

Übungsblatt #11

Bearbeitungszeitraum: bis Donnerstag, 21.01.2010

Was will uns dieses Übungsblatt sagen?

1. Beim Absetzen von SQL-Statements, die aufgrund von Benutzereingaben erstellt worden sind (z.B. nach Eingabe eines Suchbegriffs), besteht stets die Gefahr, eine sehr kritische Sicherheitslücke in dem jeweiligen Programm / Skript geschaffen zu haben.
 2. Besonders relevant ist in diesem Zusammenhang ein Angriff mittels „SQL-Injection“. Ein solcher besteht aus dem Einschleusen von böartigem SQL-Code an den Stellen, an denen Benutzereingaben erwartet werden.
 3. Perl/DBI kennt als Vorsorge gegen derartige Angriffe das Konzept der „Prepared Statements“, das nahezu identisch z.B. auch in Java bzw. JDBC Eingang gefunden hat.
 4. Für den Zugriff auf mehrzeilige Abfrageergebnisse stehen in DBI mehrere Möglichkeiten zur Verfügung.
- Welche man verwendet, ist oftmals lediglich eine Frage des persönlichen Geschmacks, ergibt sich mitunter aber auch aus dem Kontext.*
5. Es bietet sich an, ein etwas umfangreicheres Perl-Skript in Module aufzuteilen, wobei jedes Modul (nur) einen speziellen Aufgabenbereich abdeckt (und sonst nichts).
 6. Ein einfaches Beispiel für dieses Konzept der „Arbeits(auf)teilung“ ist das MVC-Pattern. Es trennt Datenhaltung (Modell) von der Verarbeitung (Controller) und von den Ein-/Ausgabefunktionen (View).
 7. Das MVC-Pattern macht es möglich, zwei Views zu haben: eine kommandozeilenbasierte und eine auf Basis von Tk (GUI). Beide Views nutzen denselben Controller und dasselbe Modell.

Aufgabe 1): SQL-Injection – Potenzielle Sicherheitslücken im Umgang mit Perl/DBI

Auf der zweiten Seite dieses Übungsblattes seht Ihr einen kleinen Comic. Eure Aufgabe ist es, den Witz dahinter herauszufinden. Lest dazu im DBI-Skript den Abschnitt über SQL-Injection durch. Nach dieser Lektüre wird der Comic verständlich. Welche Maßnahmen helfen, um – wie es die Mutter in dem Comic ausdrückt – die Benutzereingaben zu „desinfizieren“ („sanitize“)?

Aufgabe 2): Blatt 08 reloaded

Auf dem Übungsblatt Nr. 8 haben wir ein Programm geschrieben, das uns gewisse Auswertungen einer Logdatei eines Web-Servers ermöglicht hat. Die entsprechende Logdatei lag damals in Form einer CSV-Datei vor.

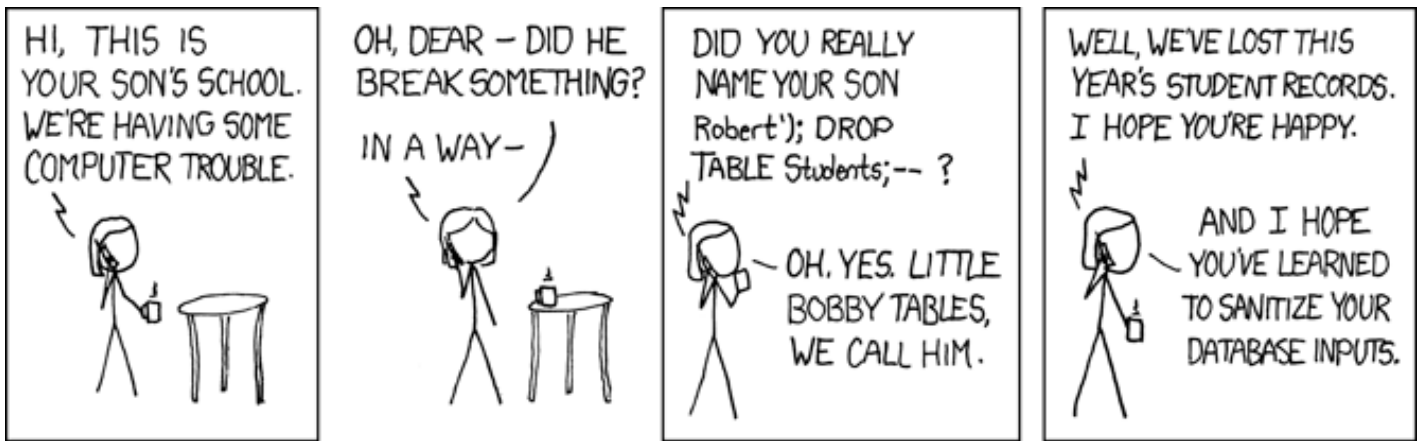
Mittlerweile haben fleißige Heinzelmännchen diese CSV-Datei in eine MySQL-Tabelle (bzw. -Datenbank) überführt. Auf dem bekannten Server (vgl. Übungsblatt Nr. 1) findet Ihr nun eine weitere Datenbank namens „accessLogs“, die aus genau einer Tabelle, nämlich jener mit den Logaufzeichnungen, besteht.



Eines der fleißigen Heinzelmännchen

Schreibt die Lösung aus Blatt 8 (entweder Eure eigene oder unsere Beispiellösung) so um, dass sie fortan die Daten nicht mehr aus der CSV-Datei, sondern aus der Datenbank bezieht. Der Funktionsumfang soll unverändert bleiben. Selbstverständlich dürft bzw. müsst Ihr dazu die Datenstrukturen nach eigenen Wünschen abändern.

(Dass Ihr bei Eurer Lösung darauf achtet, Euer in Aufgabe 1 neu erworbenes Wissen rund um SQL-Injection anzuwenden, versteht sich doch hoffentlich von selbst, oder...?)



Aufgabe 3): Module und das MVC-Pattern

Wir können das Resultat aus Aufgabe 2 dieses Blattes auch nutzen, um den Umgang mit Modulen zu üben. Module hatten wir in der Vorlesung mal kurz erwähnt, aber noch nicht weiter in den Übungen behandelt.

Nehmen wir mal an, Ihr habt Aufgabe 2 fertig gestellt. Im schlimmsten Fall habt Ihr jetzt ein *rieesiges* Perl-Skript, das alles andere als schön, ordentlich und übersichtlich ist.

Angenommen, in der nächsten Woche lautete die Aufgabe, eine Tk-GUI für dieses Programm zu schreiben. Dann müsstet Ihr ein neues Perl-Programm schreiben, das vieles von dem bisherigen Programm übernimmt, sodass Ihr diese Teile dann doppelt habt. Würde sich nun etwas an der Datenbank ändern, so müsstet Ihr beide Programme (das bisherige und das neue) entsprechend modifizieren. Das ist fehleranfällig und (unnötigerweise) doppelte Arbeit!

Besser ist es, wenn Ihr das Programm in logische Teile (Module) zerlegt: ein Modul kümmert sich (nur) um die Verbindung zu Datenbank und um das Holen der Informationen daraus, ein anderes enthält (nur) die Sortierfunktionen, und ein drittes ist für die Ein- und Ausgabe auf der Kommandozeile zuständig.

Schaut, dass Ihr Eure Lösung so strukturiert, wie es im voranstehenden Absatz vorgeschlagen ist. Denn wenn dann in der nächsten Woche die Aufgabe lauten würde, eine GUI zu Eurem Programm zu schreiben, könnt Ihr 2 der 3 Module ohne weitere Änderungen wiederverwenden (Sortierfunktionen, Datenbankzugriffe), und lediglich das Modul, das sich um die Ein- und Ausgabe kümmert, muß ersetzt werden.

Dies ist zugleich auch ein einfaches Beispiel für das sogenannte MVC-Pattern, wobei M für „Modell“, V für „View“ und „C“ für Controller steht. Grob gesagt, sollen alle Programme (hinreichenden Umfangs) diese drei Bestandteile voneinander getrennt vorhalten. In unserem Fall wäre das Modell (M) die bereits existierende Datenbank, der Controller (C) das Modul, das sich um die Abfragen kümmert und jenes, das die Sortierfunktionen enthält, und die View (V) wäre das Modul, das die Ein- und Ausgabefunktionalität bereitstellt.

In dieser neuen Begriffswelt würde die Aufgabe, eine GUI zu schreiben, also bedeuten, lediglich eine weitere „View“ zur Verfügung zu stellen. Diese verwendet dann die anderen Module (Controller). Eine Änderung am Controller würde sich andererseits auf alle (beide) Views zugleich auswirken.