


CS50's Introduction to Databases with SQL

OpenCourseWare

Donate  (<https://cs50.harvard.edu/donate>)

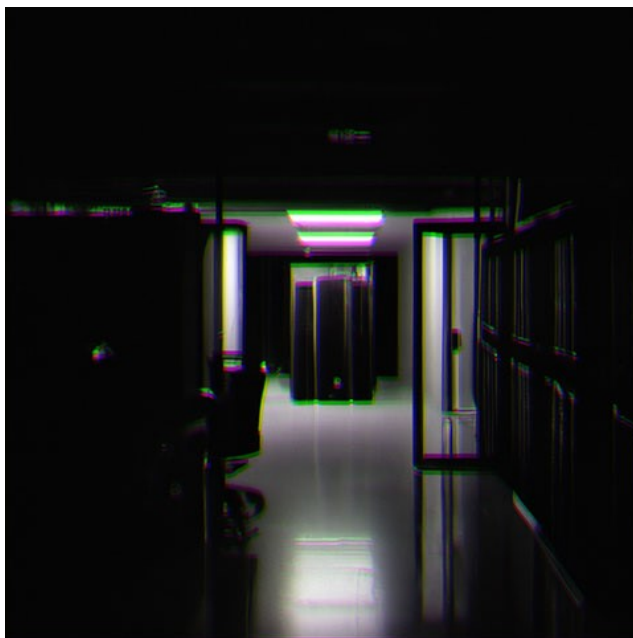
Carter Zenke (<https://carterzenke.me>)
carter@cs50.harvard.edu

 (<https://github.com/carterzenke>)  (<https://www.linkedin.com/in/carterzenke/>)

David J. Malan (<https://cs.harvard.edu/malan/>)
malan@harvard.edu

 (<https://www.facebook.com/dmalan>)  (<https://github.com/dmalan>)  (<https://www.instagram.com/davidjmalan/>)  (<https://www.linkedin.com/in/malan/>)  (<https://www.reddit.com/user/davidjmalan>)  (<https://www.threads.net/@davidjmalan>)  (<https://twitter.com/davidjmalan>)

Don't Panic!



"An empty server room in an office at night, in the style of a detective noir," generated by [DALL·E 2](https://openai.com/dall-e-2) (<https://openai.com/dall-e-2>)

Problem to Solve

You're a trained "pentester." After your success in an [earlier operation](#), a new company has hired you to perform a [penetration test \(https://en.wikipedia.org/wiki/Penetration_test\)](https://en.wikipedia.org/wiki/Penetration_test) and report the vulnerabilities in their data system. This time, you suspect you can do better by writing a program in Java that automates your hack.

To succeed in this covert operation, you'll need to...

- **Connect**, via Java, to a SQLite database.
- **Alter**, within your Java program, the administrator's password.

If you don't have experience with Java, not to worry! This problem will walk you through each step along the way.

Distribution Code

For this problem, you'll need access to a database, a Java file, and a set of SQL statements to reset the database in case your hack fails the first time (not to worry if it does!).

▼ Download the distribution code

Log into [cs50.dev \(https://cs50.dev/\)](https://cs50.dev/), click on your terminal window, and execute `cd` by itself. You should find that your terminal window's prompt resembles the below:

```
$
```

Next execute

```
wget https://cdn.cs50.net/sql/2023/x/psets/6/dont-panic-java.zip
```

in order to download a ZIP called `dont-panic-java.zip` into your codespace.

Then execute

```
unzip dont-panic-java.zip
```

to create a folder called `dont-panic-java`. You no longer need the ZIP file, so you can execute

```
rm dont-panic-java.zip
```

and respond with "y" followed by Enter at the prompt to remove the ZIP file you downloaded.

Now type

```
cd dont-panic-java
```

followed by Enter to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

```
dont-panic-java/ $
```

If all was successful, you should execute

```
ls
```

and see a database named `dont-panic.db` alongside a `Hack.java` file, a `reset.sql` file, and a `sqlite-jdbc-3.43.0.0.jar` file. If not, retrace your steps and see if you can determine where you went wrong!

Specification

In `Hack.java`, write a Java program to achieve the following:

- **Connect**, via Java, to a SQLite database.
- **Alter**, within your Java program, the administrator's password.

When your program in `Hack.java` is run on a new instance of the database, it should produce the above results.

Clock's ticking!

Walkthrough

If you're new to Java (or to connecting Java with SQL!) this walkthrough will guide you through each of this problem's steps.

Java

When you download the distribution code for this problem, you should notice a file named `Hack.java`. You can tell this program is a Java program because it ends with `.java`. The `.java` extension identifies files as Java files much like how the `.sql` extension identifies files as a set of SQL statements.

At first, `Hack.java` should only include the following:

```
public class Hack {  
    public static void main(String[] args) throws Exception {  
        System.out.println("Hacked!");  
    }  
}
```

To run this Java program, ensure that—when you type `ls`—you see `Hack.java` among the files in your current directory. Then, execute the below in your terminal:

```
javac Hack.java
```

`javac` *compiles* the program in `Hack.java` to a file named `Hack.class`. To compile a program means to translate a program from one language to another—in this case, from Java source code (the code you wrote yourself!) to Java “bytecode” (the code understandable to the `java` program on your computer). You should then see a new file named `Hack.class`. Execute the following to run `Hack.class`, the compiled program:

```
java Hack
```

If all's gone well, you should see “Hacked!” printed to your terminal. Not quite a hack, but you're on your way!

Java's SQL Package

Imagine your Java program wants to talk to a SQLite database, a bit like how you might call up a friend. But—unlike when you call your friend—there's a problem: your Java program and the SQLite database don't speak the same language.

For that reason, the process of connecting your Java program to a database is all about ensuring your Java program and SQLite have some intermediary that can translate statements between them. That's where Java's *SQL package* comes in! A package is a collection of code that someone else has written to solve a problem (and, importantly, which you can use in your own Java program!).

Java's standard SQL package provides you with a few tools to translate statements between your Java program and SQLite: `DriverManager`, `Connection`, and `Statement`. Each has a role to play in your program:

- `DriverManager` begins a database connection. It sets up a particular *driver* (that is, an intermediary program) to translate statements between your Java program and your database.
- A `Connection` is what is created by `DriverManager`. It represents a connection to a database and, as such, all SQL statements run through it.
- A `Statement` is the messenger that relays SQL statements through the database

connection.

To include these tools in your program, add the following `import` statements:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class Hack {
    public static void main(String[] args) throws Exception {
        System.out.println("Hacked!");
    }
}
```

Try recompiling your Java program, as with `javac Hack.java`, and running it, as with `java Hack`. Nothing much should change, but that's actually a good sign!

Connecting to a Database

As mentioned before, you can use `DriverManager` to establish a connection to `dont-panic.db`. To do so, replace `System.out.println("Hacked!");` with the below:

```
Connection sqliteConnection = DriverManager.getConnection("jdbc:sqlite:dont-panic.db");
```

This line of Java code is composed of a few pieces:

- `Connection sqliteConnection`, which initializes a `Connection` named `sqliteConnection`. Henceforth in your program, you can use `sqliteConnection` to reference the connection between your Java program and your database.
- `DriverManager.getConnection(...)`, which establishes a connection the database given as input within the parentheses.
- `jdbc:sqlite:dont-panic.db`, which is a URL (similar to a website URL!) that identifies the database to which to connect (`dont-panic.db`) and the SQL dialect to which to translate (`sqlite`, in this case, as opposed to `mysql` or `postgres`).

Your program should now look as follows:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class Hack {
    public static void main(String[] args) throws Exception {
        Connection sqliteConnection = DriverManager.getConnection("jdbc:sqlite:do
    }
}
```

However, if you run `javac Hack.java` and `java Hack` to compile and run your updated

program, you might encounter an error:

```
Exception in thread "main" java.sql.SQLException: No suitable driver found for jdbc:sqlite:dont-panic.db
    at java.sql/java.sql.DriverManager.getConnection(DriverManager.java:708)
    at java.sql/java.sql.DriverManager.getConnection(DriverManager.java:253)
    at Hack.main(Hack.java:7)
```

You might notice in particular that there is “No suitable driver found for `jdbc:sqlite:dont-panic.db`.” In other words, Java is telling you it can't find the program to translate between Java and SQLite!

To fix this issue, you'll need to add the Java SQLite driver to your project. Once you do, your Java program and the SQLite database can start talking to each other.

Compile your program as usual with `javac Hack.java`. But instead of running `java Hack`, execute the following:

```
java -cp .:sqlite-jdbc-3.43.0.0.jar Hack
```

This command is composed of a few pieces:

- `-cp`, which stands for “Class Path.” It's a way of telling Java where to look for extra things your program needs, like the Java SQLite driver.
- `.`, which means “look in the current directory.” That's where Java finds your `Hack.class` file.
- `:`, which is similar to saying “and also look over here.” It's how you separate different places Java should search.
- `sqlite-jdbc-3.43.0.0.jar`, which is the name of the file that contains the SQLite driver.

Putting it all together, `java -cp .:sqlite-jdbc-3.43.0.0.jar Hack` tells Java to run the `Hack` class and, at the same time, use the current directory and the `sqlite-jdbc-3.43.0.0.jar` file to find any extra programs it might need.

Try running your program now. You might not see anything happen and, if so, that's a good sign!

Executing SQL Statements with Java

Once you've created a connection to a database, you can start executing SQL statements. As mentioned above, you can use `Statement` to relay SQL statements through your newly created database connection: a `Statement` is like a messenger between your Java program and the database to which you've connected.

Here's how you can set up and execute an `UPDATE` statement:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class Hack {
    public static void main(String[] args) throws Exception {
        Connection sqliteConnection = DriverManager.getConnection("jdbc:sqlite:do

        Statement sqliteStatement = sqliteConnection.createStatement();

        sqliteStatement.executeUpdate("""
            UPDATE "users"
            SET "password" = 'hacked!'
            WHERE "username" = 'admin';
        """);

        sqliteConnection.close();
    }
}
```

Notice that you first create a `Statement` named `sqliteStatement`. `sqliteStatement` is created by using a *method* of `sqliteConnection` called `createStatement`. A method is some piece of code that takes an input and produces some output: in this case, a `Statement`!

Next, you can use `sqliteStatement`'s `executeUpdate` method, which takes as input any SQL query that writes to the database. `executeUpdate` relays the statement to the database and runs it. Once the update is complete, you can close the connection to the database, using `sqliteConnection`'s `close` method.

After you re-compile and run your program, try opening `dont-panic.db` with `sqlite3`. When you view the administrator's password, you should find it is now "hacked!"

If at any point you'd like to reset `dont-panic.db` to its original state, recall that you can use `reset.sql`.

Prepared Statements

Imagine you wanted a user to determine the new administrative password *as your Java program runs*.

Recall from lecture that a prepared statement is a SQL query with placeholders for values that are inserted ("interpolated") later. Since you don't know what password your program's user will choose, the best you can do is set a placeholder for the password and allow your program to interpolate the user-chosen password later. Using a prepared statement, then, can help!

The `java.sql` package supports using prepared statements. First, replace importing `java.sql.Statement` with importing `java.sql.PreparedStatement` and write your query with a `?` as a placeholder:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class Hack {
    public static void main(String[] args) throws Exception {
        Connection sqliteConnection = DriverManager.getConnection("jdbc:sqlite:do

        String query = ""
            UPDATE "users"
            SET "password" = ?
            WHERE "username" = 'admin';
        """;

        sqliteConnection.close();
    }
}
```

Next, use the `prepareStatement` and `setString` methods to substitute some value for that placeholder.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class Hack {
    public static void main(String[] args) throws Exception {
        Connection sqliteConnection = DriverManager.getConnection("jdbc:sqlite:do

        String query = ""
            UPDATE "users"
            SET "password" = ?
            WHERE "username" = 'admin';
        """;
        PreparedStatement sqliteStatement = sqliteConnection.prepareStatement(que
        sqliteStatement.setString(1, "hacked!");

        sqliteStatement.executeUpdate();

        sqliteConnection.close();
    }
}
```

And finally, consider what you could do to get user input. While we won't dive into all the details of how to collect user input with Java, suffice to say that the following should work!

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class Hack {
    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the new password: ");
    }
}
```



```
String password = scanner.nextLine();

Connection sqliteConnection = DriverManager.getConnection("jdbc:sqlite:do

String query = ""
    UPDATE "users"
    SET "password" = ?
    WHERE "username" = 'admin';
"";
PreparedStatement sqliteStatement = sqliteConnection.prepareStatement(que
sqliteStatement.setString(1, password);

sqliteStatement.executeUpdate();

sqliteConnection.close();
scanner.close();
}
}
```

Now, try running your program and viewing the changes in `dont-panic.db`!

How to Test

Correctness

Execute the below to evaluate the correctness of your findings using `check50`

```
check50 cs50/problems/2023/sql/sentimental/dont-panic/java
```

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/problems/2023/sql/sentimental/dont-panic/java
```

