Die lokale Variable „liste“ (Zeile 2) habe ich nur eingefügt, um den interessanten Befehl „options_from_collection_for_select“ zu überreden, das Listenelement mit der „id“ = 1 als „selektiert“ auszuweisen. Dieser „Helper“-Befehl, in der Source „form_options_helper“ zu finden, bereitet praktischerweise Listenelemente („collection“) genau in der Art und Weise auf, die eine <select>-Auswahlliste benötigt.

Der Zugriff auf das Listenfeld des ersten Datensatzes (Zeile 1) bewirkt dabei als Überschrift die Ausgabe des Listennamens, da dieser ja – gemäß der Suchstrategie – für alle Datensätze gleich sein muss und somit dem immer vorhandenen ersten Satz ohne weitere Prüfung entnommen werden kann. Natürlich hätte ich abfragen müssen, ob überhaupt ein Listenelement vorliegt, doch da diese kleine erste Anwendung nur wenige Listen mit klar vorgegebenen Listenelementen braucht, kann eine leere Liste gar nicht vorkommen.

3.2.4 Fazit

Was ich zwar prüfte, aber nicht vollendete, war, einen ausgewählten Datensatz der <select>-Auswahlliste zu bestimmen, sodass ich ihn im Fall der Fälle an die rufende Instanz zurück geben könnte. Weil das Entgegennehmen von irgendwelchen Benutzeraktivitäten ohne eine <form>-Anweisung in einem HTML freilich nicht wirklich erfolgreich zu werden verspricht – und ich längst schon den Entschluss gefasst hatte, die Listen auf sich beruhen zu lassen – suchte ich nicht weiter in dem mir immer noch unbekannten Gewirr von Ruby, Rails, HTML und Javascript herum, um die korrekte Vorgehensweise zu finden, mit der mir dieses Unterfangen glücken könnte.

Warum ich die Listen auf sich beruhen lassen will?

Weil sie einfach zu unbedeutend sind – und nicht wirklich etwas mit der kleinen Software zu tun haben, die ich hier kreieren möchte. Deshalb genügt für die Verwaltung der Listen auch phpMyAdmin.

Und für den Zugriff?

Da brauche ich nicht wie in meinen alten Umgebungen eine eigene, datenbank- und plattformunabhängige Datenschnittstelle zu bauen – Rails ist hier wirklich so komfortabel ...

... dass ich mir das schlicht sparen kann.

Doch nicht nur das – was Rails mir an fabelhaften Fähigkeiten von Ruby bereits andeutete, bewegt mich nicht nur, meine Listenprogrammierung beiseite zu schieben, es macht mir auch klar, dass ich mir meine Chancen, diese Flexibilität zu nutzen, nicht selbst kaputt machen sollte.

Wie ich das könnte?

Weil alte Hasen natürlich jahrelange Programmiererfahrung habe und damit jahrelange Routine, wie Dinge getan werden – genau damit können sie sich aber auch Möglichkeiten verbauen, die im Neuen, im Unbekannten liegen. Und sowohl Ruby als auch Rails sehen so viel versprechend aus, dass ich mir die Zeit nehmen möchte, beide kennen zu lernen und nicht nur auf meinen alten Strukturen herumzureiten.

Das ist der eigentliche Lernerfolg dieses ersten Schrittes.