Java JDBC API

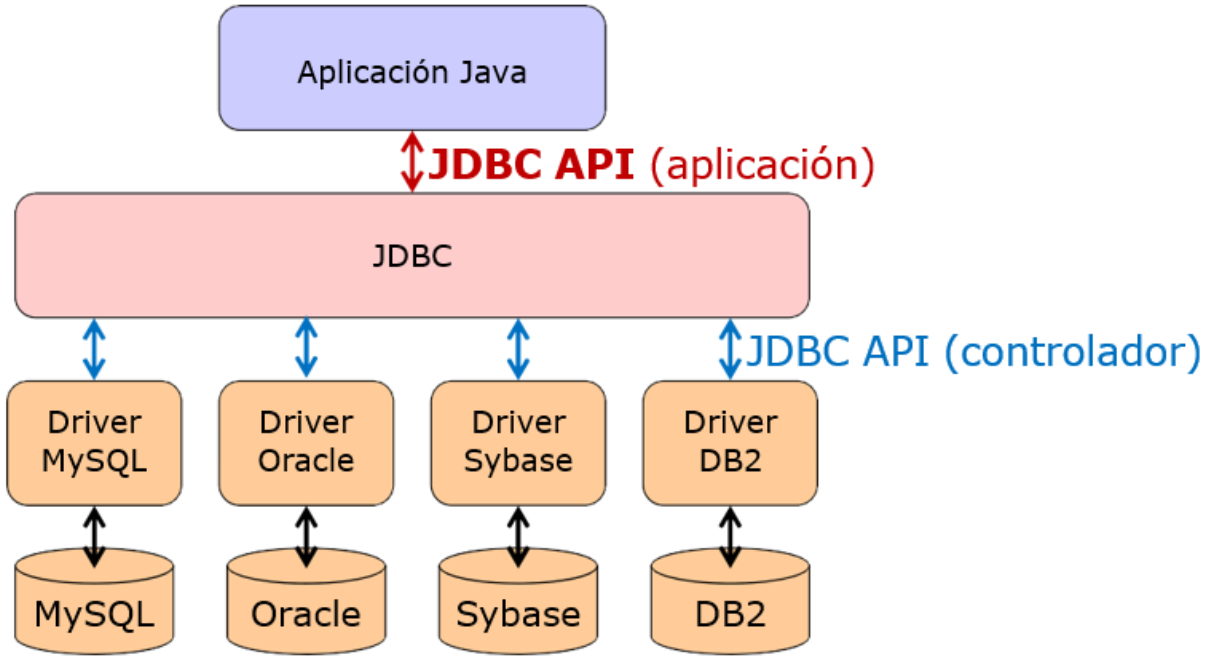
Accés a Dades

IES Serpis curso 2023-24

MªÁngeles Chover

JDBC

- Java Database Connectivity
  - API Java para conectar las aplicaciones a bases de datos
  - Arquitectura modular
    - La misma interfaz para distintos tipos de bases de datos
    - Implementa un gestor de drivers de bases de datos



Paquete java.sql

- Uso de controladores de las BD
  - Clase DriverManager
    - Permite establecer y gestionar conexiones a las BD
  - Clase SQLPermission
    - Proporciona los permisos para poder usar el DriverManager a código en ejecución dentro de un Security Manager (por ejemplo applets)
  - Interfaz Driver
    - Metodos para registrar y conectar controladores basados en tecnología JDBC
  - Clase DriverPropertyInfo
    - Propiedades de un controlador
- Excepciones
  - SQLException
  - SQLWarning

Paquete java.sql

- Interfaz con la aplicación
  - Envío de instrucciones SQL a la BD
    - Connection
      - Métodos para crear instrucciones y para gestionar conexiones y sus propiedades
    - Statement
      - Permite enviar instrucciones a la BD
    - PreparedStatement
      - Permite usar instrucciones preparadas o SQL básicas
    - CallableStatement
      - Llamada a procedimientos almacenados en la BD
    - Savepoint
      - Puntos de recuperación en una transacción
  - Recuperación de los resultados de la consulta a la BD
    - ResultSet
      - Conjunto de resultados que se devuelven de una query
    - ResultSetMetaData
      - Información sobre las columnas del objeto ResultSet

Paquete java.sql

- Interfaz con la aplicación
  - Correspondencia de tipos SQL con clases e interfaces de Java
    - Array → SQL ARRAY
    - Blob → SQL BLOB
    - Clob → SQL CLOB
    - Date → SQL DATE
    - NClob → SQL NCLOB
    - Ref → SQL REF
    - RowId → SQL ROWID
    - Struct → SQL STRUCT
    - SQLXML → SQL XML
    - Time → SQL TIME
    - Timestamp → SQL TIMESTAMP
    - Clase Types → constantes para tipos SQL
  - Correspondencia de tipos SQL definidos por el usuario a Java
    - SQLData
    - SQLInput
    - SQLOutput

Secuencia uso JDBC

- **Secuencia normal:**
  - **Establecer la conexión** con la BD
    - Cargar controladores (*si se usa una versión de Java inferior a la 6*)
    - Establecer la conexión
  - Crear un **objeto Statement** para hacer petición a la BD
    - Asociar una sentencia SQL al objeto Statement
    - Proporcionar valores de los parámetros
    - Ejecutar el objeto Statement
  - Procesar los **resultados**
  - Liberar recursos (**cerrar la conexión**)

- Si es necesario, se pueden ejecutar varias instrucciones dentro de una **transacción** (propiedades ACID)
  - Abrir transacción
    - Crear y ejecutar instrucciones
    - Procesar resultados
  - Cerrar transacción

Estableciendo la conexión con la BD

Establecimiento de conexión con la BD

- **Registrar un controlador**
  - Los DriverManager se encargan de gestionar la conexión y todas las comunicaciones con la BD
  - Necesitan conocer los controladores específicos para las BD que se vayan a utilizar
  - Registro de un controlador para MySQL
  - Utilizar el controlador MySQL Connector/J
  - Disponible en: http://dev.mysql.com/downloads/connector/j
    - Cómo instalarlo: http://dev.mysql.com/doc/refman/5.7/en/connector-j-installing.html
    - El controlador se puede registrar con el Class loader de Java
    - La clase a cargar viene dada en la documentación del controlador

```
try {
      Class.forName("com.mysql.jdbc.Driver").n
      ewInstance();
    } catch
(Exception ex)
          {
// Tratar el
       error
}
```

Estableciendo conexión con la BD

- Los controladores se identifican con un **URL JDBC** de la forma
- **`jdbc:subprotocolo:localizadorBD`**
  - El subprotocolo indica el tipo de base de datos específico
  - El localizador permite referenciar de forma única una BD
    - Host y opcionalmente puerto
    - Nombre de la base de datos
- La conexión a la BD se hace con el método **getConnection**()
  - public static Connection getConnection(String url)
  - public static Connection getConnection(String url, String user, String password)
  - public static Connection getConnection(String url, Properties info)
    - Todos pueden lanzar la excepción SQLException

```
try {
            conexion = DriverManager.getConnection(  "jdbc:mysql://localhost/tienda","pruebas", "pruebas");
} catch (SQLException ex) {
      // Tratar el error
}
```

Crear y ejecutar operaciones en la BD

- **Statement**
  - Encapsula las instrucciones SQL a la BD
  - Se crea a partir de la conexión

```
instruccion = conexion.createStatement();
```

  - Métodos
    - **executeQuery**(String sql)
      - Ejecución de consultas: SELECT
      - Devuelve un objeto ResultSet
    - **executeUpdate**(String sql)
      - Modificaciones en la BD: INSERT, UPDATE, DELETE
      - Devuelve el número de columnas afectadas
    - **execute**(String sql)
      - Ejecución de instrucciones que pueden devolver varios conjuntos de resultados
      - Requiere usar luego getResultSet() o getUpdateCount() para recuperar los resultados, y getMoreResults() para ver los siguientes resultados

Crear y ejecutar operaciones en la BD

- **ResultSet**
  - Encapsula el conjunto de resultados
  - Para obtener el valor de cada campo hay que usar el método getX("campo") correspondiente al tipo del valor SQL:
    - getInt ← INTEGER
    - getLong ← BIG INT
    - getFloat ← REAL
    - getDouble ← FLOAT
    - getBignum ← DECIMAL
    - getBoolean ← BIT
    - getString ← VARCHAR
    - getString ← CHAR
    - getDate ← DATE
    - getTime ← TIME
    - getTimesstamp ← TIME STAMP
    - getObject ← cualquier otro tipo
  - Para pasar al siguiente registro se usa el método **next**()
    - Devuelve false cuando no hay más registros

Crear y ejecutar operaciones en la BD

- Ejemplo

```
try {
     Statement instruccion = conexion.createStatement();

     String query = "SELECT * FROM clientes WHERE nombre LIKE \"Empresa%\"";
     ResultSet resultados = instruccion.executeQuery(query);

     System.out.println("Listado de clientes: ");
     while (resultados.next()) {
               System.out.println("Cliente "+resultados.getString("nif")
                         +", Nombre: "+resultados.getString("nombre")
                         +", Teléfono: " +resultados.getString("telefono") );
     }
} catch (Exception ex) {
     e.printStackTrace();
}
```

Crear y ejecutar operaciones en la BD

- **ResultSet**
  - Por defecto solo se puede recorrer hacia delante
  - Se pueden prever otras formas de utilizarlo al crear el objeto Statement:

    `createStatement(int resultSetType, int resultSetConcurrency)`

    - `resultSetType`:
      - `TYPE_FORWARD_ONLY`: sólo hacia delante con next()
      - `TYPE_SCROLL_INSENSITIVE`: métodos de posicionamiento habilitados
      - `TYPE_SCROLL_SENSITIVE`: métodos de posicionamiento habilitados pero sensible a las operaciones que se puedan hacer a los datos del ResultSet
        - Movimiento hacia atrás: afterLast(), previous()
        - Posicionamiento absoluto: first(), last(), absolute(numFila)
        - Posicionamiento relativo: relative(num)
  - Recupera fila actual: getRow()
    - `resultSetConcurrency`
      - `ResultSet.CONCUR_READ_ONLY`: El objeto ResultSet no se puede modificar
      - `ResultSet.CONCUR_UPDATABLE`: El objeto ResultSet se puede modificar

Instrucciones preparadas

- **PreparedStatement**
  - Cuando se van a ejecutar instrucciones repetidamente, se puede precompilar en la BD y ganar eficiencia
  - Primero se define el modelo de instrucción preparada

```
PreparedStatement ps =
conexion.prepareStatement("INSERT INTO clientes VALUES (?,?,?,?) ");
```

    - Cada parámetro se representan con el símbolo de interrogación ?
  - Luego se puede utilizar repetidamente simplemente indicando los parámetros con métodos setX(posición, valor)
    - El método depende del tipo de parámetro
    - La posición comienza en 1
    - executeUpdate devolverá la cantidad de elementos insertados

```
ps.setString(1, nif);
ps.setString(2, nombre);
ps.setString(3, direccion);
ps.setString(4, email);
if (ps.executeUpdate()!=1)
     throw new Exception("Error en la Inserción");
```

Excepciones

- **SQLException**
  - Es obligatorio capturar estas excepciones
  - Se puede obtener información adicional sobre el error
    - getMessage()
      - Mensaje de error de la excepción
    - getSQLState()
      - Texto de SQLstate según la convención X/Open o SQL:2003
    - getErrorCode()
      - Código de error (entero) específico del vendedor
  - Hay muchas subclases: BatchUpdateException, RowSetWarning, SerialException, SQLClientInfoException, SQLNonTransientException, SQLRecoverableException, SQLTransientException, SQLWarning, SyncFactoryException, SyncProviderException

Liberar recursos

- Normalmente en la cláusula **finally** para asegurar que se ejecuta
  - Usando el método **close()**

```
finally {
     try {
       if (resultados != null) {  // liberar los ResultSet
                resultados.close();
       }

       if (instruccion != null) {  // liberar los Statement
                instruccion.close();
       }

       if (conexion != null) {     // liberar la conexión a la BD
                conexion.close();
       }
     } catch (Exception e) {
       e.printStackTrace();
     }
}
```