JDBC Cheatsheet

Rodrigo García Carmona (v1.2.1)

This guide has been created with SQLite in mind.

First of all, remember that you need to have the *sqlite-jdbc* jar in your workspace and added as a library to the project.

Create a Connection

Before doing any work, we must create a connection to the database:

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

Replace *DATABASELOCATION* with the (preferably relative) path to your database file.

Activate Foreign Key Support in SQLite

To enable support for foreign key constraints in SQLite, the following SQL sentence must be executed in each connection:

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

Close a Connection

After finishing working with the database, we must close its connection:

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

Using Statements

Statements can be used to make updates, like this INSERT:

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

Or queries, like this SELECT:

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

When making queries, the results are returned as a Result Set.

We must always remember to close the Statement:

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

Important note: Statements **are not recommended** for queries, and they can't be used at all for queries that include anything other than INTEGERs, REALs or TEXTs. In this situations, and when we're concerned about security, we use **Prepared Statements**.

Using Prepared Statements

Prepared Statements can be used to make updates, like this INSERT:

```
String sql = "INSERT INTO employees (name, phone, salary, dob, photo) "
           + "VALUES (?,?,?,?,?)";
PreparedStatement prep = c.prepareStatement(sql);
prep.setString(1, "Bob");
prep.setInt(2, 666666666);
```

```
prep.setDouble(3, 35000.00);
prep.setDate(4, anSqlDateObject);
prep.setBytes(5, aBytesArray);
prep.executeUpdate();
```

Or queries, like this SELECT:

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

Again, when making queries, the results are returned as a Result Set.

As always, rememeber to close the Prepared Statement:

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

Processing Result Sets

When making queries, either with Statements or Prepared Statements, the results are returned as a Result Set. We can iterate over a Result Set and access its contents:

```
while (rs.next()) {
    int id = rs.getInt("id");
    String name = rs.getString("name");
    int age = rs.getInt("phone");
    String address = rs.getString("address");
    double salary = rs.getDouble("salary");
    java.sql.Date date = rs.getDate("dob");
    Employee employee = new Employee(id, name, age, address, salary);
}
```

Again, we mustn't forget to close the Result Set:

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

When retrieving a BLOB, we can either get a Stream, which we might want to convert into a byte array:

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

Or a byte array (this might not be supported by the JDBC connector for your database):

```
byte[] blobArray = rs.getBytes("photo");
```

Also, it's important to point that ResultSet *getXX* methods don't support aliases nor column names in the "table_name.column_name" form. Therefore, if there's a JOIN with several columns with the same name, the column numbers must be used instead:

```
while (rs.next()) {
    int id = rs.getInt(1);
    String name = rs.getString(2);
    int phone = rs.getInt(3);
    String address = rs.getString(4);
    double salary = rs.getDouble(5);
    java.sql.Date dob = rs.getDate(6);
    Employee employee = new Employee(id, name, phone, address, salary, dob);
}
```

Manual Commits

By default, JDBC will commit to the database any change in the moment we make it, but we can change this behaviour to manually control when commits are made:

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

After doing that, we must write the following to make a commit:

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

Or we can undo every change we made since the last commit:

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

Getting the Primary Key of the Last Inserted Row

After inserting a new row in a table without specifying the primary key, you can get the primary key that the SQLite database has assigned by running this SELECT query:

```
String query = "SELECT last_insert_rowid() AS lastId";
PreparedStatement p = c.prepareStatement(query);
ResultSet rs = p.executeQuery();
Integer lastId = rs.getInt("lastId");
```