Programación. 1º DAW

Unidad 7: Gestión de bases de datos relacionales

Gestión de bases de datos relacionales

- Introducción.
- Uso y tipos de drivers
  - Puente JDBC-ODBC
  - Protocolo Nativo
- JDBC (Java DataBase Connectivity).
  - Clases e interfaces del paquete java.sql
  - Esquemas típicos de uso:
    - Establecimiento de conexiones.
    - Recuperación de información.
      - Cursores:ResultSet
    - Manipulación de la información.
    - Ejecución de sentencias SQL sobre la base de datos.
    - Control de errores: SQLException
    - Transacciones
    - Llamada a procedimientos almacenados

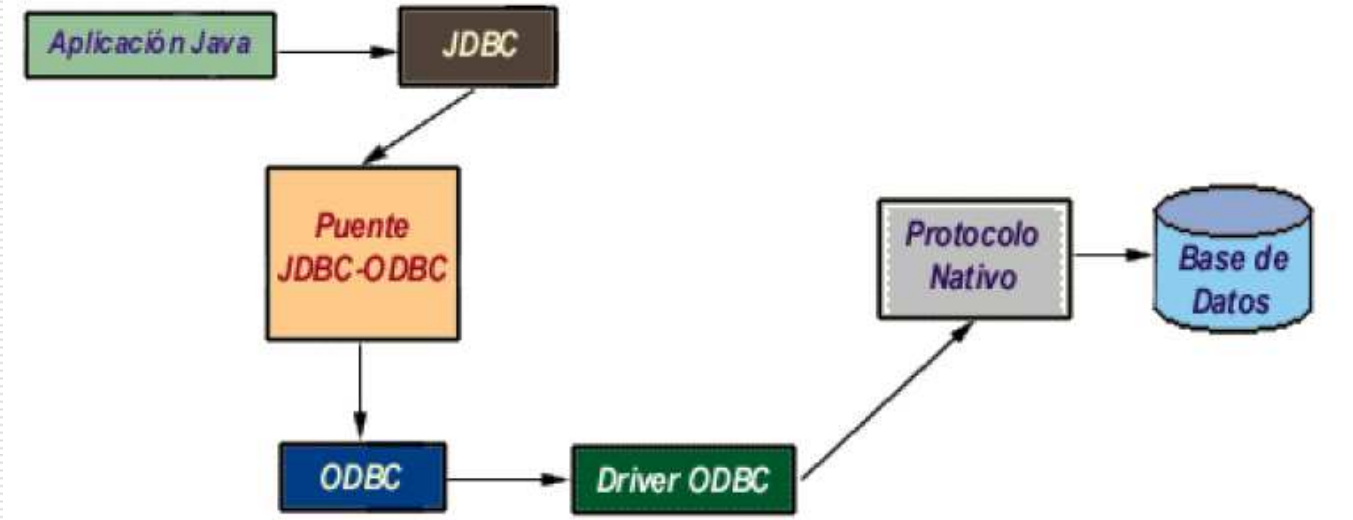
Introducción

- **SQL** (*Structured Query Language*) que es un lenguaje de muy alto nivel que permite crear, examinar, manipular y gestionar Bases de Datos relacionales.
- **JDBC** (*Java DataBase Connectivity*) es un API de Java que permite al programador ejecutar instrucciones en el lenguaje estándar de acceso a Bases de Datos **SQL**
  - Para que una aplicación pueda hacer operaciones en una Base de Datos, ha de tener una conexión con ella, que se establece a través de un *driver*, que convierte el lenguaje de alto nivel a sentencias de Base de Datos.
  - Las tres acciones principales que realizará JDBC son:
    - Establecer la conexión a una base de datos, ya sea remota o no
    - Enviar sentencias SQL a esa base de datos
    - Procesar los resultados obtenidos de la base de datos.
- **SQLJ** es un estándard ISO para embeber sentencias SQL en programas de Lenguaje de programación Java.
  - SQLJ no es una API sino una extensión del lenguaje.
  - Los programas SQLJ deben ejecutarse a través de un preprocesador (el traductor SQLJ) antes de que puedan ser compilados.

Puente JDBC-ODBC

- Para la gente del mundo Windows, **JDBC** es para Java lo que **ODBC** es para Windows.
- **JDBC** es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea.
  - Al igual que ODBC, la aplicación de Java debe tener acceso a un driver JDBC adecuado.
  - Este driver es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

Puente JDBC-ODBC



- Ventajas
  - Se proporciona con el JDK
  - Java dispone de acceso inmediato a todas las fuentes posibles de bases de datos y no hay que hacer ninguna configuración adicional aparte de la ya existente
- Inconvenientes
  - Los drivers ODBC convierten suscllamadas a llamadas a una librería nativa del fabricante DBMS, generando lentitud
  - El puente JDBC-ODBC requiere una instalación ODBC ya existente y configurada, lo que implica limitar la aplicación a entornos Windows

Protocolo Nativo JDBC

- Es un driver realizado completamente en Java que se comunica con el servidor DBMS utilizando el protocolo de red nativo del servidor.
  - El driver no necesita intermediarios para hablar con el servidor
  - Convierte todas las peticiones JDBC en peticiones de red contra el servidor.
- Ventajas:
  - es una solución *100% Java* y, por lo tanto, independiente del Sistema Operativo en la que se va a ejecutar el programa
  - Puede no necesitar ninguna clase de configuración por parte del usuario
- Inconveniente
  - El cliente está ligado a un servidor DBMS concreto

JDBC (Java DataBase Connectivity)

- **JDBC** es la interfaz que proporciona Java para la conexión a bases de datos.
  - Son un conjunto de clases e interfaces que permiten a Java ejecutar consultas y ordenes en una bases de datos
  - Para trabajar con JDBC es necesrio importar el paquete java.sql.

```
import java.sql.*;
```

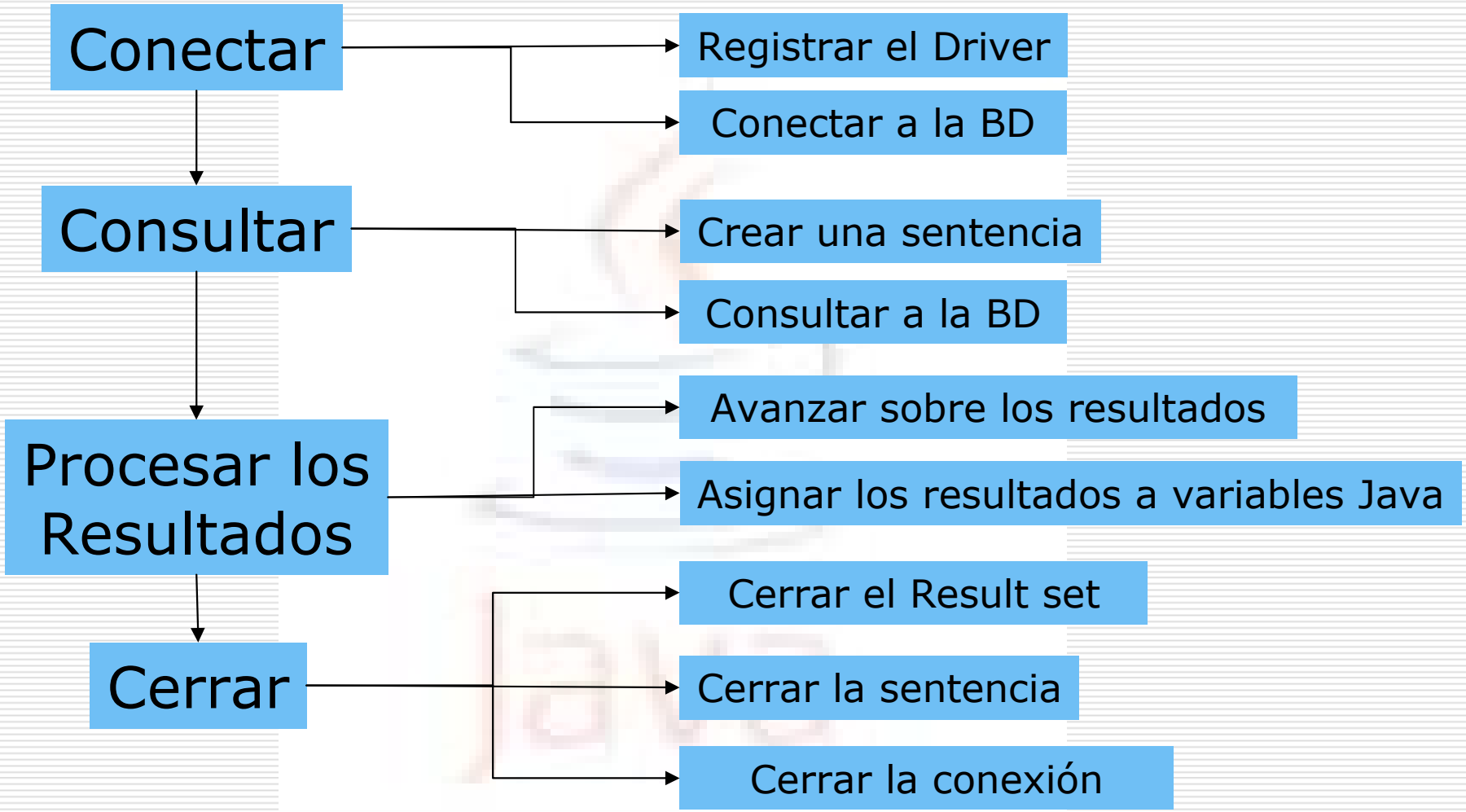
JDBC: Clases e interfaces del Paquete java.sql

- **Driver:**
  - Permite conectarse a una base de datos
  - Cada gestor de base de datos requiere un driver distinto
- **DriverManager:**
  - Permite gestionar todos los drivers instalados en el sistema
- **DriverPropertyInfo:**
  - Proporciona diversa información acerca de un driver
- **Connection:**
  - Representa una conexión con una base de datos.
  - Una aplicación puede tener más de una conexión a más de una base de datos
- **DatabaseMetadata:**
  - Proporciona información acerca de una Base de Datos, como las tablas que contiene, etc.

JDBC: Clases e interfaces del Paquete java.sql

- **Statement:**
  - Permite ejecutar sentencias SQL sin parámetros
- **PreparedStatement:**
  - Permite ejecutar sentencias SQL con parámetros de entrada
- **CallableStatement:**
  - Permite ejecutar sentencias SQL con parámetros de entrada y salida, típicamente procedimientos almacenados
- **ResultSet:**
  - Contiene las filas o registros obtenidos al ejecutar un SELECT
- **ResultSetMetadata:**
  - Permite obtener información sobre un **ResultSet**: número de columnas, sus nombres, etc.

Esquema típico de uso: Conexión, recuperación y manipulación de información



Conexión, recuperación y manipulación de datos con MySQL

```
import java.sql.*;
class dbAccess {
        public static void main (String args []) throws SQLException
        {
            //CONECTAR: Asegurarnos de que el Driver se carga en memoria de java, para ello
            Class.forName("com.mysql.jdbc.Driver").newInstance();

            //CONECTAR: CONECTAMOS A LA BASE DE DATOS
            Connection conn = DriverManager.getConnection
                        ("jdbc:mysql://localhost/mybd", "user", "passw");


            //CONSULTAR: CREAMOS UN OBJETO STATEMENT VACÍO
            Statement stmt = conn.createStatement();

            //CONSULTAR: EJECUTAMOS EL STATEMENT
            ResultSet rset = stmt.executeQuery ("select colum1,colum2,...,columN from Tabla");

            //PROCESAR LOS RESULTADOS
            while (rset.next()) //AVANZAMOS SOBRE EL RESULTSET
                System.out.println (rset.getString(N)); //OBTENEMOS EL VALOR DE LA COLUMNA N

            //CERRAR: CERRAMOS LA CONEXIÓN
            rset.close();
            stmt.close();
            conn.close();

        }
}
```

Conexión, recuperación y manipulación de datos con ORACLE

```
import java.sql.*;
□       class dbAccess {
          public static void main (String args []) throws SQLException
          {
              //CONECTAR: Asegurarnos de que el Driver se carga en memoria de java, para ello
              Class.forName("oracle.jdbc.driver.OracleDriver");

              //CONECTAR: CONECTAMOS A LA BASE DE DATOS
              Connection conn = DriverManager.getConnection
                            ("jdbc:oracle:thin:@myhost:1521:sid", "user", "passw");

              //CONSULTAR: CREAMOS UN OBJETO STATEMENT VACÍO
              Statement stmt = conn.createStatement();

              //CONSULTAR: EJECUTAMOS EL STATEMENT
              ResultSet rset = stmt.executeQuery
                            ("select colum1,colum2,...,columN from Tabla");

              //PROCESAR LOS RESULTADOS
              while (rset.next()) //AVANZAMOS SOBRE EL RESULTSET
                    System.out.println (rset.getString(N)); //OBTEN. VALOR COLUMNA N
                    System.out.println (rset.getString("COLUM1")); //OBTIENE VALOR ATRIBUTO COLUM1
                     //Los datos obtenidos son de tipo String.
                    //Si en la BD fueran numéricos, tenemos que convertirlos con parse.
              //CERRAR: CERRAMOS LA CONEXIÓN
              rset.close();
              stmt.close();
              conn.close();

          }
        }
```

Cursores: ResultSet

- Los cursores en JDBC son los ResultSet.
- Cuando se crea un ResultSet este se posiciona antes de la primera fila de datos.
- Los cursores por defecto unidireccionales y solo se mueven hacia delante.
- Se pueden crear cursores bidireccionales, que utilizan métodos para desplazarse por los datos

Cursores: ResulSet

- Tipos de ResulSet:
  - TYPE_FORWARD_ONLY
    - Cursor por defecto.
    - Unidireccional. Solo se mueve hacia delante.
  - TYPE_SCROLL_INSENSITIVE
    - Bidireccional.
    - No sensibles a cambios en los datos que subyacen a la ResultSet
  - TYPE_SCROLL_SENSITIVE
    - Bidireccional.
    - Sensibles a los cambios en los datos que subyacen en el ResultSet .

Cursores: ResulSet

- Concurrencia
  - CONCUR_READ_ONLY
    - Cursor por defecto.
    - Los datos de la BD no pueden ser actualizados.
  - CONCUR_UPDATABLE
    - Los datos de la BD pueden ser actualizados.
- Persistencia
  - HOLD_CURSORS_OVER_COMMIT
    - El ResultSet **NO** se cierra cuando se ejecuta el método commit.
  - CLOSE_CURSORS_AT_COMMIT
    - El ResultSet **SI** se cierra cuando se ejecuta el método commit.

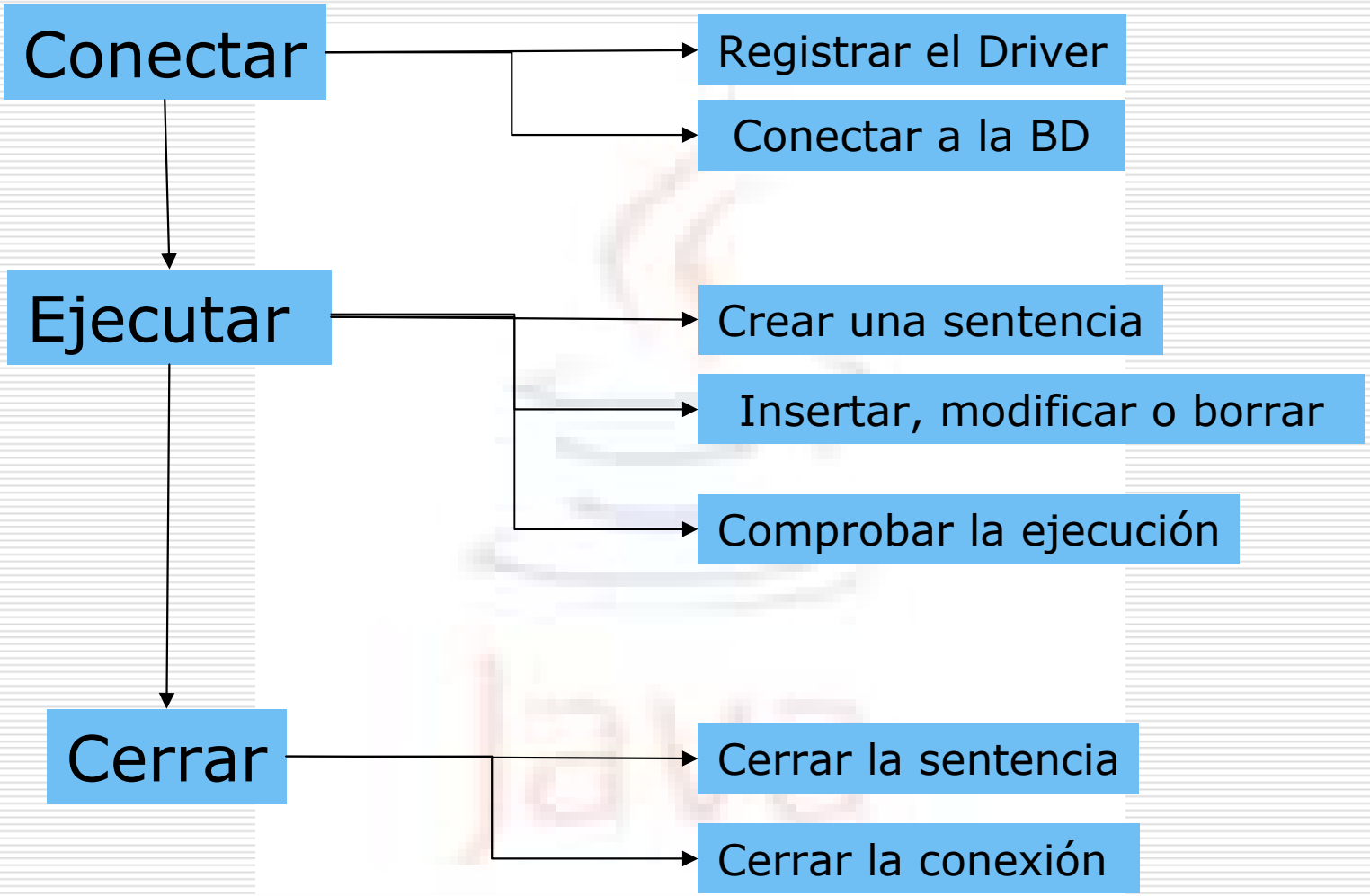
Cursores: ResulSet

- Los métodos para tratar cursores bidireccionales son:
  - **next()**
    - Mueve el cursor una posición hacia delante.
    - Devuelve true si el cursor se posiciona en una fila y false en caso de que esté después de la última fila.
  - **previous()**
    - Mueve el cursor una posición hacia atras.
    - Devuelve true si el cursor se posiciona en una fila y false en caso de que esté antes de la primera fila.
  - **first()**
    - Coloca el cursor en la primera fila.
    - Devuelve true si el cursor contiene al menos una fila y false en caso contrario.
  - **last()**
    - Coloca el cursor en la última fila.
    - Devuelve true si el cursor contiene al menos una fila y false en caso contrario.
  - **beforeFirst():** Coloca el cursor antes de la primera fila
  - **afterLast**(): Coloca el cursor después de la última fila
  - **relative(int rows):** Mueve el cursor un número relativo de filas, ya sea positivo o negativo
  - **absolute(int rows):** Mueve el cursor al número de fila indicado

Cursores: ResulSet

- Los métodos para tratar cursores bidireccionales son:
  - **deleteRow** (): Elimina la fila actual de este ResultSet y el objeto de la base de datos subyacente.
  - **getRow** () Recupera el número de fila actual.
  - **getInt (int columnIndex)** Recupera el valor de la columna designada en la fila actual del ResultSet objeto como un int
  - **getInt (String columnLabel)** Recupera el valor de la columna designada en la fila actual del ResultSet objeto como un int.
  - **updateDouble (int columnIndex, double x)** Actualiza la columna designada con un double
  - **updateDouble (String columnLabel, double x)** Actualiza la columna designada con un double valor.
  - **updateRow** () Actualiza la base de datos subyacente con los nuevos contenidos de la fila actual de este ResultSet.
  - **insertRow** () Inserta el contenido de la fila a insertar en este ResultSet y en la base de datos.

Esquema típico de uso: Ejecución de sentencias



Ejecución de sentencias y control de errores

```
//EJECUTAMOS EL STATEMENT
try {
    //Ejecutamos una sentencia SQL
    stmt.executeUpdate("Insert into.....");
} catch (SQLException e) {
    // Controlamos el error, si ocurre
    e.getErrorCode();
    e.getMessage();
    e.getSQLState();
}
```

Transacciones

- Transacciones
  - **setAutoCommit (boolean autoCommit)**
    - Establece en la conexión el modo de confirmación automática
  - **getAutoCommit ()**
    - Recupera el actual modo de confirmación automática de la conneción.
  - **commit** ()
    - Hace que todos los cambios realizados desde el anterior commit / rollback permanente.
  - **rollback** ()
    - Deshace todos los cambios realizados en la transacción actual y libera todos los bloqueos de base de datos actualmente en manos de la conexión.
  - **rollback** (Savepoint savepoint)
    - Deshace todos los cambios realizados después de que la orden Savepoint fue creada.

Llamada a procedimientos almacenados

- CallableStatement
  - Los objetos de la clse CallableStatement son utilizados en Java para llamar a procedimientos almacenados de la base de datos
  - Se utiliza de la siguiente manera:

```
Connection conn; //Conectamos
//Llamamos al procedimiento
callableStatement cs=conn.prepareCall("{call nombreProcedimiento(?)}");
// Si el procedimieto tiene parámetros de entrada (IN), lo actualizamos
cs.setString(1,"HOLA");
//Si el procedimiento tiene parámetros de salida (OUT), lo registramos con //su tipo
cs.registerOutParameter(1,Types.INTEGER);
//Ejecutamos el procedimiento
cs.execute();
//recuperamos el parametro OUT
int numero=cs.getInt(1);
```

- NOTA: La llamada a funciones de base de datos se realiza utilizando ResultSet ya que devuelven un único valor.

```
ResultSet rs=stmt.executeQuery("Select nombrefuncion(argumentos) from dual");
rs.next();
String res=rs.getString(1);
```