Java Database Connectivity

Joseph Hallett

February 13, 2023

University of
BRISTOL

What's this about?

Previously when using SQL we've used it as its own thing...

- ▶ Running queries in SQL directly
- ▶ Presenting responses as tables

In the *real world* we rarely want to access a database in its own right

- ▶ Rather it is used within a programming language as part of a program

Different languages have different APIs for different databases...

- ▶ ...but Java has the JDBC for almost all of them

JDBC

- Library is in `java.sql` and `javax.sql` packages
- Wraps all of a databases functionality into something that looks a lot like *Oracle SQL.*
- Supports *prepared statements* (you want to use these)

What does it look like?

```
try (final Connection conn = DriverManager.getConnection("jdbc:
sqlite:database.db")) {                  try-with-resources
        database.db        SQLite              DriverManager.
getConnection                                         jdbc:sqlite:
database.db         URL      JDBC           SQLite
                        database.db
```

```
conn.createStatement().executeQuery("CREATE TABLE users(username
TEXT PRIMARY KEY, password TEXT)");
Statement            SQL                   users
    username                        password
      executeQuery                        CREATE TABLE      DDL
                              execute executeUpdate
```

```
import java.sql.*;
```

```
executeQuery
try (final Connection conn = DriverManager.getConnection("jdbc:sqlite:database.db")) {
    conn.createStatement()
            .executeQuery("CREATE TABLE users(username TEXT PRIMARY KEY, password TEXT)");
} catch (final SQLException err) {
    System.out.println(err);
}

java.sql.SQLException: No suitable driver found for jdbc:sqlite:database.db
```

```
} catch (final SQLException err) {
SQLException       System.out.println(err);            SQLException
             java.sql.SQLException: No suitable driver found for jdbc:sqlite:
database.db                                      JDBC              jdbc:
sqlite                                   SQLite JDBC
                                    SQLite JDBC
```

And when you've found a suitable driver and added it to your `CLASSPATH`...

```java
import java.sql.*;

try (final Connection conn = DriverManager.getConnection("jdbc:sqlite:database.db")) {
    conn.createStatement()
        .executeQuery("CREATE TABLE users(username TEXT PRIMARY KEY, password TEXT)");
} catch (final SQLException err) {
    System.out.println(err);
}

org.sqlite.SQLiteException: [SQLITE_ERROR] SQL error or missing database (table users already exists)
```

Lets add some suitable users...

```java
import java.sql.*;
import java.util.*;

final var users = new HashMap<String, String>();
users.put("Joseph", "password");
users.put("Matt", "password1");
users.put("Partha", "12345");

try (final Connection conn = DriverManager.getConnection("jdbc:sqlite:database.db")) {
    conn.createStatement().executeUpdate("DELETE FROM users");
    final var statement = conn.prepareStatement("INSERT INTO users VALUES(?, ?)");
    for (final var user : users.keySet()) {
        statement.setString(1, user);
        statement.setString(2, users.get(user));
        statement.executeUpdate();
    }
} catch (final SQLException err) {
    System.out.println(err);
}
```

DELETE FROM users   users

And list them back out...

```java
import java.sql.*;
import java.util.*;

System.out.println("|User | Password");
try (final Connection conn = DriverManager.getConnection("jdbc:sqlite:database.db")) {
    final var results = conn.createStatement()
        .executeQuery("SELECT * FROM users");
    while (results.next())
        System.out.println("| "+results.getString(1)
                          +" | "+results.getString(2));
} catch (final SQLException err) {
    System.out.println(err);
}
```

User	Password
Matt	password1
Joseph	password
Partha	12345

Why not this...

When adding all the users we used a `PreparedStatement` to add all the users.

```java
final var statement = conn.prepareStatement("INSERT INTO users VALUES(?, ?)");
for (final var user : users.keySet()) {
    statement.setString(1, user);
    statement.setString(2, users.get(user));
    statement.executeUpdate();
}
```

Wouldn't this be easier?

```java
for (final var user : users.keySet())
    conn.createStatement()
        .executeUpdate("INSERT INTO users "+"VALUES ('"+user+"', '"+users.get(user)+"')");
```

SQL Injection

This leads to a *horrible* vulnerability called an *injection attack*

- You can do something similar with shellscript too ;-)
- Search for *Shellshock vulnerability* if you're interested...

What a prepared statement does is ensure that the things you add are what you say they are
Suppose you do the something similar for the login code:

```
SELECT username FROM users
WHERE username = "Joseph"
AND password = "password";
```

username
Joseph

Suppose the username and password are taken from a website login form...

- What happens if I try and login with a password of:

```
" OR 1 OR password = "heheh
```

Bad things

With a prepared statement:

```
SELECT username FROM users
WHERE username = "Joseph"
AND password = """" OR 1 OR password = ""heheh";
```

Without a prepared statement:

```
SELECT username FROM users
WHERE username = "Joseph"
AND password = "" OR 1 OR password = "heheh";
```

username
Matt
Joseph
Partha

ALWAYS USE PREPARED STATEMENTS
The compiler will even spew warnings and errors about this nowadays...

▶ Or *findbugs* will...

Transactions

Another *cool* thing that JDBC makes easy are *transactions*...
Suppose you want to do a bunch of additions and updates to a database...

- What happens if something goes wrong *in the middle*?

You *could* go and manually *roll back* all the new data you added and changes you made...

- Sounds tedious
- Lets automate it!

Transaction workflow

1. Start a new transaction
2. Do your work
3. Commit to it when done
4. Rollback if an error occurs

And in Java please?

```java
import java.sql.*;
import java.util.*;

try (final Connection conn = DriverManager.getConnection("jdbc:sqlite:database.db")) {
    conn.setAutoCommit(false);
    final var save = conn.setSavepoint();
    try {
        conn.createStatement() .executeQuery("INSERT INTO users VALUES ('Alice', 'pa55w0rd')");
        conn.createStatement() .executeQuery("INSERT INTO users VALUES ('Bob', 'Pa55w0Rd7')");
        if (true) throw new Exception("Whoops!");
        conn.createStatement() .executeQuery("INSERT INTO users VALUES ('Eve', 'backd00r')");
        conn.commit();
    } catch (final Exception err) {
        conn.rollback(save);
    } finally {
        conn.setAutoCommit(true);
    }
} catch (final SQLException err) {
    System.out.println(err);
}
```
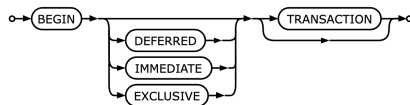
Now if we query users...

```
SELECT * FROM users;
```

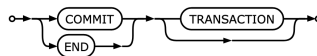| username	password
Matt	password1
Joseph	password
Partha	12345

Our table *remains* unaltered... the whole transaction was *rolled back*.

(Oh, and BTW SQLite also can do transactions in SQL)
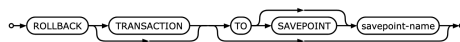

begin-stmt: [hide]

commit-stmt: [hide]

rollback-stmt: [hide]

Back to Entity Relationship Diagrams...

Right back at the start of the database run of videos we were doodling diagrams instead of building databases

- A Java class looks *an awful lot* like an entity

Wouldn't it be neat if we could take a class and get the database importing and saving all handled for us?

Hibernate!

https://hibernate.org
Builds *on top* of JDBC to do just that!

- ▶ Annotate your classes
- ▶ Write a bunch of XML to tell it about your database format
- ▶ Magic and a *slightly* higher-level query language

We'll play with it in the lab...

Conclusions

JDBC lets you access SQL from Java

- ▶ Make sure you load the right driver
- ▶ Catch SQLExceptions
- ▶ Use prepared statements and transactions to prevent errors
- ▶ And an ORM like *Hibernate* if you like.

IMPORTANT NOTE

Please don't actually implement password storage like we did in the lecture...

- ▶ Go speak to someone in the cyber or crypto groups *first*...
- ▶ Or read NIST 800-63 first

I will write papers about you if you do **;-)**

> Joseph Hallett, Nikhil Patnaik, Benjamin Shreeve and Awais Rashid. *"Do this! Do that!, And nothing will happen" Do specifications lead to securely stored passwords? 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). 2021.*