# Acceso a BBDD desde JAVA

# 1. Estableciendo una conexión

Primero, debe establecer una conexión con la fuente de datos que desea usar. Una fuente de datos puede ser un DBMS, un sistema de archivos heredado o alguna otra fuente de datos con un controlador JDBC correspondiente. Normalmente, una aplicación JDBC se conecta a una fuente de datos de destino utilizando una de dos clases:

- `DriverManager` : esta clase implementada por completo conecta una aplicación a un origen de datos, que se especifica mediante una URL de base de datos. Cuando esta clase primero intenta establecer una conexión, carga automáticamente cualquier controlador JDBC 4.0 que se encuentre dentro de la ruta de la clase. Tenga en cuenta que su aplicación debe cargar manualmente cualquier controlador JDBC anterior a la versión 4.0.
- `DataSource` : esta interfaz es preferible a `DriverManager` porque permite que los detalles sobre la fuente de datos subyacente sean transparentes para su aplicación. Las propiedades de un objeto `DataSource` se configuran para que represente una fuente de datos particular.

**Nota** : Los ejemplos de este tutorial utilizan la clase `DriverManager` lugar de la clase `DataSource` porque es más fácil de usar y los ejemplos no requieren las características de la clase `DataSource` .

# 2. Usando la clase DriverManager

Conectarse a su DBMS con la clase `DriverManager` implica llamar al método `DriverManager.getConnection` . El siguiente método ( Conectar() ), establece una conexión de base de datos:

```
public Connection Conectar() {
	Connection conexion = null;
	try {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		String BaseDeDatos = "jdbc:oracle:thin:@" + host + ":" + puerto + ":ORCL";
		conexion = DriverManager.getConnection(BaseDeDatos, usuario, password);
		if (conexion != null)
			System.out.println("Conexión realizada con éxito a MUNDIAL");

	} catch (Exception e) {
		System.out.println("FALLOOOOO EXCEPCION!!!");
		e.printStackTrace();
	}
	return conexion;
}
```

Para acceder a una base de datos desde JAVA, se debe previamente registrar nuestro driver, previamente instalado. Para hacer esto, se puede invocar el método estático `registerDriver()` de la clase `java.sql.DriverManager`. Esta clase proporciona un servicio básico para el manejo de un conjunto de drivers JDBC. También se puede utilizar como alternativa a este paso la mostrada en el ejemplo anterior, llamando al método `forName()` de la clase `java.lang.Class` para cargar el driver JDBC directamente.

```
	Class.forName("oracle.jdbc.driver.OracleDriver");
```

El método `DriverManager.getConnection` establece una conexión de base de datos. Este método requiere una URL de base de datos, que varía según su DBMS. Los siguientes son algunos ejemplos de URL de base de datos:

- Oracle: jdbc:oracle:thin:@localhost:1521:ORCL", donde `thin` es un driver para Oracle, `localhost` es el nombre del servidor que aloja su base de datos, `1521` es el número de puerto, y `ORCL` el nombre del servicio. Mas info aqui.
- MySQL: `jdbc:mysql://localhost:3306/` , donde `localhost` es el nombre del servidor que aloja su base de datos, y `3306` es el número de puerto
- Java DB: `jdbc:derby:` *`testdb`* `;create=true` , donde *`testdb`* es el nombre de la base de datos a la que conectarse, y `create=true` indica al DBMS que cree la base de datos.

# 3. Recuperación y modificación de valores de conjuntos de resultados

El siguiente método, genera los contenidos de la tabla JUGADORES y demuestra el uso de los objetos y cursores `ResultSet` :

```
public static void viewTableWithStatement (Connection con, String dbName)
throws SQLException {

            Statement stmt = null;
            String query =
                "select NOMBRE, DIRECCION,PUESTO_HAB," +
                "FECHA_NAC, EQUIPO_JUGADOR " +
                "from " + dbName + ".JUGADOR";

            try {
                stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery(query);

                while (rs.next()) {

                    String nombre = rs.getString("NOMBRE");
                    String direccion = rs.getString("DIRECCION");
                    String puesto = rs.getString("PUESTO_HAB");

                    Date fecha = rs.getDate("FECHA_NAC");
                    String equipo = rs.getString("EQUIPO_JUGADOR");

                    System.out.println(nombre + "\t" + direccion +
                                       "\t" + puesto + "\t" +
((fecha==null)?"Desconocido":fecha.toString()) +"\t" + equipo);
                }
            } catch (SQLException e ) {
                e.printStackTrace();
            } finally {
                if (stmt != null) { stmt.close();
            }
        }
  }
```

Un objeto `ResultSet` es una tabla de datos que representa un conjunto de resultados de la base de datos, que generalmente se genera al ejecutar una declaración que consulta la base de datos. Por ejemplo, el método `anterior` crea un `ResultSet` , `rs` , cuando ejecuta la

consulta a través del objeto `Statement` ,`stmt` . Ten en cuenta que se puede crear un objeto `ResultSet` través de cualquier objeto que implemente la interfaz `Statement` , incluidos `PreparedStatement` , `CallableStatement` .

Nosotros accedemos a los datos en un objeto `ResultSet` través de un cursor. Ten en cuenta que este cursor no es un cursor de base de datos. Este cursor es un puntero que apunta a una fila de datos en `ResultSet` . Inicialmente, el cursor se posiciona antes de la primera fila. El método `ResultSet.next` mueve el cursor a la siguiente fila. Este método devuelve `false` si el cursor está posicionado después de la última fila. Este método llama repetidamente al método `ResultSet.next` con un bucle while para iterar a través de todos los datos en el `ResultSet` .

## 3.1. Recuperar valores de filas

La interfaz `ResultSet` declara métodos getter (por ejemplo, `getBoolean` y `getLong` ) para recuperar valores de columna de la fila actual. Puede recuperar valores utilizando el número de índice de la columna o el alias o , como se ha realizado en el ejemplo, el nombre de la columna. El índice de columna suele ser más eficiente. Las columnas se numeran a partir de 1. Para una portabilidad máxima, las columnas del conjunto de resultados dentro de cada fila deben leerse en orden de izquierda a derecha, y cada columna debe leerse solo una vez.

## 3.2. Cursores

Como se mencionó anteriormente, se accede a los datos en un objeto `ResultSet a` través de un cursor, que apunta a una fila en el objeto `ResultSet` . Sin embargo, cuando se crea un objeto `ResultSet` primera vez, el cursor se posiciona antes de la primera fila. El método mueve el cursor llamando al método `ResultSet.next` . Hay otros métodos disponibles para mover el cursor:

- `next` : mueve el cursor hacia adelante una fila. Devuelve `true` si el cursor ahora está posicionado en una fila y `false` si el cursor está posicionado después de la última fila.
- `previous` : Mueve el cursor hacia atrás una fila. Devuelve `true` si el cursor ahora está posicionado en una fila y `false` si el cursor está posicionado antes de la primera fila.
- `first` : mueve el cursor a la primera fila en el objeto `ResultSet` . Devuelve `true` si el cursor ahora está posicionado en la primera fila y `false` si el objeto `ResultSet` no contiene ninguna fila.
- `last` : mueve el cursor a la última fila en el objeto `ResultSet` . Devuelve `true` si el cursor ahora está posicionado en la última fila y `false` si el objeto `ResultSet` no contiene ninguna fila.
- `beforeFirst` : coloca el cursor al inicio del objeto `ResultSet` , antes de la primera fila. Si el objeto `ResultSet` no contiene ninguna fila, este método no tiene ningún efecto.
- `afterLast` : coloca el cursor al final del objeto `ResultSet` , después de la última fila. Si el objeto `ResultSet` no contiene ninguna fila, este método no tiene ningún efecto.
- `relative(int rows)` : mueve el cursor relativo a su posición actual.
- `absolute(int row)` : coloca el cursor en la fila especificada por la `row` parámetro.

Ten en cuenta que la sensibilidad predeterminada de un `ResultSet` es `TYPE_FORWARD_ONLY` , lo que significa que no se puede desplazar; no puede llamar a ninguno de estos métodos que mueven el cursor, excepto a `next` , si el `ResultSet` no se puede desplazar. Para poder mover el cursor de un `ResultSet`, se debe crear el Statement de la siguiente manera:

```
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE);
```

Para más información, de cómo podemos actualizar ResultSet, insertar nuevas filas, … visita la siguiente página.

# 4. Usando declaraciones preparadas (PreparedStatement)

Algunas veces es más conveniente usar un objeto PreparedStatement para enviar sentencias SQL a la base de datos. Este tipo especial de declaración se deriva de la clase más general, Statement , que ya hemos visto.

La característica principal de un objeto PreparedStatement es que, a diferencia de un objeto Statement , recibe una instrucción SQL cuando se crea. La ventaja de esto es que en la mayoría de los casos, esta declaración SQL se envía al DBMS de inmediato, donde se compila. Como resultado, el objeto PreparedStatement contiene no solo una declaración SQL, sino una declaración SQL que ha sido precompilada. Esto significa que cuando se ejecuta el PreparedStatement , el DBMS solo puede ejecutar la declaración SQL PreparedStatement sin tener que compilarla primero.

Aunque los objetos PreparedStatement se pueden usar para sentencias de SQL sin parámetros, es probable que los usemos con mayor frecuencia para las sentencias de SQL que toman parámetros. La ventaja de utilizar sentencias de SQL que toman parámetros es que puede usar la misma sentencia y proporcionarle valores diferentes cada vez que la ejecuta.

El siguiente método, visualiza los jugadores de la selecion pasada por parametro:

```
public static void viewTableWithPreparedStatement(Connection con, String
dbName, String team) throws SQLException {

        PreparedStatement pstmt = null;
        String query = "select NOMBRE, DIRECCION,PUESTO_HAB," +
"FECHA_NAC, EQUIPO_JUGADOR " + "from " + dbName
                    + ".JUGADOR " + "where EQUIPO_JUGADOR = ?";

        try {
               pstmt = con.prepareStatement(query);

               pstmt.setString(1, team);

               ResultSet rs = pstmt.executeQuery();

               while (rs.next()) {

                      String nombre = rs.getString("NOMBRE");
                      String direccion = rs.getString("DIRECCION");
                      String puesto = rs.getString("PUESTO_HAB");
                      Date fecha = rs.getDate("FECHA_NAC");
                      String equipo = rs.getString("EQUIPO_JUGADOR");
```

```
                        System.out.println(nombre + "\t" + direccion + "\t"
    + puesto + "\t"
                                        + ((fecha == null) ? "Desconocido" :
    fecha.toString()) + "\t" + equipo);
                }
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                if (pstmt != null) {
                    pstmt.close();
                }
            }
      }
```

Podemos observar como creamos una PreparedStatement que tiene un parámetro de entrada.

```
String query = "select NOMBRE, DIRECCION,PUESTO_HAB," + "FECHA_NAC,EQUIPO_JUGADOR"
                + "from " + dbName + ".JUGADOR " + "where EQUIPO_JUGADOR = ?";
pstmt = con.prepareStatement(query);
```

En la siguiente instrucción, podemos observar como suministramos los parametros a la declaración preparada.

```
pstmt.setString(1, team);
```

Se deben proporcionar los valores marcados con el signo de interrogación antes de poder ejecutar un objeto PreparedStatement. Esto lo debemos hacer llamando a los métodos setters definidos para la clase PreparedStatement.

# 5. LLamadas a procedimientos/funciones en Oracle

El siguiente código es la funcion PRUEBAFUNCION almacenada en Oracle,

```
CREATE OR REPLACE FUNCTION MUNDIAL.PRUEBAFUNCION(TEAM IN VARCHAR, ANYO IN
NUMBER)
RETURN SYS_REFCURSOR
IS
    TYPE C_GOLES IS REF CURSOR return gol%rowtype;
    CURSOR_DEVUELTO C_GOLES;
BEGIN
    OPEN CURSOR_DEVUELTO FOR
        SELECT * FROM GOL
            WHERE   (GOL.EQUIPO_L_GOL=TEAM OR
                    GOL.EQUIPO_V_GOL=TEAM) AND
                    (ANYO IS NULL OR TO_CHAR(GOL.FECHA_GOL,'YYYY')=ANYO)
            ORDER BY GOL.FECHA_GOL ASC;
```

```
    RETURN CURSOR_DEVUELTO;
END;
```

Esta función va a ser llamada por el siguiente fragmento de código, que va a mostrar el conjunto de resultados generados en una tabla:

```
CallableStatement cs = null;
String sql = "{? = call pruebafuncion(?,?) }";
ResultSet rs = null;
try {
	con = (new ConexionOracle(ficheroBBDD)).Conectar();
	cs = con.prepareCall(sql);
	int pos = 0;
	cs.registerOutParameter(++pos, OracleTypes.CURSOR);
	cs.setString(++pos, "ESPAÑA");
	cs.setInt(++pos, 2010);
	cs.execute();
	rs = (ResultSet)cs.getObject(1);
	String Titulo[]={" MINUTO "," JUGADOR_GOL "," EQUIPO_LOCAL "," 
EQUIPO_VISITANTE ", " FECHA "};
	String fila[]=new String[5];

	DefaultTableModel modelo = new DefaultTableModel(null, Titulo);

	while (rs.next()) {
		fila[0] = rs.getObject(1).toString();
		fila[1] = rs.getObject(2).toString();
		fila[2] = rs.getObject(3).toString();
		fila[3] = rs.getObject(4).toString();
		fila[4] = rs.getObject(5).toString();
		modelo.addRow(fila);
	}

table.setModel(modelo);
} catch (Exception e) {
	e.printStackTrace();
}finally{
	SqlTools.close(rs, cs,null, con);
}
```

La interfaz `CallableStatement` extiende de `PreparedStatement` . Se usa para llamar a procedimientos almacenados. Se deben especificar los valores para los parámetros `IN (en el ejemplo, la selección y el año)` tal como se haría con un objeto `PreparedStatement` llamando al método setter apropiado. Sin embargo, si un procedimiento almacenado contiene un parámetro `OUT` , se debe registrar con el método `registerOutParameter` (en nuestro caso un cursor `OracleTypes.CURSOR`).
Por último se ejecuta y se recupera el cursor que devuelve la función, haciendo casting para trabajar con el como si fuera un ResultSet:

```
	cs.execute();
	rs = (ResultSet)cs.getObject(1);
```

Por último solo nos queda recorrer el ResultSet como lo hemos hecho hasta ahora.

# 6. Statement vs PreparedStatement vs CallableStatement

Una vez que se obtiene una conexión, podemos interactuar con la base de datos. Las interfaces `JDBCStatement`, `CallableStatement` y `PreparedStatement` definen los métodos y propiedades que le permiten enviar comandos SQL o PL/SQL y recibir datos de su base de datos.

También definen métodos que ayudan a salvar las diferencias de tipo de datos entre los tipos de datos de Java y SQL utilizados en una base de datos.

La siguiente tabla proporciona un resumen del propósito de cada interfaz para decidir sobre la interfaz que se utilizará.

| **Interfaces** | **Uso recomendado** |
|---|---|
| Statement | Usada para acceso de propósito general a base de datos. Útil cuando se utilizan sentencias SQL estáticas en tiempo de ejecución. La interfaz de Statement no puede aceptar parámetros. |
| PreparedStatement | Usada cuando se planea usar las declaraciones SQL muchas veces. La interfaz PreparedStatement acepta parámetros de entrada en tiempo de ejecución. |
| CallableStatement | Usada cuando deseemos acceder a los procedimientos almacenados de la base de datos. La interfaz CallableStatement también puede aceptar parámetros de entrada en tiempo de ejecución. |

# 7. Acceso a BBDD con tablas temporales

Además de las tablas de la base de datos permanentes, Oracle permite la creación de tablas temporales para mantener datos propios y exclusivos a una sesión Oracle determinada. Estos datos permanecerán en el sistema sólo durante el tiempo que dure la transacción o sesión involucrada. No obstante, al igual que para las tablas permanentes, la definición de las tablas temporales se almacena en las tablas del sistema.

La sentencia `CREATE GLOBAL TEMPORARY TABLE` crea una tabla temporal Oracle cuya temporalidad puede ser definida a nivel de **transacción** (los datos existen mientras se realiza la transacción) o a nivel de **sesión** (los datos existen mientras dura la sesión). Los datos en una tabla temporal son propios y privativos de la sesión Oracle que la está utilizando. Una sesión Oracle determinada puede ver y modificar los datos que durante dicha sesión se insertaron en la tabla temporal, pero estos datos no son accesibles desde otra sesión diferente. Como es lógico, la sentencia LOCK no tiene efecto sobre las tablas temporales ya que cada sesión hace uso de sus propios datos.

Así pues, los datos almacenados en una tabla temporal son visibles sólo para la sesión de Oracle que inserta datos dentro de dicha tabla. Para especificar si los datos de una tabla temporal son por sesión o por transacción, a la hora de crear la definición de la tabla, utilizaremos la cláusula `ON COMMIT DELETE ROWS` para indicar que la temporalidad es a nivel de transacción, o la cláusula `ON COMMIT PRESERVE ROWS` si queremos que la temporalidad sea a nivel de sesión.

A continuación podéis ver un ejemplo de comando SQL para crear una tabla temporal en Oracle:

```
CREATE GLOBAL TEMPORARY TABLE temp_listado (
      nombre VARCHAR2(40),
      fecha_nacimiento DATE
) ON COMMIT PRESERVE ROWS
```

Asimismo, si ejecutamos una sentencia `TRUNCATE` sobre una tabla temporal, los datos que se truncarán serán los de la propia sesión desde la que se ejecuta la sentencia. Los datos que hayan podido ser insertados desde otras sesiones que estén utilizando la misma tabla, no se verán afectados por la sentencia `TRUNCATE`.

Los datos de una tabla temporal Oracle se borran automáticamente en el caso de que la sesión termine, bien porque el usuario desconecte, bien porque la sesión termine de una de manera anormal al producirse algún tipo de fallo.

## 7.1. Utilización de una Temporary Table

En el anexo 2, podemos observar un ejemplo práctico de la creación de una tabla temporal. Esta tabla temporal, una vez creada, estará disponible para ser usada por cualquier proceso o función que creemos. Vamos a ver un ejemplo en el que utilizamos dicha tabla temporal:

```
CREATE OR REPLACE PROCEDURE MOSTRAREQUIPO(V_EQUIPO VARCHAR2, jugadores_dev out SYS_REFCURSOR)
    IS
        n_dorsal jugar.dorsal%type;
        type arrayanyos is table of number index by binary_integer;
        mundiales arrayanyos;
        cadena varchar2(1000);
    BEGIN
        -- Por cada jugador del equipo pasado por parametro realizo las siguientes acciones
        FOR i in (select nombre, puesto_hab from JUGADOR where equipo_jugador=v_equipo) loop
            cadena:=NULL;

       -- Obtengo el dorsal que mas veces a llevado, en caso de que haya mas de uno, me
quedo con el menor
            select min(dorsal)
             into n_dorsal
            from
            (select dorsal
              from jugar
              where nombre_jug=i.nombre
              group by dorsal
              having count(*)=(select max(count(*))
                               from jugar
                               where nombre_jug=i.nombre
                               group by dorsal));

       -- Obtengo los distintos mundiales que ha jugado dicho jugador y lo inserto en una
tabla
            select distinct to_char(fecha_part, 'yyyy')
              bulk collect into mundiales
             from jugar
             where  nombre_jug=i.nombre;

       -- Recorro la tabla y voy componiendo una cadena de texto con la informacion de los
mundiales disputados por ese jugador.
             if mundiales.count<>0 then
                for j in mundiales.first .. mundiales.last  loop
                     cadena:=ltrim(cadena||' '||mundiales(j));
                end loop;
             else
                cadena:='Ninguno';
             end if;

           -- mostrar(n_dorsal||'-'||i.nombre||'-'||i.puesto_hab||'-     Mundiales
jugados:('||cadena||')');

       -- Inserto todos los valores dentro de la tabla temporal con un simple insert,
utilizando los campos del mismo tipo que los datos.
            INSERT INTO TMP_ESTRUCTURA
            (C1,N1,C2,C3,C4)
            VALUES
            (v_equipo, n_dorsal, i.puesto_hab, i.nombre, 'Mundiales('||ltrim(cadena)||')');
```

```
        end loop;

       -- Una vez insertadas todas las filas en la tabla temporal, abro un cursor sobre esa
tabla temporal, que es lo que se utilizara desde la aplicacio en JAVA.
        open jugadores_dev for select * from TMP_ESTRUCTURA order by n1;

    END MOSTRAREQUIPO;
```

## 7.2. Acceso desde JAVA

Una vez el procedimiento está disponible en Oracle, podemos acceder a el desde JAVA desde el exterior. Para ello vamos a llamar al procedimiento, pasandole como parametro la selección del mundial, de la que queremos obtener los jugadore, obtener el puntero de la tabla temporal con los datos, para poder presentarlos en una tabla.

```
Connection con = null;
ResultSet rs = null;
CallableStatement cs = null;
try{
      con = (new ConexionOracle(f)).Conectar();
      cs = con.prepareCall("{ call MOSTRAREQUIPOPRUEBA (?,?)}");

      int pos=0;
      cs.setString(++pos, equipo);
      cs.registerOutParameter(++pos, OracleTypes.CURSOR);

      cs.execute();
```

Una vez declaradas las variables necesarias (`Connection`, `ResultSet`, `CallableStatement`), obtenemos la conexion y preparamos la llamada. Observa que se dejan dos parámetros para completar, el primero es el `equipo` (la variable `equipo` la podemos haber creado previamente, y obtenido su valor mediante un `ComboBox`), y el segundo es un cursor. En el ejemplo se utiliza un contador `pos`, pero recuerda que puedes perfectamente utilizar un número entero. Una vez asignados los dos parámetros, y no antes, ya podemos ejecutar la sentencia.
Ya ejecutada la llamada al procedimiento, podemos recuperar el cursor con la siguiente sentencia:

```
rs = (ResultSet) cs.getObject(2);
```

En dicha sentencia, recogemos el segundo objeto del procedimiento (`OracleTypes.CURSOR`) y procedemos a hacer un casting para poder utilizar dicho cursor como un ResultSet. Asi pues ya solo debemos recorrer dicho ResultSet para mostrar los distintos valores como hemos visto en los apartados anteriores:

```
while (rs.next()) {
      System.out.println(    “Dorsal: “ + rs.getString("N1") + “\n” +
                             “Puesto: “ + rs.getString("C2") + “\n” +
                             “Nombre: “ + rs.getString("C3")+ “\n” +
                             “M.Jugados: “ + rs.getString("C4")+“\n”)
}
```

# 8. Acceso a BBDD OO.

Vamos a ver cómo podemos almacenar y obtener objetos de o desde una base de datos. En este caso vamos a ver cómo podemos hacerlo desde una base de datos Oracle. El almacenamiento y obtención de objetos, lo podemos hacer mediante estructuras o, a través de flujo de bytes. Primero vamos a utilizar estructuras.

## 8.1. La clase Struct

Por lo general, seguramente será más sencillo usar objetos STRUCT, en lugar de objetos personalizados de Java, en situaciones en las que se esté manipulando datos. Por ejemplo, si la aplicación Java fuese una herramienta para manipular datos dentro de la base de datos, en lugar de ser una aplicación de usuario final. Se puede seleccionar datos de la base de datos en objetos STRUCT y crear objetos STRUCT para insertar datos en la base de datos. Los objetos STRUCT conservan completamente los datos, porque mantienen los datos en formato SQL.

Veamos un caso real, supongamos que tenemos declarado un objeto Persona en la base de datos como se muestra a continuación:

```
CREATE OR REPLACE TYPE MUNDIAL.PERSONA AS OBJECT (
  DNI  NUMBER,
  NOMBRE  varchar2(32),
  APELLIDOS varchar2(50)
)
/
```

Y una tabla que almacene datos del tipo persona. Por ejemplo:

```
CREATE TABLE PERSONAS
(
  ID          NUMBER PRIMARY KEY,
  PERSON      PERSONA
)
```

Para el almacenar un objeto Persona, primero crearemos la clase persona en Java de la siguiente manera:

```
class Persona implements Serializable {
```

```
        int dni;
        String nombre;
        String apellidos;

        public Persona(int dni, String name, String apellidos) {

                this.nombre = name;
                this.dni = dni;
                this.apellidos = apellidos;
        }
        public Struct getStruct(Connection c) throws Exception {
                Object[] atr = {this.dni,this.nombre,this.apellidos};
                return c.createStruct("PERSONA", atr);
        }
        public String toString(){
                return  “dni: “ + dni + “\n” +
                        “nombre: “ + nombre + “\n” +
                        “apellidos: “ + apellidos + “\n”;
        }
}
```

Una vez declarada la clase Persona con su constructor y su método getStruct. Dicho metodo mapea los atributos de Persona para obtener un objeto Struct que introduciremos en la BBDD. Una vez dicho esto, procedemos a ver el código Java para el almacenaje en la BBDD.

```
Connection con = null;
PreparedStatement pstmt = null;
Struct str=null;

try {
        Persona persona = new Persona(53052298, "Joaquin", "Alonso");

        con = new ConexionOracle(ficheroBBDD).Conectar();

        pstmt = con.prepareStatement("INSERT INTO PERSONAS (ID,PERSON) VALUES(?,?)");
        pstmt.setInt(1, 1);
        pstmt.setObject(2, per.getStruct(con));
        pstmt.executeUpdate();

} catch (Exception e1) {
        e1.printStackTrace();
} finally {
        if (pstmt != null)
                pstmt.close();
}
```

El paso contrario, es decir el obtener los datos desde la base de datos los podemos ver en el siguiente código:

```
Connection con = null;
try{
        con = new ConexionOracle(ficheroBBDD).Conectar();
}catch(Exception e){
        e.printStackTrace();
}
Statement stmt = con.createStatement();
Persona person;
```

```
Object[] attributes;

try {
         ResultSet rs = stmt.executeQuery("SELECT * FROM PERSONAS");
         while (rs.next()) {

                System.out.println(rs.getInt(1));
                str = (Struct)rs.getObject(2);

                attributes = str.getAttributes();

                person = new Person(attributes[0],attributes[1],attributes[2]);
                System.out.println(person.toString());

} catch (Exception e1) {
        e1.printStackTrace();
} finally {
        if(stmt!=null) stmt.close();
        if (rs!=null) rs.close();
        ir(con!=null)con.close();
}
```

Como podemos observar, obtenemos un objeto y le hacemos un casting para pasarlo a Struct. Una vez lo tenemos como Struct, podemos invocar a su método `.getAttributes()`, que nos devuelve un vector de objetos con los atributos de dicho objeto, asi que ya solo debemos mostrarlos, o si utilizarlos para cualquier otro fin como podemos observar en el ejemplo.

## 8.2. Flujo de bytes.

En el siguiente ejemplo podemos ver cómo almacenar objetos utilizando un flujo de bytes. A diferencia del caso anterior, en este caso, los datos almacenados en Oracle serán de tipo BLOB. Para ello debemos tener una tabla como se muestra a continuación:

```
CREATE TABLE PERSONAS
(
  ID          NUMBER PRIMARY KEY,
  PERSON      BLOB
)
```

El codigo Java para almacenar datos:

```
Connection con = null;
PreparedStatement pstmt = null;

try {
        Persona persona = new Persona(53052298, "Joaquin", "Alonso");

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(persona);
        byte[] seleccionadorAsBytes = baos.toByteArray();

        con = new ConexionOracle(ficheroBBDD).Conectar();
        ByteArrayInputStream bais = new ByteArrayInputStream(seleccionadorAsBytes);
```

```
        pstmt = con.prepareStatement("INSERT INTO PERSONAS (ID,PERSON) VALUES(?,?)");
        pstmt.setInt(1, 1);
        pstmt.setBinaryStream(2, bais, seleccionadorAsBytes.length);
        pstmt.executeUpdate();

} catch (Exception e1) {
        e1.printStackTrace();
} finally {
        if (pstmt != null)
                pstmt.close();
}
```

Como podemos ver en el codigo, este transforma, mediante flujos, el objeto persona en un vector de bytes, y posteriormente se almacena en la base de datos. El proceso para obtener el objeto persona seria exactamente igual, pero en sentido contrario, es decir:

```
Connection con = null;
try{
        con = new ConexionOracle(ficheroBBDD).Conectar();
}catch(Exception e){
        e.printStackTrace();
}
Statement stmt = con.createStatement();
Persona person;
ByteArrayInputStream baip=null;
ObjectInputStream ois=null;

try {
         ResultSet rs = stmt.executeQuery("SELECT * FROM PERSONAS");
         while (rs.next()) {

                System.out.println(rs.getInt(1));
                byte[] st = rs.getBytes(2);
                baip = new ByteArrayInputStream(st);
                ois = new ObjectInputStream(baip);
                person = (Persona) ois.readObject();
                System.out.println(person.toString());

} catch (Exception e1) {
        e1.printStackTrace();
} finally {
        if(stmt!=null) try{stmt.close();}catch(Exception ignore){}
        if (rs!=null) try{rs.close();}catch(Exception ignore){}
        ir(con!=null) try{con.close();}catch(Exception ignore){}
}
```

# 9. Ampliación

Para más información, se puede consultar la documentación oficial aquí.
También se puede obtener más información de la clase OracleCallableStatement, que hereda de CallabelStatement, pero que tiene unos métodos especiales para trabajar con la bases de datos de Oracle, como por ejemplo OracleCallableStatement.registerIndexTableOutParameter()

# Anexo 1. Código fuente completo

```
package javaOracle;

import java.io.File;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.ParseException;
import java.text.SimpleDateFormat;

public class Prueba2 {
	public static void main(String[] args) throws SQLException {
		File f = new File("./configuracion");
		Connection cn = new ConexionOracle(f).Conectar();
		if (cn == null)
			System.out.println("Error");

		viewTable(cn, "MUNDIAL");

		viewTableWithPreparedStatement(cn, "MUNDIAL", "ESPAÑA");

		cn.close();
	}

	public static void viewTable(Connection con, String dbName)
throws SQLException {

		Statement stmt = null;
		String query = "select NOMBRE, DIRECCION,PUESTO_HAB," +
"FECHA_NAC, EQUIPO_JUGADOR " + "from " + dbName
				+ ".JUGADOR";

		try {
			stmt = con.createStatement();
			ResultSet rs = stmt.executeQuery(query);

			while (rs.next()) {

				String nombre = rs.getString("NOMBRE");
				String direccion = rs.getString("DIRECCION");
```

```
                    String puesto = rs.getString("PUESTO_HAB");
                    Date fecha = rs.getDate("FECHA_NAC");
                    String equipo =
rs.getString("EQUIPO_JUGADOR");

                    System.out.println(nombre + "\t" + direccion +
"\t" + puesto + "\t"
                                + ((fecha == null) ? "Desconocido"
: fecha.toString()) + "\t" + equipo);
               }
          } catch (SQLException e) {
               e.printStackTrace();
          } finally {
               if (stmt != null) {
                    stmt.close();
               }
          }
     }

     public static void viewTableWithPreparedStatement(Connection
con, String dbName, String team) throws SQLException {

          PreparedStatement pstmt = null;
          String query = "select NOMBRE, DIRECCION,PUESTO_HAB," +
"FECHA_NAC, EQUIPO_JUGADOR " + "from " + dbName
                    + ".JUGADOR " + "where EQUIPO_JUGADOR = ?";

          try {
               pstmt = con.prepareStatement(query);

               pstmt.setString(1, team);

               ResultSet rs = pstmt.executeQuery();

               while (rs.next()) {

                    String nombre = rs.getString("NOMBRE");
                    String direccion = rs.getString("DIRECCION");
                    String puesto = rs.getString("PUESTO_HAB");
                    Date fecha = rs.getDate("FECHA_NAC");
                    String equipo =
rs.getString("EQUIPO_JUGADOR");

                    System.out.println(nombre + "\t" + direccion +
"\t" + puesto + "\t"
```

```
                                    + ((fecha == null) ? "Desconocido"
: fecha.toString()) + "\t" + equipo);
                }
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                if (pstmt != null) {
                    pstmt.close();
                }
            }
      }

}
```

# ANEXO 2. EJEMPLO DE TABLA TEMPORAL

```
 create global temporary table tmp_estructura (
      id_transaccion number,
      origen_datos varchar2(4000),
      n1 number,
      n2 number,
      n3 number,
      n4 number,
      n5 number,
      n6 number,
      n7 number,
      n8 number,
      n9 number,
      n10 number,
      c1 varchar2(4000),
      c2 varchar2(4000),
      c3 varchar2(4000),
      c4 varchar2(4000),
      c5 varchar2(4000),
      c6 varchar2(4000),
      c7 varchar2(4000),
      c9 varchar2(4000),
      c10 varchar2(4000)
) on commit preserve rows;
```

```
C:\app\Alumno\product\11.2.0\dbhome_1\NETWORK\ADMIN
```

---

```
# listener.ora Network Configuration File:
C:\app\Alumno\product\11.2.0\dbhome_1\network\admin\listener.ora
# Generated by Oracle configuration tools.
```

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = CLRExtProc)
      (ORACLE_HOME = C:\app\Alumno\product\11.2.0\dbhome_1)
      (PROGRAM = extproc)
      (ENVS =
"EXTPROC_DLLS=ONLY:C:\app\Alumno\product\11.2.0\dbhome_1\bin\oraclr11.dll")
    )
  )

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = DESKTOP-7KSRFS2)(PORT = 1521))
    )
  )

ADR_BASE_LISTENER = C:\app\Alumno
```

---

```
# tnsnames.ora Network Configuration File:
C:\app\Alumno\product\11.2.0\dbhome_1\network\admin\tnsnames.ora
# Generated by Oracle configuration tools.

ORACLR_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
    (CONNECT_DATA =
      (SID = CLRExtProc)
      (PRESENTATION = RO)
    )
  )

LISTENER_ORCL =
  (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))


ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 11.0.3.112)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = ORCL)
    )
  )
```