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 Forschungsstand und Einordnung

2.1 Einleitung [in das Chapter]

Die Idee SQL-Anfragen von Schülern/Lernenden auszuwerten ist nicht völlig neu. Weil eine Auswertung über den Standard SQL-Parser nicht sehr umfangreich ist, und bei semantischen Fehlern gar kein sinnvolles Feedback gibt, sind bereits einige Ansätze veröffentlicht worden, die es sich zum Ziel gemacht haben eine SQL-Anfrage näher zu analysieren. Verschiedene Projekte beschäftigen sich dabei zum Beispiel mit dem Aufdecken von semantischen Fehlern. Andere Plattformen konzentrieren sich auf den Lernerfolg, den der Student erreichen soll und analysieren die Art der Fehler des Studenten um ihn mit passenderen Aufgaben zu konfrontieren, damit er weder gelangweilt noch überfordert ist. [Anmerkung: Ähnlich einem Art Matchmaking System].

In diesem Abschnitt möchten wir die bereits existierenden Ansätze auf dem Gebiet kurz betrachten um dann diese Arbeit davon abzugrenzen bzw. diese dann einordnen zu können.

2.2 SQL-Tutor

In **??** beschreibt Antonija Mitrovic ein Lernsystem, was SQL-Tutor genannt wird. Nach Auswahl einer Schwierigkeitsstufe wird dem Studenten ein Datenbankschema und eine Sachaufgabe vorgelegt. Der Student hat nun ein Webformular in dem sich für jeden Teil der SQL-Anfrage ein Eingabefeld befindet. So werden `SELECT`, `FROM`, `WHERE`, `ORDER BY`, `GROUP BY` sowie `HAVING` Anteile einzeln eingetragen.

Der SQL-Tutor analysiert nun die Anfrage des Studenten und gibt spezifisches Feedback. Dabei wird nicht nur geklärt, ob die Anfrage korrekt ist, sondern auch, bei einer falschen Eingabe, was genau falsch ist. Das reicht von konkreten Hinweisen auf den spezifischen Teil der Anfrage bis hin zu eindeutigen Hinweisen wie »Musterlösung enthält einen numerischen Vergleich mit der Spalte a, ihre Lösung enthält aber keinen solchen Vergleich«.

Umgesetzt wird dieses Programm durch 199 fest einprogrammierte Constraints. Dadurch ist es potentiell möglich bis zu 199 spezifische Hinweismeldungen für den Studenten bereitzustellen. Das

reicht von syntaktischen Analysen wie »The SELECT Clauses of all solutions must not be empty« bis hin zu semantischen Analysen gepaart mit Wissen über die Domain (Datenbankschema und Musterlösung), bei denen die Lösung des Studenten mit der Musterlösung und dem Datenbankschema verglichen wird. Insbesondere versuch der SQL-Tutor Konstrukte wie numerische Vergleiche mit gewissen Operatoren in der Lösung des Studenten zu finden, wenn diese in der Musterlösung auftauchen. Auch komplexere Constraints, die sicherstellen, dass bei einem numerischen Vergleich `a > 1` das gleiche ist wie `a >= 0` sind vorhanden.

Allerdings gibt es auch hier Schwächen. Da der verwendete Algorithmus die Constraints nach einander abarbeitet, kann es zu unnötigen Analysen der Anfrage kommen und damit auch zu einem Unnötigen Zeitaufwand. Nach eigenen Tests werden manche äquivalente Bedingungen nicht erkannt. So wird `a < 0` für richtig, aber `0 > a` für falsch gehalten. Ähnlich verhält es sich, falls eine der Argumente des Vergleichs das Ergebnis einer Unterabfrage ist.

Der SQL-Tutor lässt außerdem auch den eingesendeten Lösungsvorschlag auf einer SQL-Datenbank mit Testdaten laufen und vergleicht die Tupel mit den Antworttupeln, die man mit der gespeicherten Musterlösung erhält.

Abgrenzung zum SQL-Tutor

Der Grundgedanke des SQL-Tutors überschneidet sich durchaus mit dem Grundgedanken dieser Arbeit. Ein Grundpfeiler des SQL-Tutors ist es, dem Studenten detailliertes Feedback zu geben über seine semantischen und syntaktischen Fehler. Das Programm was im Zuge dieser Arbeit entsteht soll weniger semantische Fehler analysieren, als viel mehr versuchen zwei SQL-Anfragen zu vergleichen und zwar egal wie sie aufgeschrieben sind. Des weiteren bedient sich der SQL-Tutor einer Testdatenbank mit realen Testdaten. Unser Programm soll nur das Datenbankschema kennen und ohne Daten bestimmen, ob zwei Anfragen das gleiche Ergebnis liefern. Damit entfällt für Lehrkräfte ein aufwendiges Ausdenken oder Besorgen von Testdaten. Des weiteren kann es bei ungünstig gewählten Testdaten passieren, dass der Eindruck ensteht zwei Anfragen wären gleich weil sie auf den Testdaten die gleichen Tupel zurück lieferten, auf anderen Testdaten würden aber Unterschiede aufgezeigt werden.

2.3 SQL-Exploratorium

Im Artikel **??** werden SQL-Lernplattformen in zwei Kategorien eingeteilt. Zum einen existieren Plattformen, welche durch Multimedia versuchen dem Lernenden einzelne Bestandteile der Sprache SQL bildlich darzustellen. Hierfür werden meist Websites mit Multimediainhalten erstellt

??,**??**. Die zweite Kategorie beinhaltet Software, welche die Lösung eines Lernenden in analysiert und konkrete Hinweismeldungen gibt. Dazu zählt auch der eben beschriebene SQL-Tutor.

Das SQL-Exploratorium macht es sich nun zur Aufgabe die beiden Ansätze zu verbinden und stellt sich dabei hauptsächlich verwaltungstechnische Fragen wie z.B.:

- Wie ermögliche ich dem Studenten Zugriff auf verschiedene Lernsysteme ohne sich mehrfach einloggen zu müssen?
- Wie können Lernerfolge in einem System einem anderen nutzbar gemacht werden?
- Wie kann man aus mehreren Logfiles der eingereichten Lösungen eines Studenten von unterschiedlichen Systemen einen Wissensstand des Studenten ableiten?

Da die Fragen als solche eher unwichtig für diese Arbeit sind, betrachten wir im Folgenden welche einzelnen Plattformen für das SQL-Exploratorium genutzt werden.

2.3.1 Interactive Examples

Über eine Schnittstelle, die sich WebEX **??** nennt, hat der Student Zugriff auf insgesamt 64 Beispielanfragen. Wählt man eine Anfrage aus können Teile der Anfrage in einer Detailansicht geöffnet werden. Dem Studenten wird dann ausführlich erklärt, was die einzelnen Teile der Anfrage genau bewirken. Sowohl die Beispielanfragen, als auch die Hinweise sind manuell erzeugt und abgespeichert. Hier wird nichts automatisch generiert, daher ist dieses Projekt uninteressant für die Arbeit. Der Lernerfolg des Studenten wird hier über die ein »click-log« geführt, das bedeutet es wird aufgezeichnet, was der Student wann und in welcher Reihenfolge angeklickt hat. So ist es zum Beispiel möglich herauszufinden welche Teile einer bestimmten Anfrage besonders interessant für den Lernenden sind.

Abgrenzung zur Arbeit

Wie bereits erwähnt wird bei den Interactive Examples nichts automatisch erzeugt, was diesen Ansatz für diese Arbeit uninteressant macht.

2.3.2 SQL Knowledge Tester

Der SQL Knowledge Tester, im Nachfolgendem SQL-KnoT genannt, konzentriert sich darauf Anfragen eines Studenten zu analysieren. Dabei wird dem Studenten zur Laufzeit eine Frage generiert. Dabei werden vorhandene Datenbankschemata in einer bestimmten Art und Weise verknüpft

und Testdaten so wie eine Frage für den Studenten generiert. Dies geschieht mit fest einprogrammierten 50 Templates, die in der Lage sind über 400 Fragen zu erzeugen. Zu jeder Frage werden zur Laufzeit Testdaten für die relevanten Datenbanken erzeugt. Ausgewertet wird die Anfrage des Studenten dann, in dem die zurückgelieferten Tupel mit der Studentenanfrage verglichen werden mit den Tupeln, welche die Musterlösung erzeugt.

Abgrenzung zur Arbeit

Erwähnenswert ist, dass initial keine Daten existieren. Wie beim Ansatz dieser Arbeit existieren nur Datenbankschemata. Die Daten und auch die Aufgabe an den Studenten werden aus Templates generiert. Die Auswertung erfolgt dann allerdings durch Vergleich der zurückgelieferten Tupel der Muster- und Studentenanfrage. Hierbei kann wieder das Problem auftreten, dass für beide Anfragen für die erzeugten Testdaten die gleichen Tupel zurückliefern, es bei einem anderen – nicht erzeugtem – Zustand sein kann, dass sich die Tupelmengen unterscheiden.

Der Ansatz vom SQL-KnoT ist durchaus interessant, wird aber in dieser Arbeit nicht weiter ausgeführt, da diese keine Testdaten erzeugen möchte, sondern gänzlich ohne Daten auskommen will.

2.3.3 Weiteres

Adaptive Navigation for SQL Questions

Hierbei handelt es sich nur um ein Tool, was Aufgrund früherer Antworten des Studenten, diesem möglichst passende neue Fragen vorlegen möchte. Dieser Teil des SQL-Exploratoriums dient also dazu, den Wissensstand des Studenten festzustellen und ist für diese Arbeit daher unerheblich.

SQL-Lab

SQL-Lab ist lediglich ein Hilfsmittel um SQL-KnoT zu benutzen und daher für diese Arbeit auch nicht von Bedeutung.

2.4 WIN-RBDI

Das Programm WINRBDI, welches in **??** beschrieben wird verfolgt einen weiteren, interessanten Ansatz. Anstelle von fest vorgegebenen Demoanfragen, wird die eingegebene Anfrage zunächst

in esql eingebettet. Die Ausführung der Anfrage wird dann Stück für Stück durchgeführt. Der Student hat also die Möglichkeit die Anfrage im Schrittmodus – ähnlich eines Debugger – oder im Fortsetzen-Modus auszuführen. Im Schrittmodus wird jeder Teilschritt der Abarbeitung der Anfrage aufgezeigt. Es werden temporär erzeugte Tabellen angegeben, so wie auch eine Erklärung welcher Teil der Anfrage für den aktuellen Abarbeitungsschritt verantwortlich ist. So soll es dem Studenten möglich sein, die unmittelbaren Konsequenzen seiner SQL-Anfrage für die Abarbeitung zu begreifen.

Des weiteren hilft dieser Ansatz dem Studenten die Abarbeitung einer Anfrage zu Visualisieren, in dem von der WHERE Klausel betroffene Spalten markiert werden. Dies hilft gerade Lernanfängern bei der Visualisierung von Konzepten wie JOINs.

Abgrenzung zur Arbeit

Dieser Ansatz hebt sich von den bisherig betrachteten Ansätzen ab. Hier wird dem Studenten durch eine Visualisierung der Ausführung der Anfrage versucht deutlich zu machen, welche Teile der formulierten Anfrage was genau bewirken. Für den Lernerfolg des Studenten ist dies sicherlich hilfreich, zumal eine Visualisierung stets hilft Zusammenhänge zu begreifen, jedoch verfolgt diese Arbeit ein ganz anderes Ziel, da sie zwei SQL-Anfragen miteinander vergleicht und nicht versucht die Abarbeitung einer Anfrage zu visualisieren.

3 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.

3.1 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

3.1.1 Entfernen von syntaktischen Details

Das Entfernen von syntaktischen Details übernimmt zum großen Teil bereits der Parser. Er entfernt unnötige Leerzeichen, Kommentare sowie unnötige Klammern. Aufgrund der Arbeitsweise des Parsers gibt es allerdings Situationen, in dem der Parser scheinbar nicht alle unnötigen Klammern entfernt. Wie im Abschnitt »Verwendeter Parser« erläutert wird, sind die geparsten Bäume nicht binär. Ein Baum wie in Abbildung 4.1.2 zu sehen, ist daher zu vermeiden.

Es ist daher wünschenswert, wenn ein Operator X einen Ausdruck als Kindknoten besitzt, in dem X ebenfalls der Operator ist, den Operator X im Kindknoten zu eliminieren und alle Kinder vom eliminierten Kindknoten an den verbleibenden Operatorknoten X zu hängen. Damit hätte man den Ausdruck vereinfacht, da die assoziative Klammerung wegfällt. Wir nennen dieses Vorgehen im Folgenden Operatorkompression.

Gegeben sei der ZQL-Parsebaum $B = (V, E)$. Es sei $child(v) = \{w : w \in V \wedge (v, w) \in E\}$, also die Menge aller Kindknoten von v. Gibt es einen Knoten $w \in child(v)$ mit $v = w$, so wird Knoten w eliminiert und alle Kindknoten von w werden zu Kindknoten von v, also $child(v) = child(v) \cup child(w)$. $E = E \backslash \{(w, x) : x \in child(w)\} \cup \{(v, x) x \in child(w)\}$ und $V = V \backslash \{w\}$.

Mit dem Parser und der Operatorkompression können alle unnötigen Klammern entfernt werden. Es ist für die Auswertung des Studenten wichtig abzuspeichern wie oft die Operatorkompression durchgeführt wurde. Es ist zwar kein harter Fehler, aber für die Lösung waren die Klammern

einfach unnötig und dies muss dem Studenten auch mitgeteilt werden, selbst wenn seine Lösung mit der Musterlösung am Ende der Standardisierung übereinstimmt.

3.1.2 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.

3.1.3 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.

3.1.4 Sortierung)

Im aktuell betrachteten Ansatz möchten wir zwei Anfragen dadurch vergleichen, dass wir sowohl die Musterlösung, als auch die Studentenlösung einer Standardisierung unterziehen. Ein ganz wesentlicher Aspekt dabei ist, die Art der Sortierung. Sind die ZQL-Parserbäume isomorph zueinander, dann lässt sich das leicht zeigen, in dem man beide nach gleichartigen Kriterien sortiert und dann einen direkten Abgleich vornimmt.

Dabei unterscheiden wir zwei Arten von Sortierung. Hat ein Operator als Operanden nur Ausdrücke und keine Konstanten oder Variablen dann sortieren wir die Kindknoten, welche jeweils wieder eigene Terme bilden.

Hat ein Operator als Operand mindestens eine Konstante oder Variable, so Sortieren wir das innere dieses Terms.

Sortierung von Termen

Hat ein Operator *OP1* als Kindknoten nur weitere Operatoren *OP2,OP3*, dann muss anhand dieser Operatoren die Reihenfolge im Baum festegelegt werden. Dies geschieht, in dem wir uns einfach eine Reihenfolge der Operatoren ausdenken. Wir überlegen uns folgende Ordnung $order : Relation \rightarrow \mathbb{N}$, in der eine Relation r vor einer Relation s im standardisierten Parserbaum erscheint, wenn $order(r) < order(s)$.

$order$:

<=	>=	=	IS NULL	IS NOT NULL	OR	AND
1	2	3	4	5	6	7

Sortierung im Inneren der Terme

3.2 Anpassung durch elementare Transformationen

s

4 Verwendete Software

4.1 SQL Parser

4.1.1 über den SQL Parser: ZQL

Auf der Webseite vom [ZQL] Projekt ist der Open-Source-Parser ZQL zu finden, welcher in der Lage ist SQL zu parsen und in Datenstrukturen zu überführen. Der Parser selbst ist mit [JavaCC] geschrieben, einem Java-Parsergenerator (zu vergleichen mit dem populärem Unix yacc Generator).

ZQL bietet Unterstützung für `SELECT-`,`INSERT-`,`DELETE-`,`COMMIT-`,`ROLLBACK-`,`UPDATE-` und `SET TRANSACTION`-Ausdrücke. Wichtig für diese Arbeit sind dabei insbesondere `SELECT-` und `UPDATE`-Ausdrücke, sowie – die leider nicht enthaltenen – `CREATE TABLE`-Ausdrücke.

4.1.2 Funktionsweise des Parsers

ZQL kennt zwei grundlegende Interfaces `ZExp` und `ZStatement`.

Das Interface `ZStatement` bildet eine abstrakte Oberklasse für alle möglichen Arten von SQL-Statements. Folgende Klassen implementieren dieses Interface in ZQL:

- `ZDelete` - repräsentiert ein `DELETE` Statement
- `ZInsert` - repräsentiert ein `INSERT` Statement
- `ZUpdate` - repräsentiert ein `UPDATE` Statement
- `ZLockTable` - repräsentiert ein `SQL LOCK TABLE` Statement
- `ZQuery` - repräsentiert ein `SELECT` Statement

Das Interface `ZExp` bildet eine abstrakte Oberklasse für drei verschiedene Arten von Ausdrücken:

- `ZConstant` - Konstanten vom Typ `COLUMNAME, NULL, NUMBER, STRING` oder `UNKNOWN`
- `ZExpression` - Ein SQL-Ausdruck bestehend aus einem Operator und einen oder mehreren Operanden

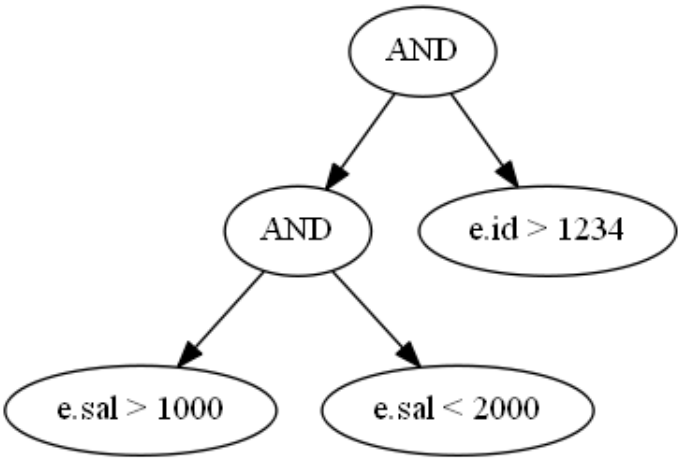

Abbildung 4.1: WHERE-Bedingung in üblichen Syntaxbäumen

- `ZQuery` - Eine `SELECT` Anfrage ist auch ein Ausdruck

Da die `SELECT` Anfragen die wohl am häufigsten gebrauchte Form der Anfragen ist, wird sich die Erklärung der Funktionsweise des Parsers beispielhaft auf diese Art der Anfragen beziehen. Wie die anderen Statements geparst werden ist dann analog schnell zu verstehen.

Eine gewöhnliches Select-Statement wird wie folgt vom Parser zerlegt:

```
SELECT e.name FROM emp e WHERE e.sal > 1000 ORDER BY e.sal DESC
```

`SELECT e.name`	*Vector* von *ZSelectItem* enthält `e.name`
`FROM emp e`	*Vector* von *ZFromItem* enthält `emp` mit Alias `e`
`WHERE e.sal > 1000`	*ZExpression* mit Operator > und Operanden*Vector* der Form {`e.sal`, `1000`}
`ORDER BY e.sal DESC`	*ZOrderBy*-Objekt mit enthaltenem `ORDER BY` Sortierausdruck und Reihenfolge

Eine Besonderheit des ZQL-Parsers sind seine Parserbäume. Ein üblicher Syntaxbaum ist binär, wobei die Wurzel den Operator mit der höchsten Priorität darstellt. Alle Teilbäume sind wieder als Ausdrücke zu verstehen, jeweils mit Operator als Wurzelknoten und Operanden als Kindknoten. Dabei kann ein Operand auch ein weiterer Ausdruck sein. Generell wird dabei das Prinzip der Assoziativität benutzt um z.B.: für gleichrangige Operatoren eine Auswertungsreihenfolge festzulegen.

So würde der `WHERE`-Teil von folgender SQL-Anfrage:

```
SELECT * FROM emp e WHERE e.sal > 1000 AND e.sal < 2000 AND e.id > 1234
```

zu folgendem geklammerten Ausdruck:

```
(((e.sal > 1000) AND (e.sal < 2000)) AND (e.id > 1234))
```

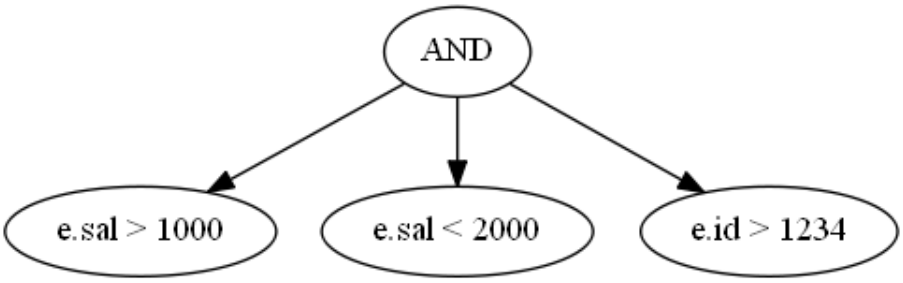

Abbildung 4.2: WHERE-Bedingung geparst mit ZQL

Der ZQL-Parser funktioniert so allerdings nicht. Wird keine spezielle Klammerung benutzt so werden gleichrangige Operatoren nicht assoziativ geklammert, sondern befinden sich auf einer Ebene des Baumes. Somit handelt es sich nicht um einen binären Baum.

Wir erhalten also aus obigen `WHERE`-Ausdruck:

```
((e.sal > 1000) AND (e.sal < 2000) AND (e.id > 1234))
```

Wie schon erwähnt werden Operanden daher in einer *Vector* Struktur gespeichert.

4.1.3 Grenzen des Parsers

Der Parser kann keine `CREATE TABLE` Statements parsen. Somit ist es im Rahmen dieser Arbeit notwendig, den Parser zu erweitern, damit Tabellen in eigene Datenstrukturen geparst werden können. Für die Arbeit ist es zunächst nur notwendig Name und Datentyp der Spalten in eine interne Datenstruktur zu überführen. Dabei wird nur zwischen Zahlen und Sonstigem (Text) unterschieden. Unser Programm soll in der Lage sein, einfache arithmetische Operationen durchzuführen. Dazu ist das wissen um Datentypen der Variablen von Nöten.

4.2 Java Server Pages

4.2.1 Überblick

4.2.2 Einbettung in JSP

4.2.3 Log

5 Praktische Umsetzung

6 Ergebnisse

7 Ausblick

Literaturverzeichnis

[JavaCC] http://www.javacc.org

[ZQL] http://zql.sourceforge.net/

15

15

Hiermit versichere ich, dass ich die Abschlussarbeit bzw. den entsprechend gekennzeichneten Anteil der Abschlussarbeit selbständig verfasst, einmalig eingereicht und keine anderen als die angegebenen Quellen und Hilfsmittel einschließlich der angegebenen oder beschriebenen Software benutzt habe. Die den benutzten Werken bzw. Quellen wörtlich oder sinngemäß entnommenen Stellen habe ich als solche kenntlich gemacht.

Halle (Saale), 27. August 2013