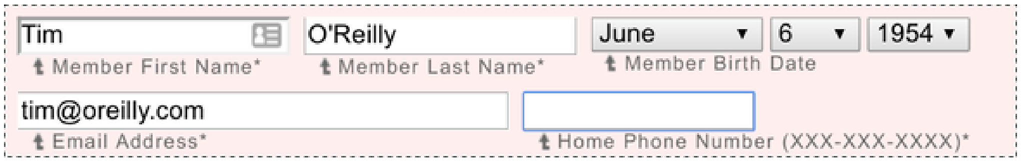
SQL Injection

SQL Injection is an exploit in which an attacker enters some carefully-coded data into form fields, in the hopes of getting it executed as SQL on your database.

This is also related to the more mundane problem of supporting special characters in form fields. For example, suppose a user named Tim O'Reilly registers for a user account on our site.



Tim	O'Reilly	June 6 1954
Member First Name*	Member Last Name*	Member Birth Date
tim@oreilly.com		
Email Address*	Home Phone Number (XXX-XXX-XXXX)*	

Figure 1: Tim O'Reilly registers for an account

This can cause a problem because that apostrophe (single quote) character is special in SQL – it is used to delimit strings. If we're not careful about how we construct and embed parameters into our SQL query, we're vulnerable to serious problems due to special characters.

```
// THIS IS BAD, DO NOT USE THIS CODE
connection.executeUpdate(
    "INSERT INTO user (first, last) VALUES ('" +
    firstName + "', '" +
    lastName + "')"
);
```

If the code above is used to process the registration form, for Tim, it will end up trying to execute this query:

```
INSERT INTO user (first, last) VALUES ('Tim', 'O'Reilly')   -- Error!
```

which contains a syntax error due to the extra single quote.

Of course, the harm could be taken much further. What happens when the school tries to add "Little Bobby Tables" (see cartoon) to their database?

```
INSERT INTO Student (name) VALUES ('Robert'); DROP TABLE Students; -- ');
                                    \_______________________________/
```

The underlined portion shows what the mom typed into the name field on the registration form. This is even worse than the Tim O'Reilly example: it produces completely valid SQL that deletes all our student data!

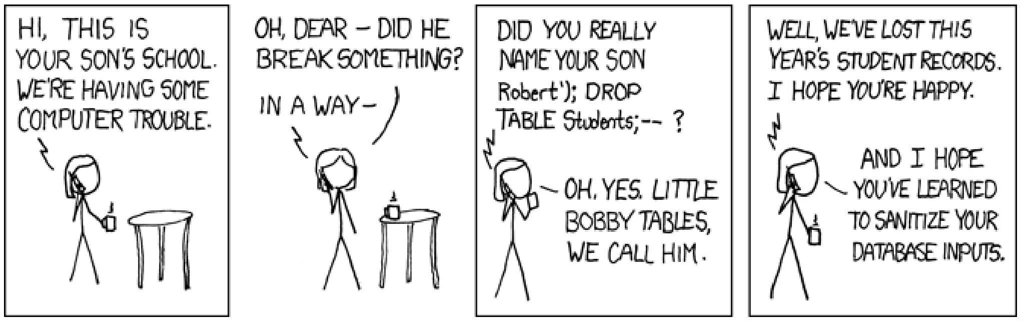

HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Figure 2: xkcd: Exploits of a Mom

To protect against SQL injection when using JDBC, we should always code queries with parameters as **prepared statements.** It looks like this:

```
PreparedStatement stm = conn.prepareStatement(
    "INSERT INTO user (first, last) VALUES (?, ?)"
);
```

We don't attempt to manipulate the SQL query using Java string concatenation. Instead, we just use the question mark character '?' at certain places in the query where parameters should be substituted.

Then we can use methods like `setInt`, `setString`, and so forth. The parameters indicated by the question marks are **numbered starting from 1,** which is somewhat unusual in programming.

```
stm.setString(1, firstName);   // Substitute first question mark
stm.setString(2, lastName);    // Substitute second question mark
```

The `PreparedStatement` object will take care of quoting the special characters as required by the database. To execute it, we then do:

```
stm.executeUpdate();
```