

Statement vs PreparedStatement y Pool de Conexiones

REPOSITORIOS DE INFORMACION

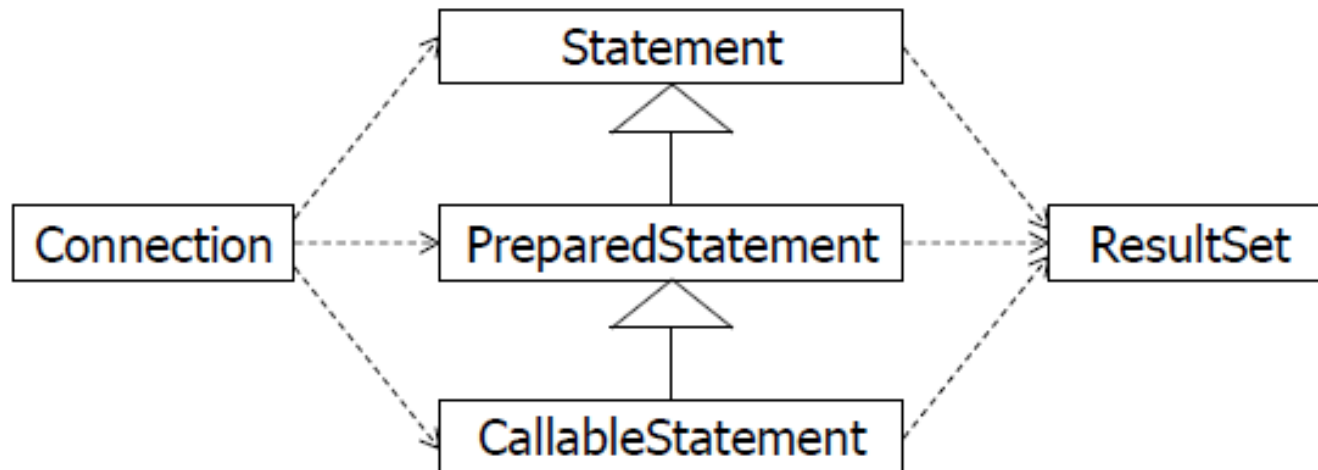
Grado en Ingeniería Informática del Software

- Objetivos de la Práctica:
 - Repasar conceptos ya conocidos de JDBC
 - Realizar una comparativa Statement/PreparedStatement
 - Hacer conexiones a diversos SGBD: Oracle y HSQLdb
 - Usar un Pool de conexiones
- Preparación de la práctica:
 - Arrancar el servidor HSQLdb → Descargar y descomprimir la base de datos CarWorkShop y ejecutar data/startup
 - Arrancar el servidor de Oracle y realizar la conexión con el usuario corporativo.
 - Ejecutar el Script para crear las tablas en Oracle
 - Descargar la carpeta lib con los drivers necesarios para las aplicaciones Java
 - Las aplicaciones Java serán desarrolladas usando Eclipse

- Datos de Conexión
 - Oracle
 - URL= "jdbc:oracle:thin:@156.35.94.99:1521:DESA"; *//(puerto 1521)*
 - USER="XXXXXX"; *//Correo corporativo (UOXXXXXXX)*
 - PASS="password";
 - HSQLDB
 - URL="jdbc:hsqldb:hsq://localhost";
 - USER="sa"; *//Usuario por Defecto*
 - PASS="";
- De esta parte de prácticas **NO HAY ENTREGA**
- Se hará uso de lo aprendido en una práctica posterior

Repaso de JDBC

- JDBC → *Java DataBase Connectivity*
- API que permite la ejecución de instrucciones sobre bases de datos desde Java de forma independiente al SO
- Basado en lenguajes SQL
- Permite la creación de Aplicaciones
- Jerarquía de clases



Repaso de JDBC (II)

- Pasos para el empleo de JDBC en una aplicación:
 - Establecer conexión
 - Realizar consulta
 - Cerrar todo
 - De forma transversal: Controlar los errores (try – catch)
 - Establecimiento de Conexión:
 - Cargar el driver
 - Realizar conexión
- ```
con = DriverManager.getConnection(URL, USER, PASS);
```
- Realizar Consulta:
    - Crear objeto de tipo “Statement” (*Statement stmt = con.createStatement();*)
    - Especificar sentencia SQL a ejecutar (ejemplo: consulta SELECT - string)
    - Ejecutar la instrucción (*stat.executeQuery(sentencia)*)
  - Cierre de Conexión:
    - *con.close() //hay que tener en cuenta el manejo de errores*

- Tres tipo de sentencias:
- **Statement:** para SQL (DML, DDL) sin parámetros
- **PreparedStatement:** para SQL con parámetros o ejecuciones repetidas. Más eficiente y seguro. Se usará siempre en vez de *Statement* para DML
- **CallableStatement:** para invocar procedimientos almacenados
- Todos se obtienen a través de un elemento “*Connection*”


- Ejecución:
  - `executeQuery (<SQL>,...)` → Ejecutar consultas → `ResultSet`
  - `executeUpdate (<SQL>,...)` → Sentencias DDL y DML → Nº Filas
  - `execute (<SQL>,...)` → Devuelve booleano indicado tipo resultado
- Ejemplo:

```
Statement stmt = con.createStatement();
String sentencia = "SELECT * FROM libros";
stat.executeQuery(sentencia);
```

# Sentencia PreparedStatement – Repaso JDBC

- Sentencias SQL precompiladas
  - Más rendimiento si hay ejecución repetida (caso real)
  - Mayor seguridad (por ejemplo ante SQL Injection)
- Admisión de parámetros de entrada (*placeholders*) identificados por el símbolo “?” → Necesitan valor antes de la ejecución → Se puede variar el valor (asignando valores)
- Asignación valores: *set<tipo>(pos,valor)* → **pos empieza en 1**
- <tipo> = todos los tipos básicos (*setInt*, *setLong*, *setString*, etc)
- Existe el tipo *setNull(pos,tipo)* → Valores de parámetros a Null → tipo SQL

```
...
consulta= "Select count(*) cuantos from bd.coches where
codcoche=?";
PreparedStatement ps = con.prepareStatement(consulta);
...
ps.setInt(1,i);
rs = ps.executeQuery();
...
```



```
pstmt.setNull(1,java.sql.Types.VARCHAR);
pstmt.setNull(2,java.sql.Types.NUMERIC);
pstmt.setNull(3,java.sql.Types.BLOB);
```



# Sentencia PreparedStatement – Repaso JDBC (II)

- Ejecución
  - Hereda de Statement (executeQuery, executeUpdate, execute)
- Importante → **NO LLEVAN SQL EN LA INVOCACIÓN** (fijamos antes la sentencia)

```
PreparedStatement ps = con.prepareStatement(
 "UPDATE coches SET modelo = ? WHERE cod = ?");
ps.setString(1, "Ibiza");
ps.setInt(2, 25);
ps.executeUpdate(); //sin SQL
```

- Resumen

1. Especificar sentencia SQL
  2. Crear el *PreparedStatement*
  3. Asignar a cada parámetro su valor con el método “set”
  4. Ejecutar la consulta
- 

```
String consulta1 = "SELECT Libros.precio, Libros.titulo " +
 "FROM Libros, Editoriales "+
 "WHERE Libros.codEditorial = Editoriales.codEditorial and Editoriales.Nombre = ?";
```

```
PreparedStatement psConsulta = con.prepareStatement(consulta1);
```

```
psConsulta.setString(1,"McGrawHill"); //EMPIEZA EN 1
```

```
ResultSet rs = psConsulta.executeQuery() //SIN PARÁMETROS
```

# ResultSet – Repaso JDBC

- Conjunto de Filas y Cursor
- Lo devuelven los métodos “*executeQuery(...)*” y “*execute(...)*”
- Posición Inicial → Antes del primer registro

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery (“SELECT a, b, c FROM tabla”);
while (rs.next())
{
 int i = rs.getInt(“a”);
 String s = rs.getString(“b”);
 float f = rs.getFloat(“c”);

 System.out.println(i+” “+s+” “+f);
}
```

# ResultSet – Repaso JDBC (II)

- Los cursores indican la fila activa del ResultSet
- Dependiendo de la dirección:
  - Sólo hacia delante
  - Bidireccionales
- Por defecto sólo “FORWARD” → menos recursos
- Movimientos: `rs.first()`, `rs.last()`, `rs.next()`, `rs.previous()`, `rs.absolute(5)` (quinta fila), `rs.relative(-3)` //tres filas antes
- Control: `rs.isBeforeFirst()`, `rs.isAfterLast()`, `rs.isFirst()`, `rs.isLast()`
- Métodos “getter”: `rs.get<TIPO>(pos|nombre)` → Indicar posición o nombre (mejor nombre, si hay cambio consulta o definición la tabla sigue funcionando)

```
String s = rs.getString(2);
String s = rs.getString("codigo");
```

# Control de Errores – Repaso JDBC

- Error → Lanza excepción Java
- Excepciones que lanzan operaciones JDBC → SQLException
- try ... catch (SQLException)
- Métodos
  - String getSQLState() → identifica el error de acuerdo a X/Open
  - int getErrorCode() → obtiene el código de excepción específico
  - String getMessage() → obtiene cadena que describe excepción
  - SQLException getNextException() → excepción encadenada
- SQLWarning → SQLWarning war = rs.getWarnings();
- Cláusula finally → importancia cerrar recursos que vamos abriendo

```
finally {
 if (rs != null){ try { rs.close(); } catch (SQLException e){}
 if (stmt != null){ try { stmt.close(); } catch (SQLException e){}
 if (con != null){ try { con.close(); } catch (SQLException e){}
}
```

- 3 alternativas
  - Driver
  - DataSource
  - Instanciar el driver directamente
- Resultado → objeto de tipo Connection → Statement, PreparedStatement, CallableStatement, etc.
- Revisar transparencias de teoría

- Varias conexiones simultáneas a un SGBD
- Problema → Tiempo y Recursos de conexiones es costoso
- Necesario DataSource para pool de conexiones
- Puede venir implementado en el driver del fabricante o puede utilizarse uno genérico
- Diferencia:
  - PooledConnection. Conexión física, puede “reciclarse”
  - Pool de Conexiones (implementado por terceros, sin estándar)
    - *OracleConnectionCacheManager (Oracle), C3p0, BoneCP, Jakarta DBCP*

```
DataSource unpooled = DataSources.unpooledDataSource(
 "jdbc:oracle:thin:@156.35.94.99:22:DESA",
 "user",
 "password"
);

Map overrides = new HashMap();
overrides.put("maxStatements", "200");
overrides.put("maxPoolSize", new Integer(50));
overrides.put("minPoolSize", "6");

DataSource pooled = DataSources.pooledDataSource(unpooled, overrides);

con = pooled.getConnection();
```



- Uso de IDE Eclipse para Java
- Creación de Statement vs PreparedStatement

```
long time_start, time_end;
time_start = System.currentTimeMillis();
Programa_Java //Ejecución de bucle con la misma consulta
time_end = System.currentTimeMillis();
System.out.println("Tiempo transcurrido"+ (time_end - time_start) +" milisegundos");
```

- Consulta compleja que vaya cambiando en cada iteración del bucle (cambio de id en función del identificador)
- Statement → Concatenación de cadenas
- PreparedStatement → Placeholders “?”

- Programa para evaluar el coste de abrir conexiones a la base de datos
- Medición de tiempos y comparación en 2 casos:
  1. Bucle de 100 repeticiones (for) teniendo la apertura de la conexión, consulta y cierre de la conexión incluidas en el bucle
  2. Bucle de 100 repeticiones (for) teniendo la apertura y cierre de la conexión fuera del bucle

- Emplearemos pool de conexiones c3p0 (disponible en campus virtual)
- Creación de pool de conexiones con las siguientes características:
  - Máximo de conexiones: 30
  - Mínimo de conexiones: 3
  - Tamaño inicial del pool: 3
- Ejercicio de comparativa entre el uso de pool de conexiones y conexión sin pool de conexiones (contar número de vehículos de la tabla con un “for”)
- Ejemplo y explicación desarrollada en Fichero “Práctica JDBC” del campus virtual

