Masterarbeit

Vergleich von SQL-Anfragen Theorie und Implementierung in Java

Robert Hartmann

Betreuer: Prof. Dr. Stefan Brass

24. September 2013

Martin-Luther-Universität Halle-Wittenberg

Inhaltsverzeichnis

1 Einleitung / Motivation

SQL (structured query language) ist eine Datenbanksprache, die in relationalen Datenbanken zum Definieren, Ändern und Abfragen von Datenbeständen benutzt wird. Basierend auf relationaler Algebra, ist sie einfach aufgebaut und ähnelt der englischen Sprache sehr, was Anfragen deutlich verständlicher gestaltet. Durch Standardisierungen in verschiedenen Versionen ist es möglich gewesen, dass mehrere DBMS (database management system) auf der Basis von SQL erstellt wurden. Zu bekannten Vertretern gehören Oracle Database von Oracle, DB2 von IBM, PostgreSQL von der PostgreSQL Global Development Group, MySQL von der Oracle Corporation und MS-SQL von Microsoft.

Die Umsetzung von SQL als quasi-natürliche Sprache erlaubt es Anfragen so zu formulieren, dass sie allein mit dem Verständnis der natürlichen Sprache verständlich sind. Dieser Umstand hat auch dazu geführt, dass heutzutage relationale Datenbanksysteme mit SQL beliebt sind und häufig eingesetzt werden. Dies führt allerdings auch dazu, dass es mehrere – syntaktisch unterschiedliche – Anfragen geben kann, welche semantisch identisch sind. Manche sehen sich dabei dennoch ähnlich andere gleiche Anfragen kann man nur nach Umformen oder umschreiben ineinander überführen.

Ein gängiges Mittel um herauszufinden ob zwei SQL-Anfragen das gleiche Ergebnis liefern, ist es die Anfragen auf einer Datenbank mit vorhandenen Daten auszuführen. Dies bildet jedoch lediglich Indizien für eine mögliche semantische Gleichheit. Da man die zwei zu vergleichenden Anfragen nur auf einer endlichen Menge von Datenbankzuständen testen kann, ist nie ausgeschlossen, dass nicht doch ein Zustand existiert, der unterschiedliche Ergebnisse liefert. Weiterhin stehen solche Testdaten nur im begrenzten Umfang zur Verfügung oder Daten müssten teuer besorgt werden, was nur für den Vergleich von zwei Anfragen möglicherweise zu teuer sein kann.

Diese Arbeit versucht daher zunächst die theoretische Grundlage und Analyse zu liefern, wie es möglich ist zwei Anfragen miteinander zu vergleichen, wenn man keine Datensätze zur Verfügung hat, sondern lediglich Kenntnis über die Datenbankstruktur, also das Datenbankschema, besitzt.

Nach der theoretischen Ausarbeitung soll ein Programm entwickelt werden, welches in der Lage ist zwei SQL-Anfragen mit erarbeiteten Strategien, zu vergleichen. Da die Fehlermeldung des Standardparser von SQL sehr allgemein gehalten sind, ist es auch wünschenswert, dass das Programm konkretere Hinweis- und Fehlermeldung ausgibt, als es der Standard SQL-Parser vermag.

Damit das Programm möglichst Plattformunabhängig bedient werden kann, soll es als Webseite auf einem Server zur Verfügung gestellt werden. Da als Programmiersprache Java gewählt wurde, bieten sich die JSP (java server pages), sowie java-servlets als Umsetzung dieser Anforderung an.

Ein mögliches Haupteinsatzgebiet ist die Lehre, so wie die Untersuchung des Lernfortschritts von Studenten oder anderen Interessierten, die den Einsatz von SQL erlernen möchten. So kann das Programm dem Lehrling(?) nicht nur sinnvolle Hinweise bei einer falschen Lösung geben, sondern auch erläutern, ob die gefundene Lösung eventuell zu kompliziert gedacht war. Des weiteren ist es aufgrund der zentralisierten Serverstruktur möglich, Lösungsversuche des Lernenden zu speichern und eine persönliche Lernerfolgskurve anzeigen zu lassen. Damit hätten Studenten und Lehrkräfte die Möglichkeiten Lernfortschritte zu beobachten und Problemfelder (etwa JOINs) zu erkennen um diese dann gezielt zu Bearbeiten.

Damit es ist möglich eine Lernplattform aufzubauen, die dem Studenten mehrere Auswertungsinformationen über seinen Lernerfolg und seine Lösung deutlich macht. So kann die Lehrkraft eine Aufgabe mit samt Musterlösung und Datenbankschema hinterlegen und der Student kann daraufhin seine Lösungsversuche in das System eintragen. Durch sinnvolles Feedback ist es ihm so möglich beim Üben direkt zu lernen. Weiterhin kann man eine solche Plattform auch für Tutorien oder Nachhilfe überall da benutzen, wo SQL gelernt wird. Vorteile hier wären, dass man mehrere verschiedene Aufgaben stellen kann ohne sich jedes mal um Testdaten zu kümmern.

Weitere Einsatzgebiete könnten im sich im Unternehmen befinden. So könnte man bei einer geplanten Umstrukturierung oder Erzeugung von Datenbanken bereits Anfragen prüfen und vergleiche bevor man sich u.U. teure Testdaten kauft oder Daten migrieren muss.

2 Theoretische Betrachtungen

Um die Frage zu beantworten wie man zwei SQL Anfragen miteinander vergleichen kann, muss man sich zunächst die Struktur einer solchen Anfrage betrachten. Exemplarisch betrachten wir im folgenden SELECT Anfragen. Es werden mehrere Ansätze in diesem Teil der Arbeit verfolgt, wie man die Gleichheit von zwei Anfragen zeigen kann. Offensichtlich sind zwei SQL-Anfragen semantisch äquivalent, wenn sie ebenfalls syntaktisch korrekt sind. Interessanter sind daher Anfragen, die zunächst nicht syntaktisch dekungsgleich sind.

Ein Ansatz besteht darin beide SQL-Anfragen einer Standardisierung zu unterziehen. Wie genau so etwas durchgeführt werden kann, wird im Folgenden noch erläutert. Wir würden dann zwei standardisierte SQL-Anfragen erhalten. Sind diese syntaktisch äquivalent, so handelt es sich um identische Anfragen. Dieser Ansatz wird uns mit einigen Problemen konfrontieren und daraus entwickeln wir einen zweiten Ansatz.

Dieser versucht durch gleichartige Umformungen, die zwei Anfragen zu unifizieren (gleich zu machen). Bei diesem Ansatz würden wir also versuchen die geparsten Operatorbäume miteinander zu vergleichen. Auch diese Lösung birgt Vorteile aber auch Probleme mit sich, die im Folgenden besprochen werden.

2.1 Standardisierung von SQL-Anfragen

Es gibt syntaktisch unterschiedliche Anfragen, die jedoch semantisch äquivalent sind. So liefern die folgenden Anfragen die gleichen Ergebnisse, sind aber nicht syntaktisch äquivalent.

```
SELECT * FROM emp e WHERE e.enr > 5
```

```
SELECT * FROM emp e WHERE 5 < e.enr
```

```
SELECT * FROM emp e WHERE e.enr >= 6
```

In diesem kurzem Beispiel sieht man einige Probleme auf die man trifft, wenn man versucht SQL-Anfragen syntaktisch zu standardisieren. Folgende Schritte sind notwendig um eine Standardisierung durchzuführen, welche zum Ziel hat, dass unterschiedliche syntaktischen Anfragen,

die auf allen Datenbankzuständen die gleichen Tupel liefern, zu vereinheitlichen (SATZ UEBERARBEITEN! TODO).

Schritte zur Standardisierung von SQL-Anfragen

1. Entfernung syntaktischer Details wie mehrere Leerzeichen hintereinander, Zeilenumbrueche, Kommentare, unnoetige Klammern: Das macht schon der Parser.
2. Ersetzung von syntaktischen Varianten, z.B. X BETWEEN Y AND Z durch X >= Y AND X <= Z. Hierzu koennte man vielleicht auch die Beseitigung von NOT und die Umwandlung in DNF rechnen. Wahrscheinlich koennte man hier auch unnoetige Schluesselworte entfernen, z.B. SELECT ALL oder ORDER BY ... ASC
3. Hinzufuegen einfacher Implikationen, z.B. transitiv implizierte Gleichungen/Ungleichungen, Wenn A = B AND B = C dann auch A = C, entsprechend mit < und gemischt.
4. Sortierung, Standardisierung

2.2 Entfernen von syntaktischen Details

Das Entfernen von syntaktischen Details übernimmt zum großen Teil bereits der Parser. Er entfernt unnötige Leerzeichen, Kommentare sowie unnötige Klammern. Allerdings treten beim benutzten Parser einige spezielle Eigenheiten auf, die im Bereich » verwendeter SQL-Parser « näher erläutert werden.

2.3 Ersetzung von syntaktischen Varianten

Um eine Anfrage zu standardisieren müssen wir den syntaktischen Zucker entfernen. Dies geschieht, in dem man nur eine syntaktische Schreibweise anerkennt und alle anderen Schreibweisen werden in die zulässige umgewandelt. Zu erwähnen sind folgende Ersetzungen, die durchgeführt werden sollen um syntaktisch vielfältige, aber semantisch äquivalente Ausdrücke zu minimieren.

`A BETWEEN B AND C`	→	`A >= B AND A <= C`
`NOT(A AND B)`	→	`NOT(A) OR NOT(B)`
`A > B`	→	`A >= (B + 1)`, wenn `A` und `B` numerisch
`A < B`	→	`A <= (B - 1)`, wenn `A` und `B` numerisch
...	→	...

Einige andere syntaktische Varianten finden sich im Abschnitt »Sortierung«. Begründet wird das damit, dass `A > B` und `B < A` ebenfalls syntaktische Varianten sind, die sich aber dadurch eliminieren lassen, dass man gewisse Sortierungskriterien festlegt.

Es existieren auch Teile in SQL-Ausdrücken, die redundant bzw. unnötig sind. Ohne diese Angaben würden immer, die gleichen Ergebnisse erzielt werden. Diese Teile blähen SQL-Anfragen nur unnötig auf und erschweren das unifizieren/standardisieren. Das Entfernen solcher Bestandteile soll aber nicht ohne Hinweise geschehen. Der Lernende soll ein Feedback erhalten, dass einige von ihm aufgeschriebene Teile der SQL-Anfrage, diese unnötig verkomplizieren. Dazu gehören folgende Teile:

`SELECT ALL`	→	`SELECT`
`SELECT DISTINCT`	→	`SELECT`, bei unnötigem `DISTINCT` vgl. TODO
`ORDER BY VAR ASC`	→	`ORDER BY VAR`
`A = NULL`	→	`NULL/UNKNOWN/SYNTAX ERROR`
`EXISTS (SELECT A,B,C ...)`	→	`EXISTS (SELECT 1 ...)`
...	→	...

Bei Anwendung dieser Ersetzungsregeln, soll dem Lernenden ein klares Feedback gegeben werden. Es soll verdeutlicht werden, dass eine korrekte Anfrage dennoch Mängel aufweist, da unnötige Formulierungen benutzt wurden.

Eventuell ist es hier auch bereits möglich Terme, die nur aus numerischen Konstanten bestehen zu Ersetzen durch das jeweilige Ergebnisse. So könnten arithmetische Operationen bereits ausgeführt und Vergleiche, die nur aus numerischen Konstanten bestehen, durch entsprechende Wahrheitswerte ersetzt werden.

2.4 Hinzufügen einfacher Implikationen (transitiv)

Die Anfrage des Lernenden kann trotz zahlreicher Umformungen noch einige komplizierte Bedingungen enthalten, die wir mit bisherigen Methoden nicht entdecken konnten. Ein wichtiger Teil dabei sind transitiv-implizierte Bedingungen. Finden wir beispielsweise in der Musterlösung die Bedingung `A = C AND B = C` und der Student schreibt `A = B AND B = C`, so handelt es sich um semantisch äquivalente Aussagen. Damit unser Ansatz funktioniert, ist es also notwendig transitiv-implizierte Formeln immer hinzuzufügen. Wir bilden also im gewissen Sinne den transitiven Abschluss über den Operatoren $\{=, >, <\}$.

Beim Ansatz der Standardisierung mit Sortierung ist ein Betrachten von symetrischen Implikationen unnötig. Wie im Abschnitt Sortierung noch erläutert wird, werden zwei Bedingungen `A = B` und `B = A` nach Sortierungsregeln beide zu `A = B` standardisiert.

2.5 Sortierung (TODO, innerhalb der Terme)

3 Verwendete Software

4 Praktische Umsetzung

5 Ergebnisse

6 Ausblick

Literaturverzeichnis

[St87] L. Staiger, Sequential Mappings of ω-Languages, 1987, pp. 148–170.

[St97] L. Staiger, ω-Languages. in: Handbook of Formal Languages, (G. Rozenberg and A. Salomaa Eds.), Springer-Verlag, Berlin 1997, Vol. 3, pp. 339–387.

[Wg94] K. Wagner, Einführung in die theoretische Informatik, Springer-Verlag, Berlin 1994.

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Halle (Saale), 24. September 2010