return insercionCorrecta; //Retorna el resultado de la expresion
}
```

```java
public double buscarUltimaCompra(int nitCliente) { //Busca la ultima compra con el nit del cliente
    double totalAnterior = 0; //Retorne el total gastado de esa compra
    //Se formula la Query
    String consulta = "SELECT total FROM ControlTienda.Venta WHERE nitcli=? ORDER BY fechaVenta DESC LIMIT 1";
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se prepara la call
        preSt.setInt(1, nitCliente); //Se sustituyen los valores
        ResultSet resultado = preSt.executeQuery(); //Se instancia un cuerpo de resultados
        if (resultado.next()) { //Verifica si retorno un resultado
            totalAnterior = resultado.getDouble(1); //Se le asigna el valor
        }
        resultado.close(); //Se cierra para evitar problemas de optimizacion
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un error por si lo hay
    }
    return totalAnterior; //Retorna el valor
}
```

```java
public int buscarUltimaInsercion() {
    int idMax = 0; //Id de la ultima venta
    String consulta = "SELECT MAX(idVenta) FROM ControlTienda.Venta"; //Query que selecciona el ID de la ultima venta
    try (PreparedStatement preSt = Conexion.getConexionDB().prepareCall(consulta)) { //Se realiza la consulta
        ResultSet resultado = preSt.executeQuery();  //Se obtiene el id
        if (resultado.next()) { //Valida que la operacion resulto con exito
            idMax = resultado.getInt(1); //Se guarda el ultimo id de la venta
        }
        resultado.close(); //Se cierra el close
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ocurrio un Problema" + e.getMessage()); //Muestra un error por is lo hubiera
    }
    return idMax; //Se retorna el ultimo id de la venta
}
```

Métodos utilizados para darle efecto de botón a los paneles

```
private void jLabelBtnIngresarMouseEntered(java.awt.event.MouseEvent evt) {
    this.jPanelBtnIngresar.setBackground(new Color(152, 155, 158));
}

private void jLabelBtnIngresarMouseExited(java.awt.event.MouseEvent evt) {
    this.jPanelBtnIngresar.setBackground(new Color(94, 96, 98));
}
```

Generalmente los eventos MouseEntered() y MouseExite(), fueron utilizados para lograr efectos de botón en los paneles.

Métodos utilizados para obtener el número de inventario a través del nombre de la sucursal y viceversa

```
public static int determinarSucursal(String nombreSucursal) {
    int inventario = 0;
    String sucursalName = nombreSucursal.trim();
    if (sucursalName.equals("Sucursal Central")) {
        inventario = 2;
    } else if (sucursalName.equals("Sucursal Norte")) {
        inventario = 3;
    } else if (sucursalName.equals("Sucursal Sur")) {
        inventario = 4;
    } else if (sucursalName.equals("Bodega")) {
        inventario = 1;
    }
    return inventario;
}
```

```
public static String determinarNombreSucursalInv(int inventario) {
    String sucursal = "";
    if (inventario == 1) {
        sucursal = "Bodega";
    } else if (inventario == 2) {
        sucursal = "Sucursal Central";
    } else if (inventario == 3) {
        sucursal = "Sucursal Norte";
    } else if (inventario == 4) {
        sucursal = "Sucursal Sur";
    }
    return sucursal;
}
```

Otros métodos de apoyo

```java
public static String determinarNombreSucursal(int id) {
    if (id == 3) {
        return "Sucursal Central";
    } else if (id == 4) {
        return "Sucursal Norte";
    } else if (id == 5) {
        return "Sucursal Sur";
    }
    return null;
}

//Metodo que regresa el pocertaje de descuento

/**
 *
 * @param totalAnterior de la ultima compra
 * @return un double que hace referencia al descuento que tiene derecho el cliente
 */
public static double retornarDescuento(double totalAnterior) {
    if (totalAnterior < 1000) {
        return 0.00;
    } else if (totalAnterior >= 1000 && totalAnterior < 5000 ) {
        return 0.02;
    } else if (totalAnterior >= 5000 && totalAnterior < 10000) {
        return 0.05;
    } else {
        return 0.1;
    }
}
```

```java
public static double calcularTotal(JTable tabla, String desc) {
    int columna = 3;
    double total = 0.00;
    for (int i = 0; i < tabla.getRowCount(); i++) {
        double subtotal = Double.valueOf(tabla.getValueAt(i, columna) + "");
        total += subtotal;
    }
    total = total - (total*Double.valueOf(desc));
    return total;
}

//Metodo que inserta los detalles de venta

/**
 *
 * @param tabla de la cual se obtienen los detalles
 * @param idVenta que indica el id de la venta a la que pertenecera
 * @param inventario al cual hace referencia el detalle
 * @return un booleano qu eindica si se realizo con exito la insercion de detalles de venta
 */
public static boolean insertarDetalles(JTable tabla, int idVenta, int inventario) {
    boolean operacionExitosa = true;
    for (int i = 0; i < tabla.getRowCount(); i++) {
        DetalleVenta dtl = new DetalleVenta(idVenta, String.valueOf(tabla.getValueAt(i, 0)).trim(), inventario,
        String.valueOf(tabla.getValueAt(i, 1)).trim(), Integer.valueOf(String.valueOf(tabla.getValueAt(i, 2)).trim()),
        Double.valueOf(String.valueOf(tabla.getValueAt(i, 3)).trim()));
        dtl.agregarDtlVenta();
        Producto product = new Producto();
        product.buscarProducto(dtl.getCodigoProducto(), dtl.getInventario());
        product.actualizarStock(dtl.getCodigoProducto(), dtl.getInventario(), product.getCantidad() - dtl.getCantidadProducto());
    }
    return operacionExitosa;
```

Métodos utilizados para llenar la tabla y cambiar el modelo

```java
/**
 *
 * @param tabla La tabla a la cual se le cambiara el titulo
 * @param titulos Los titulos se le agregaran a la tabla
 */
public static void cambiarEncabezado(JTable tabla, String[] titulos) {
    JTableHeader header = tabla.getTableHeader();
    header.setBackground(new Color(94, 96, 98));
    header.setForeground(Color.white);
    header.setFont(new Font("SansSerif", Font.PLAIN, 18));
    header.setReorderingAllowed(false);
    header.setResizingAllowed(false);
    DefaultTableModel modelo = new DefaultTableModel();
    for (int i = 0; i < titulos.length; i++) {
        modelo.addColumn(titulos[i]);
    }
    tabla.setRowHeight(30);
    tabla.setBackground(new Color(238, 238, 238));
    tabla.setGridColor(Color.white);
    tabla.setBorder(BorderFactory.createLineBorder(Color.white, 2));
    tabla.setFont(new Font("SansSerif", Font.PLAIN, 16));
    tabla.setEnabled(false);
    tabla.setCellSelectionEnabled(false);
    tabla.setModel(modelo);
}
```

```java
/**
 *
 * @param tabla La tabla a la cual se le insertaran datos
 * @param lista La matriz de la cual se obtendran los datos
 */
public static void llenarTabla(JTable tabla, Lista lista){
    DefaultTableModel modelo = (DefaultTableModel) tabla.getModel();
    Object[] objeto = new Object[lista.hallarTamanio()];
    Nodo temp = lista.getPrimero();
    int indice = 0;
    while (temp != null) {
        objeto[indice] = " " + temp.getContenido();
        indice++;
        temp = temp.getSiguiente();
    }
    modelo.addRow(objeto);
}
```

```java
/**
 *
 * @param tabla La tabla ala cual se le insertaran los datos
 * @param matriz La matriz de la cual se obtendran los datos
 */
public static void llenarTabla(JTable tabla, Matriz matriz) {
    DefaultTableModel modelo = (DefaultTableModel) tabla.getModel();
    int tamanio = matriz.getPrimera().hallarTamanio();
    Lista temp = matriz.getPrimera();
    int orden = 1;
    while (temp != null) {
        Object[] objeto = new Object[tamanio + 1];
        Nodo tempo = temp.getPrimero();
        int indice = 1;
        objeto[0] = "  " + orden;
        while (tempo != null) {
            objeto[indice] = "  " + tempo.getContenido();
            indice++;
            tempo = tempo.getSiguiente();
        }
        orden++;
        modelo.addRow(objeto);
        temp = temp.getSiguiente();
    }
}
```

```java
/**
 *
 * @param tabla La tabla a la cual se le insertaran los datos
 * @param matriz La matriz de la cual se obtendra los valores
 */
public static void llenarTablaInventario(JTable tabla, Matriz matriz) {
    DefaultTableModel modelo = (DefaultTableModel) tabla.getModel();
    int tamanio = matriz.getPrimera().hallarTamanio();
    Lista temp = matriz.getPrimera();
    while (temp != null) {
        Object[] objeto = new Object[tamanio];
        Nodo tempo = temp.getPrimero();
        int indice = 0;
        while (tempo != null) {
            objeto[indice] = "  " + tempo.getContenido();
            indice++;
            tempo = tempo.getSiguiente();
        }
        modelo.addRow(objeto);
        temp = temp.getSiguiente();
    }
}
```

```java
/**
 *
 * @param table La tabla en la cual se verificara un producto
 * @param codigoPdt El codigo del producto a buscar
 * @param cantidadPdt La cantidad del producto en el stock de la tienda
 * @param cantidad La cantidad que se le sumara a la venta
 * @return retorna un int que indica si el producto ingresado ya existia o no
 */
public static int verificarCompra(JTable table, String codigoPdt, int cantidadPdt, int cantidad){
    int existe = 0;
    int filas = table.getRowCount();
    for (int i = 0; i < filas; i++) {
        if (codigoPdt.equals(String.valueOf(table.getValueAt(i, 0)).trim())) {
            int newcantidad = Integer.valueOf(String.valueOf(table.getValueAt(i, 2)).trim()) + cantidad;
            if (newcantidad <= cantidadPdt) {
                table.setValueAt(" " + newcantidad, i, 2);
                existe = 1;
            } else {
                existe = 2;
            }
        }
    }
    return existe;
}
```

Método utilizado para encriptar la contraseña

```java
/**
 *
 * @param contrasenia que que sera hasheada
 * @return un string de la constrasenia hasheada
 */
public static String hashearContrasenia(String contrasenia) {
    String passwrd = "";
    try {
        MessageDigest msg = MessageDigest.getInstance("SHA-256");
        byte[] datosEncriptados = msg.digest(contrasenia.getBytes());
        StringBuilder contraseniaFinal = new StringBuilder();
        for (int i = 0; i < datosEncriptados.length; i++) {
            String temp = Integer.toHexString(0xff & datosEncriptados[i]);
            if (temp.length() == 1) {
                contraseniaFinal.append('0');
            }
            contraseniaFinal.append(temp);
        }
        passwrd = contraseniaFinal.toString();
    } catch (NoSuchAlgorithmException ex) {
        Logger.getLogger(Contrasenia.class.getName()).log(Level.SEVERE, null, ex);
    }
    return passwrd;
}
```

Métodos utilizados para la estructura de datos

1. **Nodo**

```java
/**
 * Atributos
 */
private Nodo siguiente;
private Object contenido;

/*Constructor*/

/**
 *
 * @param siguiente El nodo siguiente
 * @param contenido El contenido que almacenara
 */

public Nodo(Nodo siguiente, Object contenido) {
    this.siguiente = siguiente;
    this.contenido = contenido;

}
```

2. Lista

```java
/**
 *
 * @param siguiente La lista que le seguira
 */

public Lista(Lista siguiente) {
    this.primero = null;
    this.ultimo = null;
    this.siguiente = siguiente;
}

/*Metodo para expandir la matriz*/

/**
 *
 * @param objeto El contenido que tendra el nodo
 */

public void agregarNodo(Object objeto) {
    if (this.primero == null) { //Si el primer nodo es nulo
        this.ultimo = new Nodo(null, objeto); //El ultimo sera un nuevo nodo
        this.primero = this.ultimo; //El primero apuntara al ultimo
    } else {
        Nodo nuevo = new Nodo(null, objeto); //Se crea un nuevo nodo
        this.ultimo.setSiguiente(nuevo); //El siguiente del ultimo sera este nuevo nodo
        this.ultimo = nuevo; //El ultimo ahora sea este nuevo nodo
    }
}
```

3. Matriz

```java
/**
 *
 */

public Matriz() {
    this.primera = null;
    this.ultima = null;
}

/*Metodo para agregar una nueva lista*/

/**
 *
 * @param lista La lista que sera agregada
 */

public void agregarNuevaLista(Lista lista) {
    if (this.primera == null) { //Si la primer lista es nula
        this.ultima = lista; //La ultima lista apuntara a una lista
        this.primera = this.ultima; //La primera apuntata a la ultima
    } else {
        this.ultima.setSiguiente(lista); //La ultima setea su siguiente a otra lista
        this.ultima = lista; //La ultima ahora apunta a la nueva lista
    }
}
```

Métodos utilizados par los JTextField

```
private void jTextFieldCodigoPdtKeyTyped(java.awt.event.KeyEvent evt) {
    if (this.jTextFieldCodigoPdt.getText().length() > 11) { //Valida que no se pase de la cantidad maxima de caracteres
        evt.consume();
        Toolkit.getDefaultToolkit().beep();
    }
}

private void jTextFieldCodigoPdtKeyReleased(java.awt.event.KeyEvent evt) {
    if (evt.getKeyCode() == KeyEvent.VK_BACK_SPACE || evt.getKeyCode() == KeyEvent.VK_DELETE) {
        if (this.jLabelAgregarCompra.isEnabled()) {
            this.jLabelAgregarCompra.setEnabled(false);
            this.jSpinnerCantidad.setEnabled(false);
            this.jSpinnerCantidad.setValue(1);
        }
    }
    if (this.jTextFieldCodigoPdt.getText().isBlank()) { //Valida si el campo esta vacio
        this.jLabelBuscarPdt.setEnabled(false); //Si lo esta lo desactiva
    } else {
        this.jLabelBuscarPdt.setEnabled(true); //Sino lo vuelve a activar
    }
}
```

Generalmente se utilizaron esos dos eventos el KeyTyped() y el KeyReleased(). El primero se usaba para restricciones de tamaño o caracteres y el segundo para saber si realizaba algún cambio en los mismos. Obviamente esto varía para cada JTextField, puesto que cada uno tiene diferentes usos, pero generalmente estos dos eventos se repetían constantemente.

Clases implementadas de JDialog para mostrar mensajes de información o registrar clientes

```
public class ShowMsg extends javax.swing.JDialog {

    /**
     * Creates new form ShowMsg
     * @param parent El frame que bloqueara
     * @param modal El modo que dentra
     * @param mensaje El mensaje que mostrara
     */
    public ShowMsg(java.awt.Frame parent, boolean modal, String mensaje) {
        super(parent, modal);
        initComponents();
        this.jTextPaneMsg.setText(mensaje);
        this.setLocationRelativeTo(null);
        this.setResizable(false);
        this.setVisible(true);
    }
```


```java
public class InsertClientForm extends javax.swing.JDialog {

    /**
     * Atributos
     */
    int nitCliente;

    /**
     *
     * @param frame El frame al cual bloqueara
     * @param modal El modo que realizada
     * @param nitcliente El nit del cliente
     */
    public InsertClientForm(JFrame frame, boolean modal, int nitcliente) {
        super(frame, modal);
        initComponents();
        this.nitCliente = nitcliente;
        this.jLabelNit.setText(this.jLabelNit.getText() + this.nitCliente);
        this.setSize(450, 350);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.setVisible(true);
    }
```