

EBERHARD KARLS  
UNIVERSITÄT  
TÜBINGEN



## **Lehrstuhl für Datenbanksysteme**

Wilhelm-Schickard-Institut für Informatik

### **TRYDSH - Webanwendung**

#### **Studienarbeit**

von

Tim Digel

Betreuer: George Giorgidze

2. November 2011

# Inhaltsverzeichnis

|  |            |
|--|------------|
| <b>Inhaltsverzeichnis.....</b>               | <b>II</b>  |
| <b>Abbildungsverzeichnis.....</b>            | <b>III</b> |
| <b>Abstract .....</b>                        | <b>IV</b>  |
| <b>Eidesstattliche Erklärung .....</b>       | <b>V</b>   |
| <b>1 Einleitung .....</b>                    | <b>1</b>   |
| <b>2 Funktionsübersicht .....</b>            | <b>2</b>   |
| <b>3 Übersicht der Implementierung .....</b> | <b>4</b>   |
| <b>4 Details zur Implementierung.....</b>    | <b>5</b>   |
| 4.1 Clientseitige Implementierung.....       | 6          |
| 4.2 Serverseitige Implementierung.....       | 8          |
| 4.3 Datenbankinteraktion .....               | 11         |
| 4.4 Sicherheit.....                          | 13         |
| <b>5 Getesteter Funktionsumfang.....</b>     | <b>15</b>  |
| <b>6 Fazit.....</b>                          | <b>15</b>  |
| <b>Referenzen &amp; Quellen .....</b>        | <b>17</b>  |

## Abbildungsverzeichnis

|  |    |
|--|----|
| Abbildung 1: Schema zur Ferry-Optimierung (rechts) .....                           | 1  |
| Abbildung 2: Übersicht TRYDSH Webanwendung mit beispielhafter Auswertung .....     | 2  |
| Abbildung 3: Tabellenverwaltung.....   | 3  |
| Abbildung 4: Schema zur Interaktion von Browser, Server, DB und Haskell .....      | 5  |
| Abbildung 5: Kommunikation per AJAX zwischen Client und Server .....               | 7  |
| Abbildung 6: Schema zur Ergebnisanzeige am Beispiel von geschachtelten Daten ..... | 7  |
| Abbildung 7: Serverseitige Programmauswertung .....                                | 10 |
| Abbildung 8: Schema zur Datenbankinteraktion .....                                 | 12 |
| Abbildung 8: Schema zu Sicherheitssystemen .....                                   | 13 |

## Abstract

Database Supported Haskell (DSH) ermöglicht Programme, die auf einer großen Menge an Daten operieren, direkt in relationalen Datenbankmanagementsystemen (RDBMS) auszuführen. Dadurch ergeben sich große Performancevorteile, da Datenbanken für große Datenmengen optimiert sind. TRYDSH bietet, auf DSH aufbauend, dem Benutzer eine Webanwendung, mit der DSH direkt ausprobiert werden kann. Dabei können die benötigten Daten zuvor über das Webinterface hochgeladen werden, um anschließend auf diesen Daten operierende Haskell-Programme direkt in der Datenbank auszuführen.

Database Supported Haskell (DSH) allows programmers to formulate programs that process a large scale data. A DSH program can be run directly on a relational database management system (RDMBS). Database-executable programs take advantage of the performance of the RDBMS. TRYDSH is a web application that allows to run DSH programs directly in your browser. The necessary data can be imported through the web application to the database. Subsequently a user can run her DSH program operating on the imported data directly at the database.

## Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Tübingen, .....

.....

Unterschrift

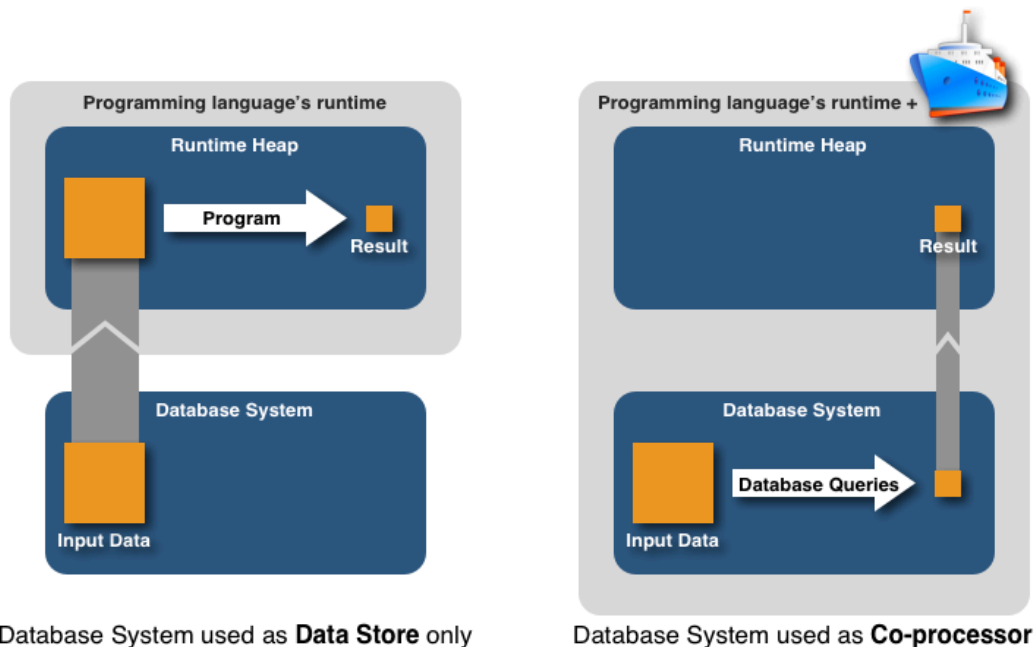
# 1 Einleitung

Oft kann man Daten, wie sie in einer Datenbank gespeichert sind, nicht direkt weiterverwenden. Man benötigt Anpassungen, bei denen man meist mit einer oder mehreren Schleifen durch den kompletten Datensatz laufen muss um am Ende womöglich eine viel kleinere Ergebnismenge zu erhalten.

An dieser Stelle setzt Ferry<sup>[1]</sup> an: Ferry ermöglicht es, datenlastige Programme noch innerhalb des relationalen DBMS auszuführen. Die eigentliche Programmierumgebung kann dann mit der, unter Umständen viel kleineren, Datenmenge arbeiten. Weitere Informationen zu Ferry siehe [1].

Database-supported Haskell (DSH)<sup>[2]</sup> stellt eine solche Implementation für Haskell<sup>[3]</sup> dar. DSH kompiliert Programme zu mehreren SQL-Anweisungen, die dann sehr rasch vom DBMS auf den Daten ausgeführt werden können. Weitere Informationen zu DSH können der Veröffentlichung unter [2] entnommen werden.

Im Rahmen dieser Studienarbeit wurde eine webbasierte Testumgebung für DSH



<sup>1</sup> Quelle: <http://www.db.inf.uni-tuebingen.de/research/ferry> (abgerufen am 15.09.2011).

entwickelt. Um ohne große Softwareinstallation- und –konfiguration einen Einblick zu bekommen, bietet die TRYDSH genannte Anwendung eine einfach zu bedienende Testumgebung. Benutzer können ihre DSH-Programme von einem Server bequem ausführen lassen. Registrierten Benutzer wird zudem ermöglicht Tabellen und Programme hochzuladen und zu speichern. Die Webanwendung erspart den Benutzern dadurch die aufwendige Installation und Konfiguration von einer Datenbank, einem Haskell-Interpreter und der DSH-Bibliothek. Ein Nachteil hierbei ist aber die hohe Latenzzeit zum entfernten TRYDSH-Server, was aber bei der heutigen guten Internetstruktur zu vernachlässigen ist.

Auf den folgenden Seiten wird zuerst der Funktionsumfang von TRYDSH beschrieben, ehe ein Blick in die Implementierung und die verwendeten Programmiersprachen und –techniken gewährt wird.

## 2 Funktionsübersicht

Folgender Abschnitt beschreibt den Funktionsumfang der Webanwendung, der sich in einen allgemeinen „Gast-Modus“ und eine Version mit erweitertem Umfang für registrierte Benutzer gliedern lässt.

Die wichtigste Funktion von TRYDSH ist das Eingabefeld. In diesem können die DSH-Programme eingegeben werden und zum Ausführen

TRYDSH v0.1 | Register | Login

DSH :: [SQL]

[Home](#) [About](#)

```
-- Here you can type your own database-executable program.
employees :: Q [(Text, Text, Integer)]
employees = toQ [
  ("Simon", "MS", 80)
  , ("Erik", "MS", 90)
  , ("Phil", "Ed", 40)
  , ("Gordon", "Ed", 45)
  , ("Paul", "Yale", 60)
]

departments :: Q [Text]
departments = nub [ dept | (view -> (name,dept,salary)) <- employees ]

deptSalary :: Q Text -> Q Integer
deptSalary dept = sum [ salary
  | (view -> (name,dept',salary)) <- employees
  , dept == dept' ]
```

Run

Table Haskell JSON Download JSON Download CSV SQL Query Plan Optimised Query Plan

| dept | salary |
|------|--------|
| MS   | 170    |
| Ed   | 85     |
| Yale | 60     |

© DB Uni Tübingen - www-db.informatik.uni-tuebingen.de - (0,034 s)

Abbildung 2: Übersicht TRYDSH Webanwendung mit beispielhafter Auswertung

an den Server geschickt werden. Wie DSH-Programme formuliert werden, kann in der Publikation „Bringing Back Monad Comprehensions“ (siehe [4]) nach gelesen werden. Das Ergebnis des Programms, zum Beispiel eine Tabelle, eine Liste oder geschachtelte Daten, kann nun auf verschiedene Weise betrachtet werden: Als Tabelle, in Haskell-Notation, in JSON-Notation, als SQL-Anweisungen, als XML-Baum und zum Download als CSV-Datei (Comma Separated Values, Komma getrennte Werte). Dabei können bei allen Arten, außer CSV, auch geschachtelte Daten repräsentiert werden.

Um die DSH-Bibliothek umfangreich testen zu können, ist der oben genannte Funktionsumfang unzureichend. DSH bietet große Vorteile sofern ein Programm auf vielen Daten operieren muss. Um diese Daten bequem der Webanwendung zu übermitteln gibt es für registrierte Benutzer umfangreiche Möglichkeiten Tabellen und Programme zu verwalten. Sobald der Benutzer sich direkt innerhalb der Webanwendung einmalig registriert hat, erhält er seine eigene Datenbank, in der er Tabellen erstellen und füllen kann. Das Anlegen der Tabellen geschieht über ein

### Edit Table

Please enter a tablename and specify the columns.

Tablename:

| Columns                               | Datatype                           | Delete                   | New Index                |
|---------------------------------------|------------------------------------|--------------------------|--------------------------|
| <input type="text" value="id"/>       | <input type="text" value="INT"/>   | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="text" value="a_number"/> | <input type="text" value="FLOAT"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="text" value="a_text"/>   | <input type="text" value="TEXT"/>  | <input type="checkbox"/> | <input type="checkbox"/> |

| Indexes                                  | On Columns           | Delete                   |
|--|----------------------|--------------------------|
| <input type="text" value="index_on_id"/> | id                   | <input type="checkbox"/> |
| <input type="text" value="newindex_0"/>  | id, a_number, a_text | <input type="checkbox"/> |

### My Tables

Create, upload and delete your tables

| Name          | Rows | Cols |                             |                      |                       |                        |
|---------------|------|------|-----------------------------|----------------------|-----------------------|------------------------|
| another_table | 0    | 2    | <a href="#">Upload data</a> | <a href="#">Edit</a> | <a href="#">Empty</a> | <a href="#">Delete</a> |
| my_table      | 0    | 3    | <a href="#">Upload data</a> | <a href="#">Edit</a> | <a href="#">Empty</a> | <a href="#">Delete</a> |

Create a new Table with  columns.

Abbildung 3: Tabellenverwaltung

Formular, wo neben dem Tabellennamen, den Spaltennamen, den Spaltentypen (Integer, Floating Point Numbers und Text) auch Indizes erstellt werden können. Da jeder Benutzer seine eigene Datenbank erhält kommt es zu keinen Namensüberschneidungen mit Tabellen anderer Benutzer.

Nach dem die Struktur einer Tabelle erstellt wurde, können die Daten mittels Upload von CSV-Dateien eingefügt werden. Bei registrierten Benutzern, verbindet sich der Haskell-Interpreter automatisch mit der entsprechenden Datenbank. Dadurch stehen alle zuvor erstellten Tabellen der Programmausführung zur Verfügung.

Neben Tabellen können registrierte Benutzer auch Ihre Programme abspeichern lassen. Im Eingabefeld ausgeführte Programme werden auf Wunsch für spätere Aufrufe gespeichert. Dabei werden bei gleichem Programmname bis zu 10 verschiedene Versionen zurück gehalten. So können nicht zielführende Modifikationen wieder rückgängig gemacht werden.

### 3 Übersicht der Implementierung

Folgender Abschnitt soll einen groben Überblick über die verwendeten Techniken und Programmiersprachen von TRYDSH schaffen. Der Quellcode kann unter [5] eingesehen werden.

Bei heutigen Webanwendungen rückt die ständige Interaktion des Benutzers immer stärker in den Vordergrund. Um diesen Anforderungen gerecht zu werden, wird neben serverseitigen Programmiersprachen, auch Javascript und jQuery<sup>[6]</sup> eingesetzt. Dadurch bleibt die Anzeige einer Seite nicht statisch. Inhalte können so ohne Neuladen oder nur durch teilweise Aktualisierungen verändert werden.

Die vom Benutzer eingegebenen Programme werden per AJAX (Asynchronous JavaScript and XML) an den Server gesendet. Der Server übermittelt die Antwort als JSON (JavaScript Object Notation) Objekt an den Client zurück. Der Client wiederum verarbeitet das JSON-Objekt und stellt dem Benutzer das Ergebnis dar.

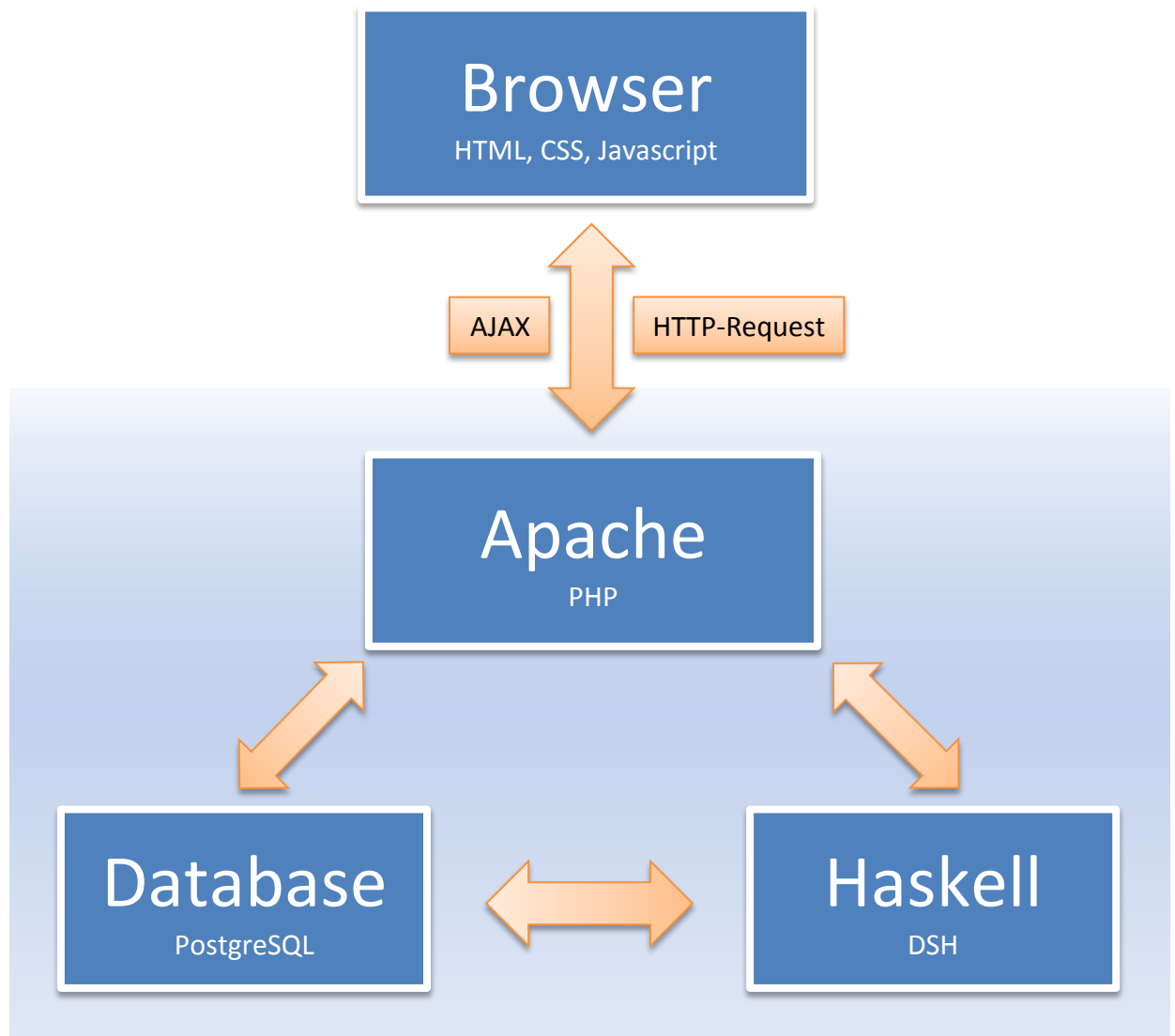


Abbildung 4: Schema zur Interaktion von Browser, Server, DB und Haskell

Der Server lässt sich in drei Teile gliedern: Den Apache-Webserver<sup>[7]</sup> mit der Skript-Sprache PHP<sup>[8]</sup>, die PostgreSQL-Datenbank<sup>[9]</sup> und den Haskell-Interpreter GHC<sup>[10]</sup>. Dabei greift neben dem Haskell-Interpreter auch der Apache zur Verwaltung der registrierten Benutzer, der Benutzertabellen und der gespeicherten Programme auf eine Datenbank zurück.

#### 4 Details zur Implementierung

Zur Implementierung bedarf es auf Grund der in Abbildung 4 zu erkennenden komplexeren Beziehungen einen genaueren Blick. Dabei wird im Folgenden zuerst die

Clientseite beschrieben, gefolgt von der Serverseite. Zuletzt wird noch das Thema Sicherheit erörtert.

#### 4.1 Clientseitige Implementierung

TRYDSH soll eine einfache Möglichkeit bieten die DSH-Bibliothek ohne aufwendige Softwareinstallation zu Testen. Dafür bietet sich eine Webanwendung an, die mit beliebigem Browser aufgerufen werden kann. Um eine hohe Kompatibilität für alle Browser zu gewährleisten ist gängiger HTML-Code erforderlich. Dabei wird im Wesentlichen das Design durch Verwendung zentral eingebundener CSS-Vorlagen (Cascading Style Sheets<sup>[11]</sup>) vom HTML-Code getrennt. Durch die zentrale Einbindung lassen sich so Design- und Layout-Änderungen mühelos einpflegen, zudem gewinnt jedes HTML-Dokument an Übersicht.

TRYDSH ist in mehrere Seiten unterteilt. So gibt es neben der Startseite, die die Hauptfunktionalität beinhaltet, weitere Seiten für Erklärungen, Registrierung, Benutzerverwaltung, sowie ganze Unterbereiche zur Tabellen- und Programmverwaltung.

Um den Benutzern eine gewisse Dynamik, ohne ständige neue Seitenaufrufe, zu gewähren, setzt TRYDSH auf AJAX-Anfragen und Javascript- bzw. jQuery-Funktionalität. jQuery ist eine Javascript-Bibliothek, die einige sehr nützliche Funktionen zur Verfügung stellt (siehe [6]). Um nur einige Beispiele zu nennen: AJAX-Funktionalität, einfachere Selektionsmöglichkeiten der vorhandenen HTML-Objekte, vordefinierte Animation zum Ein- und Ausblenden von Ebenen. Viele weitere nützliche Funktionen werden durch Erweiterungen und Plugins zur Verfügung gestellt. So benutzt TRYDSH unter anderem jQuery-Plugins zur Design- und Layoutoptimierung.

Wie bereits in Abschnitt 3 kurz erwähnt wird ein Programm eines Benutzers per AJAX an den Server übermittelt. Bei AJAX verläuft die Kommunikation zwischen Browser und Server im Hintergrund. Dadurch bleibt zum einem die bisherige Ansicht

bestehen und dem Benutzer kann ein „Bitte warten...“-Hinweis angezeigt werden. Zum anderen ist dadurch kein Neuaufruf der Seite erforderlich, da das Ergebnis bei AJAX direkt an die Javascript-Umgebung des Browsers zurück geschickt wird. Dabei werden

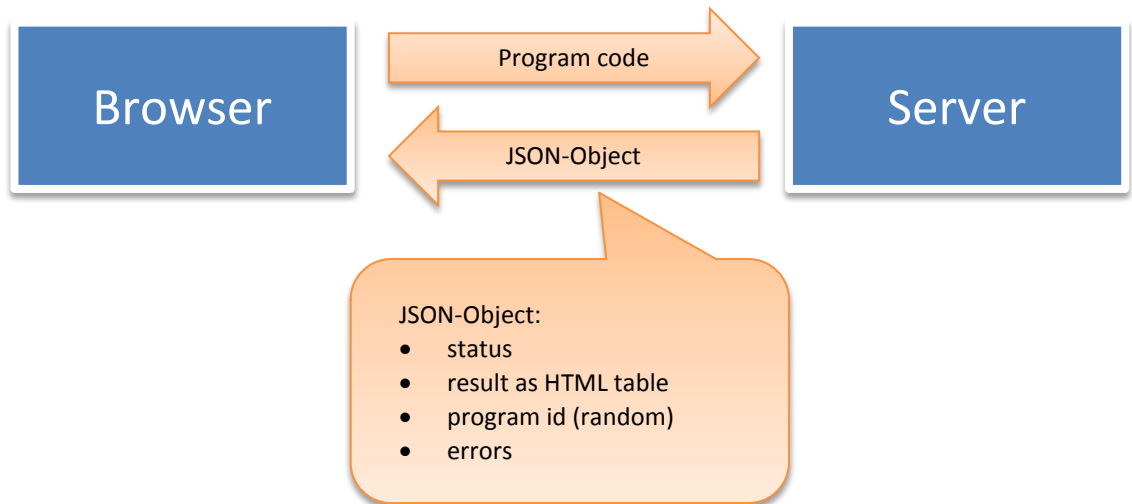


Abbildung 5: Kommunikation per AJAX zwischen Client und Server

alle nötigen Informationen, wie Status und Ergebnis der Programmausführung, evtl. Fehlermeldungen, eine zufällig festgelegte Programm-ID, usw. in einem JSON-Objekt übermittelt. Dieses Objekt wird mittels jQuery eingelesen und die Ergebnisse dem Benutzer entsprechend dargestellt.

Die Ergebnismeldung wird in folgendem Abschnitt genauer beschrieben. Sollte ein

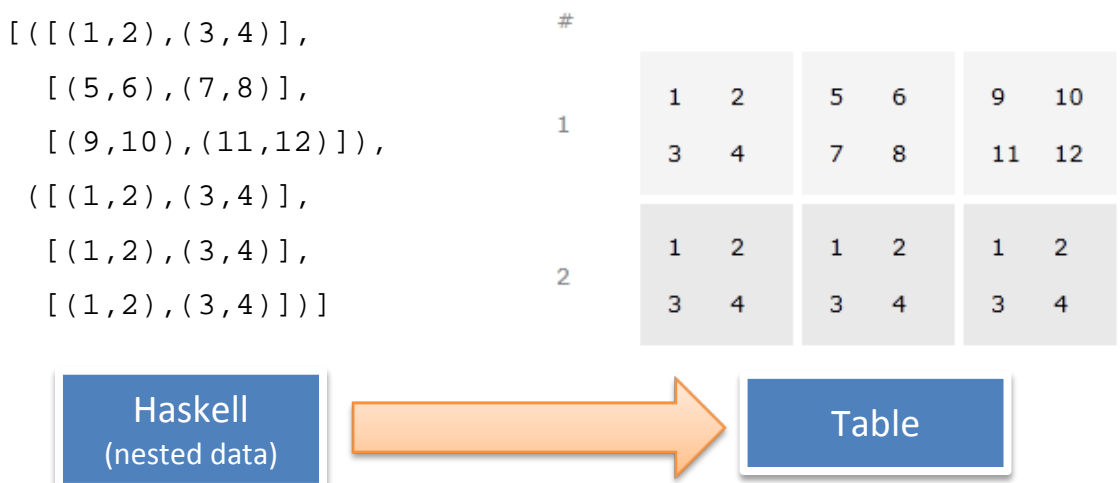


Abbildung 6: Schema zur Ergebnisanzeige am Beispiel von geschachtelten Daten

Programm auf Grund syntaktischer Fehler kein Ergebnis liefern, beinhaltet die Server-Antwort eine detaillierte Fehlermeldung, die dem Benutzer angezeigt wird. Bei erfolgreicher Antwort enthält das JSON-Objekt das Ergebnis als HTML-Tabelle. Diese HTML-Tabelle wird dem Benutzer mit entsprechenden Style-Vorlagen direkt ausgegeben. In dieser Notation lassen sich neben Tabellen, Listen, und Basiswerten wie Zahlen oder Zeichenketten auch geschachtelte Daten ausgeben. Die geschachtelten Daten werden dabei einfach als geschachtelte HTML-Tabellen angezeigt. Inhalt einer HTML-Zelle ist dann wiederum eine HTML-Tabelle (siehe auch Abbildung 6).

Neben der Standardansicht als Tabelle gibt es noch weitere Ergebnisdarstellungen: Neben JSON- und Haskell-Notation bietet TRYDSH auch den Download als CSV-Datei an, sowie weitere Ansichten, die zum Debugging und Verständnis sehr hilfreich sind. So lassen sich zum einem die von DSH kompilierten SQL-Abfragen begutachten und zum anderen besteht die Möglichkeit, XML-Query-Bäume direkt mithilfe von FerryLeaks<sup>[12]</sup> zu begutachten. FerryLeaks ist ebenfalls eine Webanwendung, die Query-Pläne visualisiert und grafisch anzeigt.

Die weiteren Ansichten und Downloadmöglichkeiten werden dabei durch eine weitere Anfrage an den Server zur Verfügung gestellt. Neben der gewünschten Ausgabe wird dabei die zuvor erhaltende Programm-ID und nicht erneut das komplette Programm abgeschickt. Das Ergebnis wird im Fall von CSV direkt als Download gekennzeichnet, bei SQL, JSON und Haskell erscheint die gewünschte Darstellung direkt im Ergebnisbereich. XML-Query-Bäume werden direkt in FerryLeaks geöffnet, wobei der benötigte Code zuvor über ein HTTP-Post Interface übertragen wird.

## 4.2 Serverseitige Implementierung

In Abbildung 4 sind die groben Beziehungen der Dienste innerhalb des Servers dargestellt. Die Schnittstelle ist hierbei der Apache-Server und die als Apache-Modul geladene Programmierumgebung PHP (**PHP Hypertext Preprocessor**, siehe [8]). PHP wurde als serverseitige Skriptsprache verwendet, da es sich durch eine sehr breite

Unterstützung vieler relationaler Datenbankmanagementsysteme aus (unter anderem auch das von TRYDSH verwendete PostgreSQL) und durch einen sehr großen Funktions- und Bibliothekenumfang ausgezeichnet.

Die Webanwendung TRYDSH ist in mehrere PHP Skripte unterteilt. Neben den Seiten zur Verwaltung der Benutzertabellen und –programmen und der Startseite stellt das Programmausführungsskript die wichtigste Komponente dar (siehe auch Abbildung 7). Dieses Skript erhält das auszuführende Programm mittels HTTP-Post Anfrage vom Browser. Das eigentliche Programm wird dann in ein vordefiniertes Haskell-Skript eingebaut. Dadurch werden zum einen die benötigten Haskell-Import-Befehle definiert, die Verbindung mit der zur Ausführung benötigten Datenbank hergestellt und die DSH-Routinen aufgerufen.

Das eingebettete Programm wird anschließend in eine temporäre Datei geschrieben. PHP bietet nun die Möglichkeit mithilfe der Funktion `proc_open`<sup>2</sup> beliebige Shell-Kommandos auszuführen. In diesem Fall wird mit dieser Funktion der Haskell-Interpreter, genauer gesagt `runghc`<sup>3</sup>, geöffnet. `Runghc` ist eine Funktion des Haskell-Compiler GHC, die keine vorherige Kompilierung des Quellcodes benötigt, sondern direkt die Ausgabe eines entsprechenden Programmes liefert.

Mithilfe von `proc_open` werden nun drei Pipes zwischen PHP und `runghc` hergestellt. In einer Pipe kann PHP weitere Befehle schreiben, in der zweiten wird die Ausgabe von `runghc` übermittelt. In der dritten Pipe werden ggf. auftretenden Fehler zurückgeschrieben.

---

<sup>2</sup> Quelle: <http://php.net/manual/de/function.proc-open.php>

<sup>3</sup> Quelle: [http://www.haskell.org/ghc/docs/latest/html/users\\_guide/runghc.html](http://www.haskell.org/ghc/docs/latest/html/users_guide/runghc.html)

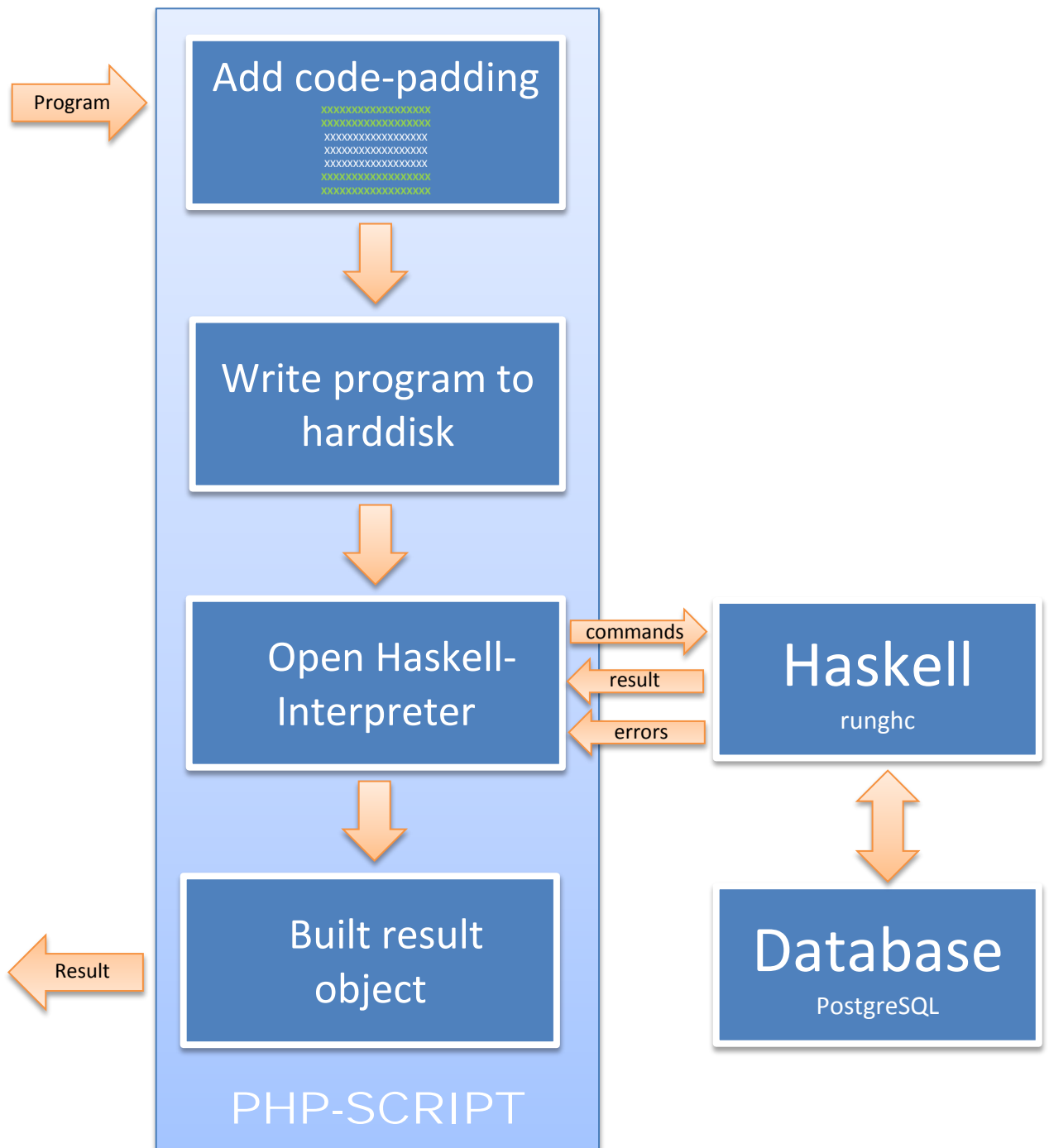


Abbildung 7: Serverseitige Programmauswertung

Dem Aufruf von `runghc` wird neben dem Dateinamen des Programcodes auch die gewünschte Ausgabemethode als weiteres Argument übergeben. Sobald die Übermittlung des Ergebnis und ggf. auftretender Fehlermeldungen beendet ist, wird der `runghc`-Prozess wieder geschlossen und mit der Ausgabe fortgefahren. Das Ergebnis wird in ein Rückgabearray eingebaut (vgl. Abbildung 5) und dann mithilfe der Funktion `json_encode` in JSON-Notation an den Browser übermittelt.

Bei den alternativ angebotenen Ergebnisdarstellungen (JSON, Haskell, CSV, SQL, Query-Plan-Baum, etc.) läuft obiges Skript geringfügig anders ab. Wie bereits erwähnt werden hierfür nur noch die Programm-ID und die gewünschte Ausgabemethode an den Server übermittelt. Dadurch muss das Programm nicht erneut in eine Datei geschrieben werden. Die DSH-Bibliothek enthält Funktionen zur Ausgabe aller angebotenen Formate. Es findet eine Fallunterscheidung innerhalb des zuvor angefügten allgemeinen Padding-Codes statt. `runghc` erhält als weiteres Argument im Kommandoaufruf die gewünschte Ausgabemethode (z. B. `sql` für SQL). Das eigentliche Programm wird ausgeführt und das gewünschte Ergebnis zurückgeliefert. Eine weitere Verarbeitung innerhalb des PHP-Skriptes ist hierbei nicht mehr nötig. Bei JSON, Haskell, CSV und SQL wird das Ergebnis direkt an den Browser geschickt, beim Query-Baum im XML-Format direkt weiter an FerryLeaks.

### 4.3 Datenbankinteraktion

Sowohl die Webanwendung an sich, als auch DSH verwenden PostgreSQL als relationales DBMS. Weitere Informationen zu PostgreSQL siehe [9]. Die Webanwendung beschränkt sich dabei auf Tabellen zur Benutzerverwaltung, der Programmabspeicherung und des Login-Management.

Darüber hinaus erhält jeder registrierte Benutzer seine eigene Datenbank innerhalb PostgreSQL. In dieser Datenbank können über die Webanwendung bis zu 1000 Tabellen angelegt werden. Dabei stellt TRYDSH eine gewisse Grundfunktionalität, vergleichbar mit Tools wie phpMyAdmin (MySQL) oder pgMyAdmin (PSQL), zur Verfügung, die es erlaubt, Tabellen, Spalten und Indizes zu verwalten. Der Import von Daten wird durch Upload von CSV-Dateien ermöglicht.

Um die Verwaltung der Benutzerdatenbank durch die Webanwendung zu gewährleisten werden die von PostgreSQL angelegten Tabellen des `information_schema`<sup>4</sup> und des `pg_catalog`<sup>5</sup> verwendet. Damit ist ein aufwendiges Pflegen von Tabellen, die Informationen zu Tabellen, Spalten und Indizes

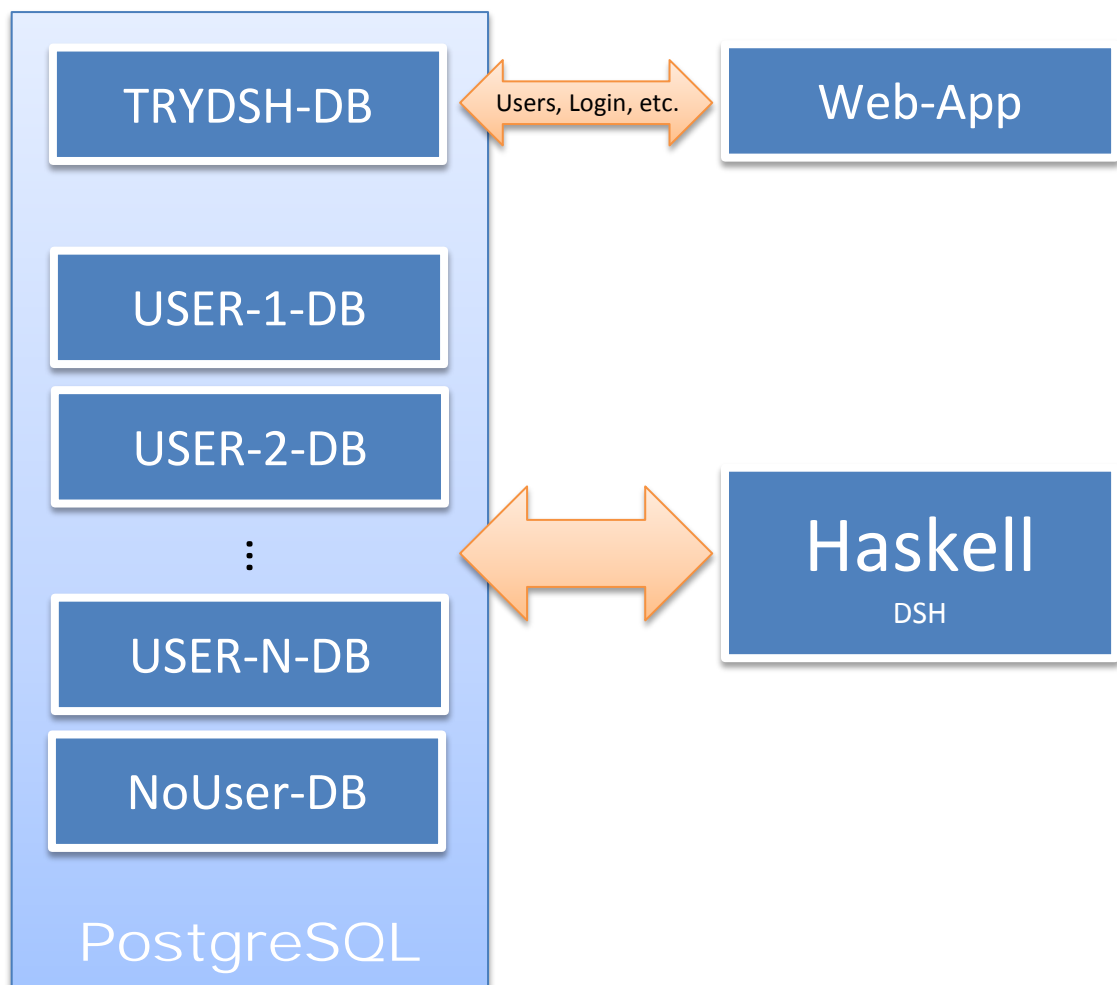


Abbildung 8: Schema zur Datenbankinteraktion

der Webanwendung zugänglich machen, nicht erforderlich.

Nach dem die entsprechenden Tabellen angelegt und gefüllt wurden, können Programme auf diesen Daten operieren. Der Padding-Code, der vom PHP-Skript um

---

<sup>4</sup> Quelle: <http://www.postgresql.org/docs/7.4/static/information-schema.html>

<sup>5</sup> Quelle: <http://www.postgresql.org/docs/9.1/static/catalogs.html>

das eigentliche Programm herumgebaut wird, baut hierbei die Verbindung zur Datenbank des Benutzers auf. Im Gastmodus wird dieser Teil durch die Verbindung zu einer leeren Datenbank ersetzt.

#### 4.4 Sicherheit

TRYDSH bietet den Benutzern an, beliebigen Code vom Server auszuführen. Normalerweise müssen Hacker erst Sicherheitslücken oder Schwachstellen in der Programmierung ausnutzen um an diesen Punkt zu gelangen. Deshalb ist es bei Webanwendungen, die eine Test-Umgebung für eine Programmiersprache zur Verfügung, immens wichtig, dass entsprechender Code keine Zugriffe auf andere Teile des Servers erlangt.

Um dies zu gewährleisten benutzt TRYDSH eine separate Installation des Haskell-Compiler GHC. Dieser Compiler wird innerhalb des Ubuntu-Systems<sup>[13]</sup> mit AppArmor<sup>[14]</sup> geschützt. Mit AppArmor kann für jeden Prozess ein eigenes Profil erstellt werden. In diesem Profil werden die bei normaler Verwendung benötigten

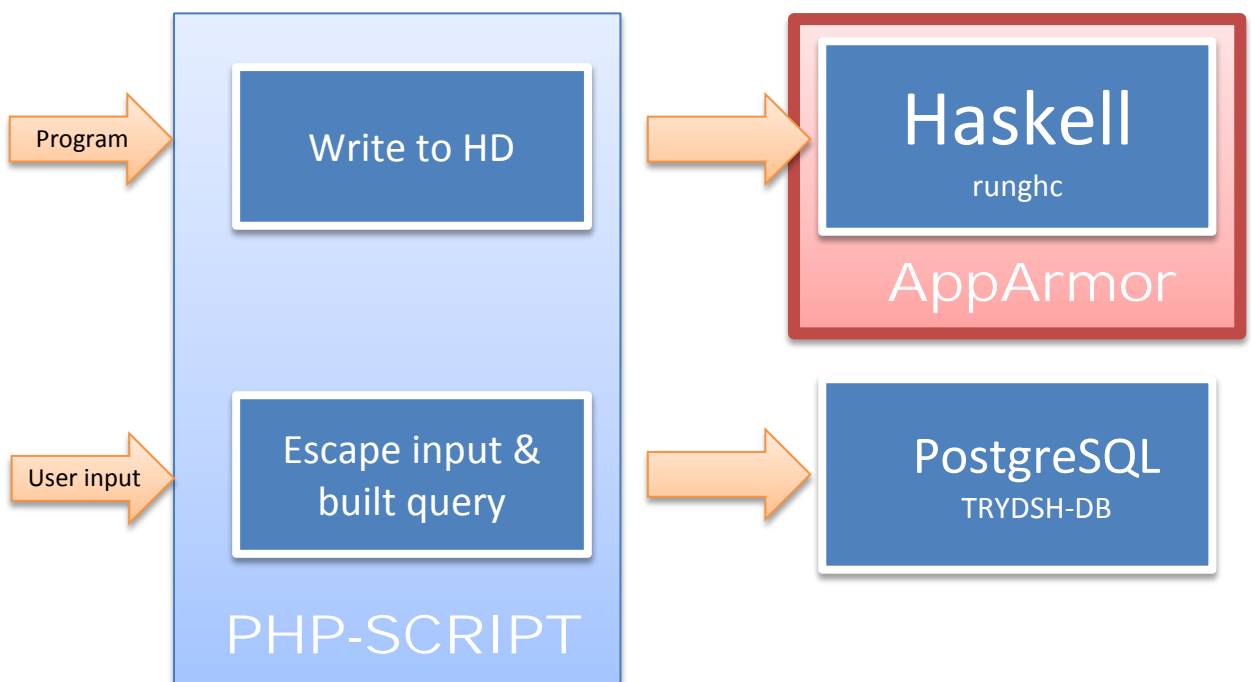


Abbildung 9: Schema zu Sicherheitssystemen

Unterprozesse, sowie den lesenden und/oder schreibenden Zugriff auf Verzeichnisse und Dateien, definiert. Bei dem Prozess `runghc`, der vom PHP-Skript zum Ausführen des Benutzer-Programmes aufgerufen wird, muss zum Beispiel der eigentliche Compiler-Prozess zugelassen werden. Dieser Prozess erhält wiederum ein Profil, das zum Beispiel den Zugriff auf das auszuführende Programm, ein schreibender Zugriff auf ein temporäres Verzeichnis, sowie den lesenden Zugriff auf alle Haskell-Bibliotheken ermöglicht.

Alles was nicht in den AppArmor-Profilen definiert wurde, wird nicht erlaubt. Programme von Benutzern laufen damit in einer sehr eng definierten Umgebung ab. Weitere Teile des Servers, wie der Apache oder die Datenbank bedürfen keines besonderen Schutzes durch AppArmor, da hier kein Code vom Benutzer ausgeführt wird. Das Verwalten von Tabellen wird zum Beispiel durch vorgegebene Formulare innerhalb der Webanwendung und nicht durch frei gestaltbare SQL-Anweisungen ermöglicht. Weitere Informationen zu AppArmor siehe [14].

Bei Webanwendungen generell gilt es dem Thema Sicherheit eine große Aufmerksamkeit zu widmen. Jede Eingabe, die vom Benutzer kommt, muss überprüft werden. PHP stellt hier nützliche Funktionen wie `pg_escape_string`<sup>6</sup> (PostgreSQL Escape String) zur Verfügung. Benutzereingaben werden zuerst auf Zeichen mit besonderer Bedeutung untersucht, bevor sie in eine Datenbank-Abfrage (zum Beispiel bei `SELECT * FROM user = 'Name'`) eingebaut werden. Zeichen mit besonderer Bedeutung sind zum Beispiel das einfache Anführungszeichen zur Begrenzung von Zeichenketten. Diesen Zeichen wird ein Slash vorangestellt, womit die besondere Bedeutung aufgehoben wird. Sogenannte SQL-Injections, die versuchen die Abfrage in Ihrer Sinnhaftigkeit zu verändern, werden damit wirksam unterbunden.

Des Weiteren werden alle Passwörter verschlüsselt gespeichert. Damit ist zwar ein Zusenden des Passwortes, sollte ein Benutzer es vergessen, nicht mehr möglich,

---

<sup>6</sup> Quelle: <http://de.php.net/manual/de/function.pg-escape-string.php>

allerdings können so die Passwörter nicht mit einfachen Mitteln im Klartext den Umlauf geraten, z. B. falls sie durch Sicherheitslücken oder menschliches Versagen bekannt wurden.

## 5 Getesteter Funktionsumfang

TRYDSH wurde gründlich getestet. Im Bereich der Tabellenverwaltung wurden neben Spalten hinzufügen, Anlegen und löschen von Tabellen, auch Typumwandlungen durchgeführt. Hier traten bei bestimmten Umwandlungen (z. B. Text nach Integer) behandelte Fehler auf, da solche Umwandlungen vom DBMS nicht unterstützt werden. Dem Benutzer wird bei allgemeinen Fehlern (zum Beispiel doppelter Tabellen- oder Spaltennamen) die hilfreiche PostgreSQL-Fehlermeldung angezeigt. Unbehandelte Fehler wurden keine erkannt.

Die DSH-Bibliothek wurde über die Webanwendung mit verschiedenen Programmen getestet. Teils greifen die Programme auf im Code definierte Tabellen und teils auf zuvor erstellte Tabellen in der Datenbank zurück. Dabei wurden stets alle möglichen Ausgabeformate betrachtet und keine Fehler festgestellt.

Ebenso wurde die Registrierung und die Loginfunktionalität getestet. TRYDSH funktioniert in allen modernen Browser problemlos (getestet mit Firefox, Safari, Opera und Internet Explorer 9).

## 6 Fazit

TRYDSH bietet eine einfache Möglichkeit die sonst aufwendig zu installierende DSH-Bibliothek, zur Optimierung von datenlastige Haskell-Programmen, zu testen. Dabei stellt TRYDSH eine einfach zu bedienende Webanwendung zur Verfügung, mit dem Benutzer neben ihren Programmen auch eine Vielzahl von Tabellen in ihrer eigenen Datenbank erstellen können.

Auf dieser Webanwendung kann zukünftig aufgebaut werden, wenn die DSH-Bibliothek im Umfang erweitert wird. Als Beispiel seien hier nur die Unterstützung von

weiteren Basiswerten für Tabellen innerhalb des Datenbanksystems oder die Möglichkeit weiterer Ausgabeformate anzubieten genannt. Weitere Ausgabeformate können sehr einfach umgesetzt werden, indem innerhalb DSH entsprechende Routinen dafür programmiert werden.

## Referenzen & Quellen

- [1] The Ferry project explores how far we can push the idea of relational database engines that directly and seamlessly participate in program evaluation to support the super-fast execution of data-intensive programs  
<http://db.inf.uni-tuebingen.de/research/ferry>
- [2] George Giorgidze, Torsten Grust, Tom Schreiber, and Jeroen Weijers. Haskell boards the Ferry: Database-supported program execution for Haskell. In Revised selected papers of the 22nd international symposium on Implementation and Application of Functional Languages, Alphen aan den Rijn, Netherlands, volume 6647 of Lecture Notes in Computer Science. Springer, 2010  
<http://db.inf.uni-tuebingen.de/files/publications/ferryhaskell.pdf>
- [3] Haskell is an advanced purely-functional programming language.  
<http://www.haskell.org>  
<http://www.haskell.org/onlinereport/haskell2010>
- [4] George Giorgidze, Torsten Grust, Nils Schweinsberg, and Jeroen Weijers. Bringing back monad comprehensions. In Proceedings of the ACM SIGPLAN Haskell symposium, Tokyo, Japan. ACM, 2011.  
<http://db.inf.uni-tuebingen.de/files/giorgidze/haskell2011.pdf>
- [5] Source Code of TRYDSH published at  
<https://github.com/giorgidze/TryDSH>
- [6] jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.  
<http://jquery.com>
- [7] Apache-Webserver  
<http://httpd.apache.org>
- [8] PHP Hypertext Preprocessor  
<http://www.php.net>
- [9] PostgreSQL Datenbank  
<http://www.postgresql.org/docs>
- [10] The Glasgow Haskell Compiler is a state-of-the-art, open source, compiler and interactive environment for the functional language Haskell  
<http://www.haskell.org/ghc>
- [11] Cascading Style Sheets  
<http://www.w3.org/Style/CSS>

- [12] FerryLeaks - A Web-based Editor for the Table Algebra  
Patrick Brosi  
Bachelor Thesis, Uni Tübingen  
<http://db.inf.uni-tuebingen.de/files/publications/FerryLeaks.pdf>  
<http://dbwiscam.informatik.uni-tuebingen.de/AlgebraEditor>
  
- [13] Linux Distribution Ubuntu  
<http://www.ubuntu.com>
  
- [14] AppArmor – Security Module for the Linux kernel  
<http://www.novell.com/documentation/apparmor>