Chuleta de JDBC

Rodrigo García Carmona (v1.1.2)

Esta guía está pensada para SQLite.

Antes de empezar, no olvides que necesitas tener el jar *sqlite-jdbc* en tu workspace, así como tenerlo añadido como biblioteca al proyecto.

Crear una conexión

Antes de trabajar con JDBC tenemos que crear una conexión a la base de datos:

```
Class.forName("org.sqlite.JDBC");
Connection c = DriverManager.getConnection("jdbc:sqlite:DATABASELOCATION");
```

Activar soporte a Foreign Keys en SQLite

Para activar el soporte a las restricciones de foreign keys en SQL es necesario ejecutar la siguiente sentencia SQL para cada conexión:

```
c.createStatement().execute("PRAGMA foreign_keys=ON");
```

Cerrar una conexión

Y, tras acabar, debemos cerrarla:

```
c.close();
```

Usar Statements

Los Statements se pueden utilizar para hacer actualizaciones, como con este INSERT:

```
Statement stmt = c.createStatement();
String sql = "INSERT INTO employees (name, age, address) "
             + "VALUES ('" + name + "', " + age + ", '" + address + "')";
stmt.executeUpdate(sql);
```

O para consultas, como con este SELECT:

```
Statement stmt = c.createStatement();
String sql = "SELECT * FROM employees";
```

```
ResultSet rs = stmt.executeQuery(sql);
```

Cuando hacemos consultas, los resultados son devueltos en forma de Result Set.

Debemos acordarnos de cerrar el Statement:

```
stmt.close();
```

No se recomienda usar Statements para consultas. Es más, no pueden usarse en SQLite para consultas que tengan tipos de dato distintos a INTEGER, REAL o TEXT. En estos casos, además de en aquellos en los que la seguridad nos preocupe, debemos usar **Prepared Statements**.

Usar Prepared Statements

Los Prepared Statements se pueden utilizar para hacer actualizaciones, como con este INSERT:

```
String sql = "INSERT INTO employees (name, age, salary, birthDate, photo) "
             + "VALUES (?,?,?,?,?)";
PreparedStatement prep = c.prepareStatement(sql);
prep.setString(1, "Bob");
prep.setInt(2, 20);
prep.setDouble(3, 35000.00);
prep.setDate(4, anSqlDateObject);
prep.setBytes(5, aBytesArray);
prep.executeUpdate();
```

O para consultas, como con este SELECT:

```
String sql = "SELECT * FROM employees WHERE name LIKE ?";
PreparedStatement prep = c.prepareStatement(sql);
prep.setString(1, "%Bob%");
ResultSet rs = prep.executeQuery();
```

Una vez más, cuando hacemos consultas, los resultados son devueltos en forma de Result Set.

Como siempre, hay que acordarse de cerrar el Statement:

```
prep.close();
```

Procesar Result Sets

Cuando hacemos consultas, ya sea con Statements o con Prepared Statements, los resultados vienen dados en forma de Result Set. Podemos iterar sobre un ResultSet para acceder a sus contenidos:

```
while (rs.next()) {
    int id = rs.getInt("id");
    String name = rs.getString("name");
    int age = rs.getInt("age");
    String address = rs.getString("address");
```

```
    double salary = rs.getDouble("salary");
    Employee employee = new Employee(id, name, age, address, salary);
}
```

No debemos olvidarnos de cerrar el Result Set:

```
rs.close();
```

Cuando sacamos un BLOB de la base de datos, debemos tener en mente que lo hacemos en forma de Stream, por lo que podríamos querer transformar dicho Stream en un array de bytes:

```
InputStream blobStream = rs.getBinaryStream("photo");
byte[] blobArray = new byte[blobStream.available()];
blobStream.read(blobArray);
```

Commits Manuales

Por defecto, JDBC aplica inmediatamente los cambios efectuados sobre la base de datos (hace *commit*), pero podemos modificar este comportamiento para controlar de forma manual cuándo se hacen los commits:

```
c.setAutoCommit(false);
```

Tras hacer esto, debemos escribir lo siguiente para hacer un commit:

```
c.commit();
```

También podemos deshacer todos los cambios efectuados desde el último commit:

```
c.rollback();
```