Bases de datos con JDBC

Sesión 3: Información de la BD y optimización

Índice

- Obtención de información propia de la BD
- Optimización de sentencias
- Llamadas a procedimientos

Información propia de la BD

- Podemos obtener información de la BD que estamos consultando (Metadatos)
- Esta información puede ser el nombre de la base de datos, el nombre del driver con el que estamos accediendo, los nombres de las tablas, de los campos, etc.
- Disponemos de una clase para acceder a los metadatos: *DatabaseMetaData*
- Los datos se obtienen de la conexión
  - DatabaseMetaData dbmd = con.getMetaData();

Métodos de la clase *DatabaseMetaData*

- Existen más de cien métodos para obtener toda la información de la BD
- Veremos los más importantes

getDatabaseProductName	**Devuelve el nombre de la BD**
getDatabaseProductVersion	**Ídem versión**
getDriverName	**Ídem nombre del driver**
supportsResultSetType(int)	**Pasándole por parámetro algunos de los tipos de ResultSet (TYPE_FORWARD_ONLY, TYPE_SCROLL_SENSITIVE, etc.), nos devuelve si la BD los soporta**

Información de las tablas

- Haremos uso de esta llamada:
  - String[] tipos = {"TABLE"};
    ResultSet resul = dbmd.getTables(null,null,null,tipos);
- Los parámetros de *getTables* son de tipo *String* y especifican:
  - El catálogo del cual queremos obtener la información
  - Ídem el esquema
  - El nombre de la tabla. *null* obtiene todas. Podemos utilizar comodines “Nom%” (todas las tablas que empiecen por Nom
  - El tipo: TABLE, SYSTEM TABLE, VIEW, …
- Como vemos, la llamada a este método devuelve un *ResultSet* el cual podemos consultar como antes

Esquema para obtener los datos de las tablas

```
while (resul.next()) {
    String nombreTabla = resul.getString("TABLE_NAME");
    ResultSet columnas = dbmd.getColumns(null,null,nombreTabla,null);
    while (columnas.next()) {
        String nombreColumna = columnas.getString("COLUMN_NAME");
        int tipoDato = columnas.getInt("DATA_TYPE");
        String nombreTipo = columnas.getString("TYPE_NAME");
        if (tipoDato==java.sql.Types.CHAR ||
            tipoDato==java.sql.Types.VARCHAR)
                int tamanyo = columnas.getString("COLUMN_SIZE");
        String nulo = columnas.getString("IS_NULLABLE");
        if (nulo.equalsIgnoreCase("no")) System.out.println("NOT NULL");
    }
    ResultSet
clavesPrimarias=dbmd.getPrimaryKeys(null,null,nombreTabla);
    while (clavesPrimarias.next())
        String nombreClave = clavesPrimarias.getString("COLUMN_NAME");
}
```

Metadatos de una consulta

- También podemos obtener la información de una consulta
  - ResultSet resul = stmt.executeQuery("SELECT * FROM ALUMNOS");
    ResultSetMetaData rsmd = resul.getMetaData();
    int columnas = rsmd.getColumnCount();
    for (int i=1; i<=columnas; i++) {
        System.out.println("Nombre tabla="+rsmd.getTableName(i));
        System.out.println("Nombre columna="+rsmd.getColumnName(i));
        System.out.println("Tipo de dato="+rsmd.getTypeName(i));
      System.out.println("Autoincremento=“+rsmd.isAutoIncrement(i));
    }

Sentencias preparadas

- Cuando ejecutamos una sentencia SQL, esta se compila y se envía al SGBD
- Si la vamos a ejecutar muchas veces, es preferible dejar la sentencia preparada (precompilada) para aumentar la eficiencia
- Disponemos de una clase, *PreparedStatement*, que nos permite realizar la precompilación
  - PreparedStatement ps = con.prepareStatement(
    "UPDATE FROM alumnos
    SET sexo = 'H'
    WHERE exp>1200 AND exp<1300");

Sentencias preparadas

- Cuando creamos el objeto ya se le pasa la sentencia a ejecutar. En ese momento la precompila
- Esta clase nos va a permitir parametrizar la sentencia:
  - PreparedStatement ps = con.prepareStatement(
    "UPDATE FROM alumnos
    SET sexo = 'H'
    WHERE exp > ? AND exp < ?");
- Los ‘?’ son parámetros que podremos asignar más tarde

Asignación de parámetros

- Para dar valor a los parámetros usaremos los métodos *setXXXX* al igual que anteriormente
- Estos métodos reciben dos parámetros, el número de orden del parámetro y el valor a asignar
  - ps.setInt(1,1200);
    ps.setInt(2,1300);
- Como la sentencia era de actualización, para ejecutarla llamaremos al método *executeUpdate*, pero ahora sin parámetros
  - int n = ps.executeUpdate();
- Si la sentencia fuera de consulta usaríamos *executeQuery* y también disponemos de *execute*

Restricciones en las sentencias preparadas

- Los parámetros sólo pueden ser de datos, es decir, no podemos usar como parámetros el nombre de la tabla, por ejemplo
- Una vez asignados los parámetros, estos no desaparecen
- Podemos llamar a *clearParameters*

Llamadas a procedimientos

- En las empresas que ya disponen de BD es muy común encontrar procedimientos
- Un procedimiento es una unidad de código que contiene un conjunto de sentencias SQL u otro lenguaje propio de la BD. No son estándar, por lo que cambia mucho de un SGBD a otro
- Los procedimientos (algunas veces llamados funciones) aumentan la eficiencia del acceso y actualización de la BD
- Vamos a ver cómo podemos invocar estos procedimientos

Ejemplo de procedimiento

- Este ejemplo declara un procedimiento que contiene parámetros de entrada
  - String procedure = "CREATE PROCEDURE EFECTUAR_VENTA
    (IN cod_cliente INT, IN cod_producto INT, IN cantidad INT)
    LANGUAGE SQL
    BEGIN
    INSERT INTO ventas(cliente, producto, cant)
    VALUES(cod_cliente, cod_producto, cantidad);
    UPDATE productos SET stock = stock - cantidad
    WHERE producto = cod_producto;
    END";
    stmt.executeUpdate(procedure);

Otro ejemplo

- En este caso no recibe parámetros de entrada, pero sí que genera salida
  - String procedure = "CREATE PROCEDURE VER_VENTAS_CLIENTE
    AS SELECT cliente, sum(precio) FROM ventas, productos
        WHERE ventas.producto = productos.producto
        GROUP BY cliente";
    stmt.executeUpdate(procedure);
- Podremos tener parámetros de entrada y salida, incluso varios ResultSet de salida
- Esto último depende mucho de si lo soporta la BD y/o el driver de acceso

Llamada a un procedimiento

- Utilizaremos la clase *CallableStatement*
- Podemos llamar a un procedimiento
  - CallableStatement cs = con.prepareCall("{call VER_VENTAS_CLIENTE}");
    ResultSet rs = cs.executeQuery();
- Si el procedimiento acepta parámetros, asignarlos como hacíamos con las sentencias preparadas
  - CallableStatement cs = con.prepareCall("{call EFECTUAR_VENTA[?, ?, ?]}");
    cs.setInt(1, 112);
    cs.setInt(2, 3380);
    cs.setInt(3, 1);
    cs.executeUpdate();

Obtención de los parámetros de salida

- Para obtener un parámetro de salida, primero debemos registrarlo
  - cs.registerOutParameter(2, java.sql.Types.STRING);
- Si tenemos parámetros de entrada/salida, debemos registrarlo como de salida y asignarle un valor
- Para obtener definitivamente el valor, llamamos a los métodos *getXXX*, después de ejecutar la llamada
- Los parámetros de entrada de un procedimiento deben asignarse antes de la llamada. Sino es así se producirá una excepción

¿Preguntas...?