

# SQLITE - INJECTION

[http://www.tutorialspoint.com/sqlite/sqlite\\_injection.htm](http://www.tutorialspoint.com/sqlite/sqlite_injection.htm)

Copyright © tutorialspoint.com

If you take user input through a webpage and insert it into a SQLite database there's a chance that you have left yourself wide open for a security issue known as SQL Injection. This lesson will teach you how to help prevent this from happening and help you secure your scripts and SQLite statements.

Injection usually occurs when you ask a user for input, like their name, and instead of a name they give you a SQLite statement that you will unknowingly run on your database.

Never trust user provided data, process this data only after validation; as a rule, this is done by pattern matching. In the example below, the username is restricted to alphanumeric chars plus underscore and to a length between 8 and 20 chars - modify these rules as needed.

```
if (preg_match("/^\w{8,20}$/", $_GET['username'], $matches)){
    $db = new SQLiteDatabase('filename');
    $result = @$db->query("SELECT * FROM users WHERE username=$matches[0]");
}else{
    echo "username not accepted";
}
```

To demonstrate the problem, consider this excerpt:

```
$name = "Qadir'; DELETE FROM users;";
@$db->query("SELECT * FROM users WHERE username='{$name}'");
```

The function call is supposed to retrieve a record from the users table where the name column matches the name specified by the user. Under normal circumstances, **\$name** would only contain alphanumeric characters and perhaps spaces, such as the string ilia. But here, by appending an entirely new query to \$name, the call to the database turns into disaster: the injected DELETE query removes all records from users.

There are databases interfaces which do not permit query stacking or executing multiple queries in a single function call. If you try to stack queries, the call fails but SQLite and PostgreSQL, happily perform stacked queries, executing all of the queries provided in one string and creating a serious security problem.

## Preventing SQL Injection:

You can handle all escape characters smartly in scripting languages like PERL and PHP. Programming language PHP provides the function **string sqlite\_escape\_string** to escape input characters that are special to SQLite.

```
if (get_magic_quotes_gpc())
{
    $name = sqlite_escape_string($name);
}
$result = @$db->query("SELECT * FROM users WHERE username='{$name}'");
```

Although the encoding makes it safe to insert the data, it will render simple text comparisons and **LIKE** clauses in your queries unusable for the columns that contain the binary data.

Keep a note that addslashes should NOT be used to quote your strings for SQLite queries; it will lead to strange results when retrieving your data.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js