HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Ein System zur Lernunterstützung und Bewertung für die relationale Algebra

Firas Shmit

Bachelor-Arbeit im Studiengang „Angewandte Informatik“

26. Februar 2021

Autor: Firas Shmit
1346981
Firas.Shmit@stud.hs-hannover.de

Erstprüferin: Prof. Dr. Felix Heine
Abteilung Informatik, Fakultät IV
Hochschule Hannover
Felix.Heine@hs-hannover.de

Zweitprüfer: Prof. Dr. Carsten Kleiner
Abteilung Informatik, Fakultät IV
Hochschule Hannover
Carsten.Kleiner@hs-hannover.de

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelor-Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 26. Februar 2021 Unterschrift

Inhaltsverzeichnis

1. Einleitung

In diesem Kapitel werden die Motivation und die Zielsetzung dieser Arbeit erklärt. Es wird außerdem die Struktur dieser Arbeit erläutert.

1.1. Motivation

Das E-Learning ist eine sehr verbreitete unterstützende Methode für den Lernprozess. Zudem hat sich E-Learning in den vergangenen Jahren stark bewiesen und es wird in der Zukunft mehr an Bedeutung gewinnen. Dies liegt zum einen daran, dass E-Learning das selbstständige Lernen fördert und zum anderen ermöglicht es ein ortsunabhängiges, flexibles Arbeiten. Weiterhin bietet E-Learning ein interaktives Umfeld und steigert die Effizienz des Lehrprozesses. Vor diesem Hintergrund fehlt es im Bereich des E-Learnings an der Hochschule Hannover an ein Auswertungssystem zur Unterstützung des Lernens der relationalen Algebra im Rahmen der Datenbankvorlesung. Aufgrund der aktuellen Situation der Corona-Pandemie müssen Studierende vorwiegend von zu Hause lernen. Dadurch wird deutlich wie stark solch ein System Abhilfe verschaffen würde.

In dieser Bachelorarbeit wird ein Evaluationssystem für die Hochschule Hannover entwickelt, welches zur automatischen Auswertung von Aufgaben der relationalen Algebra eingesetzt wird. Zielgruppe des Systems sind Studierende der Datenbankvorlesungen, die im Rahmen der Lehrveranstaltung die relationale Algebra erlernen. Das Evaluationssystem unterstützt einerseits die Studierenden durch detaillierte und lernunterstützende Feedbacks beim Erlernen der relationalen Algebra. Andererseits ermöglicht die Software dem Dozenten bzw. der Dozentin, Aufwand bei der Korrektur von Übungsaufgaben einzusparen.

1.2. Zielsetzung und Erkenntnisinteresse

Das Ziel dieser Arbeit ist es ein Auswertungssystem zu entwickeln, welches studentische Aufgabenstellungen aus dem Bereich der relationalen Algebra automatisch auswertet und in Form einer Punktevergabe bewertet. Das System soll die studentische Lösung

mit der Musterlösung vergleichen und auf Basis dieses Vergleichs, ausführliche und lernunterstützende Feedbacks generieren können. Dadurch sollen Studenten eine korrekte und optimierte Sprachanwendung im Bereich der Relationalen Algebra lernen. Weiteres Ziel ist die Optimierung und Implementierung der Grundidee des Ergebnisvergleiches des existierenden Systems aSQLg [JBo17].

1.3. Struktur der Arbeit

Nachdem die Motivation, der wissenschaftliche Hintergrund und das Ziel dieser Arbeit dargelegt wurden, werden im Kapitel 2 zunächst einmal wichtige Grundlagen, darunter Konzepte und Begriffe, das Tool ANTLR und seine Komponenten vorgestellt.

Kapitel 3 befasst sich mit der Entwicklung des Auswertungssystems von Ausdrücken der relationalen Algebra. Zunächst wird die Ausgangssituation und die Idee dieser Arbeit erläutert. Außerdem wird der Entwurf der Syntax für relationale Ausdrücke mit dem Parsergenerator ANTLR und die Implementierung des Auswertungssystems dargelegt.

Kapitel 4 befasst sich mit dem strukturellen, sowie dem inhaltlichen Vergleich der Musterlösung mit der studentischen Lösung. Darüber hinaus wird in diesem Kapitel die Implementierung des Vergleichs erklärt.

In Kapitel 5 wird das Assessment behandelt, welches aus Punktevergabe und lernunterstützenden Feedbacks besteht. Dazu werden zur besseren Erklärung Beispiele genannt. Anschließend wird die Generierung des Endberichts der Lösung und der dazugehörigen Feedbacks dargestellt.

In Kapitel 6 wird das Testen des Systems mit der Test-Driven-Development Methode vorgestellt. Zudem werden einige Testfälle dargestellt, mit denen die Funktionsweise des Systems gezeigt wird.

In Kapitel 7 wird schließlich ein Fazit gezogen und ein Ausblick auf die zukünftige Entwicklung des Systems gegeben.

2. Grundlagen

In diesem Kapitel werden einige Konzepte und Begriffe erklärt, die sich auf diese Arbeit beziehen. Darüber hinaus wird der Parsergenerator 'ANTLR' und seine Funktionsweise vorgestellt.

2.1. Begriffe

Im folgenden werden die Definitionen einer Relation und der relationalen Algebra erläutert. Zudem werden die Grundoperationen der relationalen Algebra mit Beispielen erklärt.

2.1.1. Relation

Relation: Stellt durch Spalten, Zeilen und ein eigenes Schema Daten der realen Welt dar. Das Schema einer Relation beinhaltet den Namen der Relation sowie die dazugehörigen Attribute. Die Relation ist die einzige Datenstruktur des relationalen Modells. [Hei18]

Die Attribute einer Relation definieren den Typ eines möglichen Attributwertes. Jeder Datensatz einer Relation wird Tupel genannt. Jedes Tupel besteht in der Regeln aus einer Kombination von Attributwerten und repräsentiert damit einen einzelnen Datensatz.

2.1.2. Relationale Algebra

Relationale Algebra oder Relationenalgebr: Ist eine Abfragesprache des relationalen Modells. Sie enthält eine Menge von Operationen zur Manipulation von Relationen. Sie ermöglicht es, Relationen zu filtern, zu modifizieren, zu aggregieren oder zu verknüpfen, um komplexe Anfragen an eine relationale Datenbank zu formulieren. [Mic12]

2.1.3. Grundoperationen der relationalen Algebra

Die Operationen der relationalen Algebra lassen sich wie folgt klassifizieren:[DrM]

- Mengenorientiere Operationen: Vereinigung, Schnittmenge, Differenz.
- Relationenorientierte Operationen: Selektion, Projektion.
- Kombinierende Operatoren: Kartesisches Produkt, Join, Joinvarianten.

Im Weiteren werden die Operationen der relationalen Algebra mit Beispielen erläutert: Gegeben seien die Relationen Mitarbeiter und Studenten mit den in Tabelle 2.1, sowie Tabelle 2.2 gezeigten Daten.

Mitarbeiter	
Name	Vorname
Huber	Maria
Mayer	Josef
Schmidt	Alex

Tabelle 2.1.: Mitarbeiter

Studenten	
Name	Vorname
Wolfgang	Maria
Müller	Heinz
Schmidt	Alex

Tabelle 2.2.: Studenten

Vereinigung (`UNION` $\cup$) : Die Vereinigung zweier Relationen sammelt Tupel aus beiden Relationen unter einem gemeinsamen Schema auf. Dabei müssen Attributmengen (Spaltenanzahl, Datentypen und Reihenfolge) beider Relationen identisch sein. Eine Eigenschaft der Vereinigung ist die Duplikatentfernung der Tupel, die jeweils einmal in beiden Relationen vorkommen.

`Beispiel: Mitarbeiter` $\cup$ `Studenten` ergibt 2.3

Mitarbeiter $\cup$ Studenten	
Name	Vorname
Huber	Maria
Mayer	Josef
Schmidt	Alex
Wolfgang	Maria
Müller	Heinz

Tabelle 2.3.: (Mitarbeiter $\cup$ Studenten)

Differenz (`-`) : Die Differenz R - S zweier Relationen eliminiert die Tupel aus der ersten Relation R, die auch in der zweiter Relation S vorkommen.
`Beispiel: Mitarbeiter - Studenten` ergibt 2.4

Mitarbeiter - Studenten	
Name	Vorname
Huber	Maria
Mayer	Josef

Tabelle 2.4.: (Mitarbeiter - Studenten)

Schnittmenge (`UND` $\cap$): Ergibt die Tupel, die in beide Relationen gemeinsam vorkommen.
`Beispiel: Mitarbeiter` $\cap$ `Studenten` ergibt 2.5

Mitarbeiter $\cap$ Studenten	
Name	Vorname
Schmidt	Alex

Tabelle 2.5.: (Mitarbeiter $\cap$ Studenten)

Selektion (σ): Wird über einer Eingangsmenge (Relation) ausgeführt. Die Selektion dient dazu, bestimmte Tupel (Zeilen) aus einer bestehenden Relation auszuwählen. Das Ergebnis einer Selektion ist wieder eine Relation mit dem gleichem Schema, nämlich die Ergebnismenge von Tupeln. Hierfür wird das Selektionsprädikat ausgewertet. Dieses kann mehrere Bedingungen enthalten, die über logische Operatoren verbunden sind. Die Selektion kann nicht durch eine Kombination aus anderen Operationen erzeugt werden.

Selektion der Mitarbeiter, die in der Abteilung mit der Nummer 01 arbeiten:

`Beispiel:` $\sigma_{ANr=01}$ `(Mitarbeiter)` ergibt 2.6

PNr	Name	Vorname	Abteilung
001	Huber	Maria	01

Tabelle 2.6.: $\sigma_{ANr=01}$ (Mitarbeiter)

Mitarbeiter			
PNr	Name	Vorname	Abteilung
001	Huber	Maria	01
002	Mayer	Josef	02
003	Schmit	Alex	02
004	Mayer	Anton	03

Tabelle 2.7.: erweiterte Mitarbeiter Relation

Abteilungen	
ANr	Abteilungsname
01	Personal
02	Buchhaltung
03	Produktion

Tabelle 2.8.: Abteilungen Relation

Projektion (Π): Mit der Projektion lassen sich Attribute (Spalten) aus einer bestehenden Relation beliebig auswählen und anordnen.

Projektion der Name und Abteilung Spalten aus der Mitarbeiter-Relation:

`Beispiel:` $\pi_{Name,Abteilung}$ `(Mitarbeiter)` ergibt 2.9

Name	**Abteilung**
Huber	01
Mayer	02
Schmit	02
Mayer	03

Tabelle 2.9.: $\pi_{Name,Abteilung}$ (Mitarbeiter)

Kartesisches Produkt (`Kreuzprodukt` ×): ist ein binärer Operator. Das Ergebnis des Kreuzproduktes zweier Relationen R und S ist die Menge aller Tupel, die man erhält, wenn man jedes Tupel aus R mit jedem Tupel aus S paart.
`Beispiel: (Mitarbeiter` × `Studenten)` ergibt 2.10

$(Mitarbeite \times Abteilungen)$					
PNr	Name	Vorname	Abteilung	ANr	Abteilungsname
001	Huber	Maria	01	01	Personal
001	Huber	Maria	01	02	Buchhaltung
001	Huber	Maria	01	03	Produktion
002	Mayer	Josef	02	01	Personal
002	Mayer	Josef	02	02	Buchhaltung
002	Mayer	Josef	02	03	Produktion
003	Schmit	Alex	02	01	Personal
003	Schmit	Alex	02	02	Buchhaltung
003	Schmit	Alex	02	03	Produktion
004	Mayer	Anton	03	01	Personal
004	Mayer	Anton	03	02	Buchhaltung
004	Mayer	Anton	03	03	Produktion

Tabelle 2.10.: $(Mitarbeite \times Abteilungen)$

Durch die Paarung aller Datensätze der beiden Relationen ergeben sich Datensätze mit unsinnigem Inhalt, die aussortiert werden müssen. So ist z.B. der zweite Datensatz in der Ergebnisrelation "Maria Huberärbeitet in der Personalabteilung mit der Abteilungsnummer 01. Durch die Verwendung des kartesischen Produkts ist aber ein Datensatz entstanden, in dem Maria Huber der Buchhaltung mit der Abteilungsnummer 02 zugeordnet wurde, was in der Realität nicht stimmt. Um dieses Problem zu lösen, muss eine Bedingung hinzugefügt werden, bei der nur die Datensätze gepaart werden, die diese Bedingung erfüllen. Dies wäre, dass die Abteilung in der Relation Mitarbeiter gleich ANr in der Relation Abteilung sein muss. Dies kann durch ein Selektion-Prädikat realisiert werden.

`Beispiel:` $\sigma_{Abteilung=ANr}(Mitarbeiter \times Abteilungen)$ 2.11

(Mitarbeiter × Abteilungen)					
PNr	Name	Vorname	Abteilung	ANr	Abteilungsname
001	Huber	Maria	01	01	Personal
002	Mayer	Josef	02	02	Buchhaltung
003	Schmit	Alex	02	02	Buchhaltung
004	Mayer	Anton	03	03	Produktion

Tabelle 2.11.: $(Mitarbeiter \times Abteilungen)$

Natürlicher Join ($\bowtie$): ist eine Erweiterung des kartesischen Produktes. Aber anstatt alle Paare zu bilden, werden hier Tupelpaare gefiltert. Diese Einschränkung basiert nur auf gleichen Spaltennamen.

`Beispiel:` $(Mitarbeiter \bowtie_{Abteilung=ANr} Studenten)$ 2.12

$(Mitarbeite \bowtie_{Abteilung=ANr} Abteilungen)$					
PNr	Name	Vorname	Abteilung	ANr	Abteilungsname
001	Huber	Maria	01	01	Personal
002	Mayer	Josef	02	02	Buchhaltung
003	Schmit	Alex	02	02	Buchhaltung
004	Mayer	Anton	03	03	Produktion

Tabelle 2.12.: $(Mitarbeite \bowtie_{Abteilung=ANr} Abteilungen)$

Umbenennung(ρ): Der Umbenennungsoperator ändert den Namen einer Relation A zu B und kann ebenfalls den Namen eines Attributes x zu y in einer bestimmten Relation ändern. Eine solche Umbenennung könnte erforderlich sein, um die Durchführung des natürlichen Joins, um die semantisch korrekten Attribute namensgleich zu haben.

Die Schreibweise des Umbenennungsoperator ist: $\rho_{alt \to neu}$

Das folgende Beispiel zeigt die Erforderlichkeit der Nutzung des Umbenennungsoperators mit dem natürlichen Join. Es sei folgende Mitarbeiter Relation gegeben und die Abteilung Relation.

Mitarbeiter					
M_ID	Name	Vorname	Ort	Abteilung_ID	Gehalt
112	Huber	Maria	Köln	10	3000,00
113	Mayer	Josef	Hannover	20	2400,00
114	Müller	Max	Hamburg	30	3100,00

Tabelle 2.13.: Mitarbeiter

Abteilung		
Abt_ID	Bezeichnung	Ort
10	Einkauf	Köln
20	Verkauf	Hannover
30	Produktion	Hamburg

Tabelle 2.14.: Abteilung

2.2. Tools

In diesem Kapitel werden die Extended Backus-Naur-Form Syntax und der Abstract Syntax Tree, kurz AST, anhand von Beispielen erläutert. Außerdem wird der Parser-Generator ANTLR und seine Komponenten vorgestellt.

2.2.1. Extended Backus-Naur-Form

Die erweiterte Backus-Naur-Form, kurz EBNF, ist eine häufig verwendete Notation für kontextfreie Grammatiken. Dies wurde von J.Backus und P.Naur erfunden. Ein wesentliches Merkmal dieser Form ist die wechselseitig rekursive Definition der syntaktischen Kategorien einer Programmiersprache. ANTLR verwendet eine spezielle Variante der EBNF, bei der die Operatoren für Wiederholung, Verkettung, Option usw. durch Operatoren ersetzt wurden, die üblicherweise in regulären Ausdrücken verwendet werden. Die Tabelle 2.15 stellt einige Ausdrücke der ANLTR EBNF Variante und deren Beschreibung dar:

ANTLR	**Beschreibung**
X Y	X gefolgt von Y (ohne Komma)
X | Y	X oder Y
X?	null oder ein Vorkommen von X
X+	ein oder mehr Vorkommen von X
X*	null oder mehr Vorkommen von X

Tabelle 2.15.: Ausdrücke der ANTLR EBNF-Variante [AJA10]

Beispiel:

```
grammar Example;
start: Hello ID;
ID:
    [A-Za-z0-9]+;
WS:
    [ \t\r\n]+ -> skip;
```

Abbildung 2.1.: Beispiel einer ANLTR-EBNF Syntax

Sei „Hello World“ eine Eingabe dieser Grammatik, dann sieht der AST-Baum dieser Eingabe wie folgt aus:

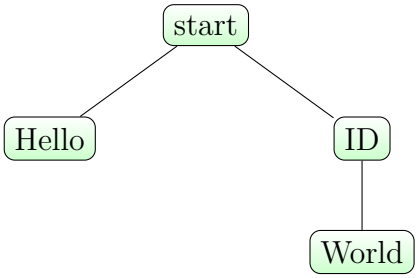


Abbildung 2.2.: AST für den Satz Hello there

Das Schlüsselwort Grammar definiert die Grammatik „Example“. Hierbei muss der Dateiname Example.g4 heißen. Die Start Regel beschreibt die grundlegendste Regel der Grammatik, mit der ANTLR anfängt, zu parsen. Diese besteht aus einem Schlüsselwort `Hello` gefolgt von einem Bezeichner `ID`. Die Regel `ID` kann mit einem regulären Ausdruck [A-Za-z0-9] dargestellt werden, der zur Erkennung vom Bezeichner in Kleinbuchstaben dient. Das + Zeichen nach dem Ausdruck sorgt dafür, dass die Bezeichner ID mindestens aus einem Kleinbuchstaben besteht.

2.2.2. Another Tool for Language Recognition

ANTLR (Another Tool for Language Recognition): ist ein Parser-Generator, der eine Sprache in einer EBNF-ähnlichen Syntax parsen kann und einen C, C++, Java, Python, C# oder Objective-C-Quellcode für einen Parser generiert, der die betreffende Sprache parst. Die EBNF-Sprachbeschreibung kann mit dem Quellcode, der den Parser Prozess leitet, annotiert werden. ANTLR unterscheidet drei Kompilierungsphasen: die lexikalische Analysephase, die Parsing-Phase und Tree-Walking Phase. [Par89]

Lexikalische Analysator (Lexer):
Der lexikalische Analysator ist für die Umwandlung des Zeichenstroms in einen Tokenstrom verantwortlich. Token sind logisch zusammenhängende Komponenten. Für ANTLR gibt die Regeln des lexikalischen Analysators genau an, wie Token erkannt werden können. Die lexikalische Analyse ist ein grundlegendes Verfahren, das vor dem Parsen geschieht.

Parser:
Der Parser ist dafür verantwortlich, aus dem Tokenstrom, den der lexikalische Analysator erzeugt hat, die initiale AST zu erstellen. Um die Struktur des Baumes zu bestimmen, wird die EBNF-Syntax für die Parser Regeln verwendet.

AST Abstract Syntax Tree

Ein abstrakter Syntaxbaum oder AST ist ein gerichteter beschrifteter Baum, bei dem jeder Knoten ein Konstrukt der Programmiersprache repräsentiert. Die Kinder jedes Knotens stellen die Komponenten des Konstrukts dar. Ein AST-Abschnitt ist ein Teil des ASTs, der durch genau eine Regel einer Grammatik gebildet wurde. Die folgende Abbildung zeigt einen einfachen AST zu dem Ausdruck `x = 2 + 4`

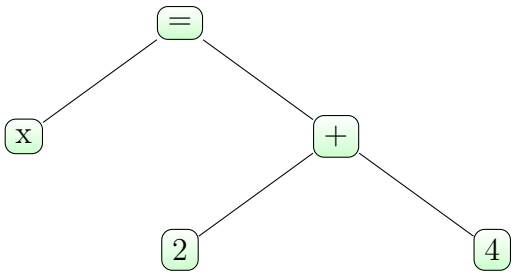


Abbildung 2.3.: AST (Abstract Syntax Tree) für x = 2 + 4

Typischerweise verwendet der Compiler den lexikalischen Analysator (Lexer) um die Eingabe zu analysieren. Der lexikalischer Analysator ist dafür verantwortlich, die Zeichen aus der Eingabe oder aus dem Zeichenstrom, in einen Strom von Wörtern oder Token zu gruppieren. Dies wird als Eingabe an den Parser verwendet. Der Parser analysiert diesen Tokenstrom und erstellt einen Baum von Token, der als abstrakter Syntaxbaum oder AST bezeichnet wird.

Lexer Grammatik

Lexer Regeln: Diese Regeln definieren die Token, die die Grundlage für die Parser Regeln bilden. Die Token werden jeweils über eine Lexer-Regel definiert. Eine Lexer-Regel kann verknüpft werden mit:

- einer einzelnen literalen Zeichenfolge, die in der Eingabe erwartet wird
- einer Auswahl von literalen Zeichenfolgen, die gefunden werden können
- einer Folge von bestimmten Zeichen und Zeichenbereichen unter Verwendung der Kardinalitätsindikatoren (?, * und +) (Abbildung 2.4)

```
ID:
    [A-Za-z0-9]+;
WS:
    [ \t\r\n]+ -> skip;
```

Abbildung 2.4.: Definition eines Tokens über eine Lexer-Regel

Ein Beispiel für Lexer Regeln sind die beiden Zeilen aus dem Beispiel in der Abbildung 2.1. A-Z steht für einen beliebigen Buchstaben zwischen A und Z, a-z für einen beliebigen Buchstaben zwischen a und z, während 0-9 eine beliebige Zahl zwischen 0 und 9 bezeichnet. Standardmäßig tokenisiert der Lexer alles einschließlich Leerzeichen. Das bedeutet, dass das Leerzeichen an jeder Stelle, wo es verwendet wird, in der Grammatik definiert werden muss. Daher definiert man ein Whitespace-Token WS, das aus einem oder mehreren Leerzeichen, Tabulatoren und Zeilenumbrüchen besteht und ANTLR diese überspringen lässt.

Eine Lexer-Grammatik besteht aus Lexer-Regeln, die optional in mehrere Modi unterteilt werden können. Lexikalische Modi erlauben es, eine einzelne Lexer-Grammatik in mehrere Sublexer aufzuspalten. Der Lexer kann nur Token zurückgeben, die mit Regeln aus dem aktuellen Modus übereinstimmen. Lexer-Regeln spezifizieren Token-Definitionen und folgen mehr oder weniger der Syntax von Parser-Regeln, mit der Ausnahme, dass Lexer-Regeln keine Argumente, Rückgabewerte oder lokale Variablen haben können. Außerdem sind Lexer-Regeln von Parser-Regeln unterscheidbar, durch einen beginnenden Großbuchstaben des Regelnamens. Parser-Regelnamen beginnen mit einem Kleinbuchstaben.

Parser Grammatik

Parser Regeln: beginnen mit einem Kleinbuchstaben und helfen dabei, einen abstrakten Syntaxbaum bzw. AST aus den geparsten Token aufzubauen. Ein Beispiel für eine Parser Regel ist in der Abbildung 2.5 dargestellt

```
start: Hello ID;
```

Abbildung 2.5.: Definition einer Parser Regel

Parser Grammatik bestehen aus einem Satz von Parser Regeln entweder in einer reinen Parser Grammatik oder einer kombinierten Grammatik von Parser- und Lexer-Regeln. Die grundlegendste Regel ist ein Regelname, gefolgt von einer einzigen Alternative, die mit einem Semikolon abgeschlossen wird. Regeln können auch Alternativen haben, die durch das | getrennt werden.

In der Abbildung 2.6 ist der Workflow von ANTLR dargestellt. Auf der Abbildung ist es zu sehen, welche Klasse ANTLR aus Lexer und Parser Grammatik generiert. Dies können weiterhin von anderen Klassen des Systems aufgerufen werden.

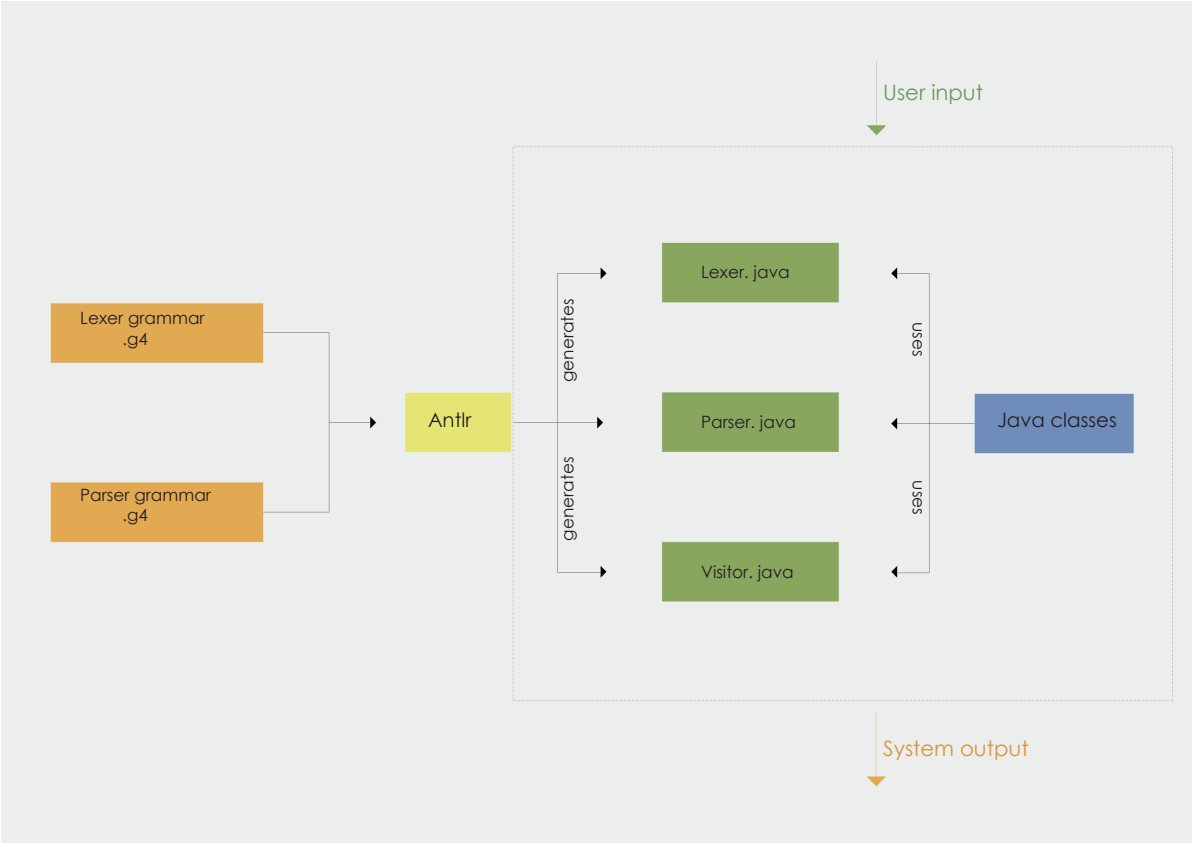


Abbildung 2.6.: ANTLR Workflow

3. Entwicklung des Auswertungssystems

In diesem Kapitel werden die Ausgangssituation sowie die Idee dieser Arbeit erläutert. Weiterhin wird die im Rahmen dieser Arbeit entworfene Syntax für Ausdrücke in der relationalen Algebra vorgestellt. Am Ende dieses Kapitels wird die Implementierung des Auswertungssystems der Operationen der relationalen Algebra beschrieben.

3.1. Ausgangssituation und Idee

Die Übungsaufgaben im Rahmen der Lehrveranstaltungen Datenbanken lassen sich grob in zwei Typen unterteilen. Der erste Typ sind Pflichtaufgaben, deren Lösungen online von Studierenden hochgeladen werden und vom Dozenten bzw. von der Dozentin korrigiert werden. Da die Pflichtaufgaben eine Rolle für die Zulassung zur Klausur spielen, müssen die Dozierenden nach der Korrektur eine kurze Rückmeldung zu jeder Lösung formulieren und die erreichten Punkte abhängig von der Korrektheit vergeben.
Der zweite Typ sind Übungsaufgaben, die von Studenten auf Papier gelöst werden und deren Lösungen in Präsenzübungen mit den Dozierenden bzw. Tutoren besprochen werden. Die Präsenzübungen bieten den Studierenden den unmittelbaren Kontakt zu den Dozierenden bzw. Tutoren, wodurch sie auch die Möglichkeit haben, Fragen zu stellen und unmittelbares Feedback zu bekommen. Die Präsenzübungen sind zeitlich begrenzt, was dazu führen kann, dass nicht alle möglichen Lösungen einer Aufgabe besprochen werden können oder die Dozierenden aufgrund der Zeitbegrenzung nicht jedem Studierenden ein Feedback zu seiner Lösung geben können. Zudem bekommen die Dozierenden keinen Überblick, wie gut das Thema der relationalen Algebra von jedem einzelnen Studenten verstanden worden ist.

Die Idee des Auswertungssystems ist, den Studierenden bei den Übungsaufgaben der relationalen Algebra im Rahmen der Datenbank-Vorlesung zu helfen und sie mit Feedback und Punktevergabe zu unterstützen. Das System ermöglicht es, Lösungen beliebig oft wiederholt prüfen zu lassen. Dies soll ermöglichen, relationale Algebra effizienter zu lehren und den Studierenden beim Lernen zu helfen. Durch lernunterstützende Feedbacks und Punktevergabe zu jeder eingereichten Lösung bietet das System den Studierenden

die Möglichkeit, verschiedene Lösungsansätze auszuprobieren und so selbst auf bessere Lösungen der Aufgaben zu kommen. Darüber hinaus könnte das System den Lehrenden durch automatische Korrektur und Punktevergabe jeder eingereichten Lösung helfen, Zeitaufwand gegenüber der üblichen Korrektur der Lösungen einzusparen.

3.2. Konzept dieser Arbeit

Das zu entwickelnde System soll in der Lage sein, die studentischen Eingaben mit der entworfenen Syntax anstelle von griechischen Buchstaben einzulesen und zu verarbeiten. Daher wird zunächst eine Syntax entworfen, die leicht auf einer üblichen Tastatur eingegeben werden kann. Dazu wird ein Parser benötigt, der diese Syntax interpretieren kann. Dieser Parser wird mit dem Tool ANTLR anhand einer Parser Grammatik erstellt. Darüber hinaus soll das System eine Auswertung der relationalen Operationen ermöglichen. Hierfür war eine Basis für ein Auswertungssystem von Prof. Dr. Felix Heine vorhanden, die relationale Operationen auswertet und das Ergebnis als Relation zur Verfügung stellt. Im Rahmen dieser Arbeit wird dieses Auswertungssystem um neue Funktionalitäten erweitert sowie das System um die fehlenden relationalen Operationen ergänzt. Basierend auf diesem Auswertungssystem soll ein Vergleich zwischen der Ergebnisse der studentischen Lösung und der Musterlösung durchgeführt werden. Auf Basis dieses Vergleichs soll das System in der Lage sein, lehrunterstützende Feedbacks für die studentische Lösung zu generieren und jeder eingereichten Lösung als weitere lehrunterstützende Maßnahme mit Punkten versehen. Schließlich soll das System in der Lage sein, die generierten Feedbacks, die erreichten Punkte und das Endergebnis einer Lösung in Form eines Endberichts auszugeben.

3.3. Entwurf der Syntax der relationalen Ausdrücke

In diesem Kapitel wird die entworfene Syntax für die relationalen Ausdrücke dargestellt. Zudem wird erklärt, wie die Syntax mit Hilfe von dem Parsergenerator ANTLR entworfen wurde.

3.3.1. Problemstellung

Wie in den Beispielen im zweiten Kapitel gezeigt wurde, werden üblicherweise die griechischen Buchstaben verwendet, um einen relationalen Ausdruck darzustellen. Diese sind jedoch schwierig auf einer Standardtastatur mit deutscher oder englischer Tastenbelegung einzugeben. Daher wurde zunächst eine Syntax für relationale Ausdrücke entworfen, die leicht auf einer solchen Tastatur einzugeben ist.

Die folgende Tabelle stellt die ausgewählten Befehlszeichen der entworfenen Syntax dar, die jeweils eine Operationen der relationalen Algebra repräsentieren:

Name	**Operation**	**Befehlszeichen**
Projektion	Π	PR
Selektion	σ	SL
Join	$\bowtie$	JN
Kartesisches Produkt	$\times$	X
Vereinigung	$\cup$	UN
Differenz	-	DI
Durchschnitt	$\cap$	IN
Umbenennung	ρ	R

Tabelle 3.1.: Relationale Operationen und deren Befehlszeichen

Die Grammatik der entworfenen Syntax wurde mit dem Parsergenerator ANTLR anhand einer kombinierten EBNF Grammatik erstellt. Diese besteht aus zwei Typen von Regeln:

- **Lexer Regeln:** dienen zum Zerlegen der Eingabe durch den Lexer. Lexer können anhand seiner Regeln zusammengehörende Zeichenketten innerhalb einer Eingabe erkennen und als Token an den Parser zurückgeben.
- **Parser Regeln:** Anhand der Parser Regeln und den vom Lexer übertragenen Token kann der Parser die Eingaben analysieren und daraus eine hierarchische Repräsentation erstellen, die für die weitere Verarbeitung benötigt wird. Hierzu müssen die eingelesenen Informationen in die kleinstmöglichen Bestandteile zerlegt werden, die nach den Regeln der jeweiligen Grammatik möglich sind. Außerdem wird analysiert, wie diese Bestandteile logisch zueinander in Beziehung stehen.

Mit Hilfe von Parser und Lexer Regeln erstellt ANTLR für jede Eingabe einen AST Baum. Der Wurzelknoten des Baumes besteht aus einer Expression, die bei jedem Durchlauf des Baumes eine einzelne Operation der relationalen Algebra repräsentieren kann.

3.3.2. Entwurf der Syntax

Im Weiteren werden einige Beispiele gezeigt, wie die Operationen der relationalen Algebra unter Verwendung der in dieser Arbeit entworfenen Syntax anstelle der griechischen Buchstaben dargestellt werden können. Zudem werden einige AST-Bäume vorgestellt, die jeweils von ANTLR für jede Eingabe generiert werden.

- $\pi_{Name,Abteilung}$ (Mitarbeiter):
  PR [Name,Abteilung] (Mitarbeiter) (Abbildung 3.1)

- $\sigma_{ANr=01}$ (Mitarbeiter):
  SL [ANr = 01] (Mitarbeiter) (Abbildung 3.2)

- $Mitarbeiter \bowtie_{Abteilung=FakNr} Studenten$:
  (Mitarbeiter) JN [Mitarbeiter.Abteilung = Studenten.FakNr] (Studenten)

- Mitarbeiter ∪ Studenten:
  (Mitarbeiter) UN (Studenten)

- $\sigma_{Wohnort=Hannover \quad \wedge \quad Gehalt>=2000}$ (Mitarbeiter):
  SL [Wohnort = Hannover & Gehalt >= 2000] (Mitarbeiter)

- $Mitarbeiter \bowtie_{Abteilung=ANr} Studenten$:
  (Mitarbeiter) JN [Abteilung = ANr] (Studenten)
  oder:
  (Mitarbeiter) JN [Mitarbeiter.Abteilung = Studenten.ANr] (Studenten)

- $Mitarbeiter \quad ⟕_{Abteilung=ANr} \quad Studenten$:
  (Mitarbeiter) L JN [Abteilung = ANr] (Studenten)

- Mitarbeiter - Studenten:
  (Mitarbeiter) DI (Studenten)

- Mitarbeiter ∩ Studenten:
  (Mitarbeiter) IN (Studenten) (Abbildung 3.3)

- $\rho_{Angestellte}$ (Mitarbeiter):
  R [Angestellte] (Mitarbeiter)

- $\sigma_{Wohnort=Hannover \quad \vee \quad Gehalt<=2000}$ (Mitarbeiter):
  SL [Wohnort = Hannover | Gehalt <= 2000] (Mitarbeiter)

- $\sigma_{Mitarbeiter.Abteilung \, = \, Studenten.ANr}$ (Mitarbeiter × Studenten):
  SL [Mitarbeiter.Abteilung = Studenten.ANr] ((Mitarbeiter) × (Studenten))

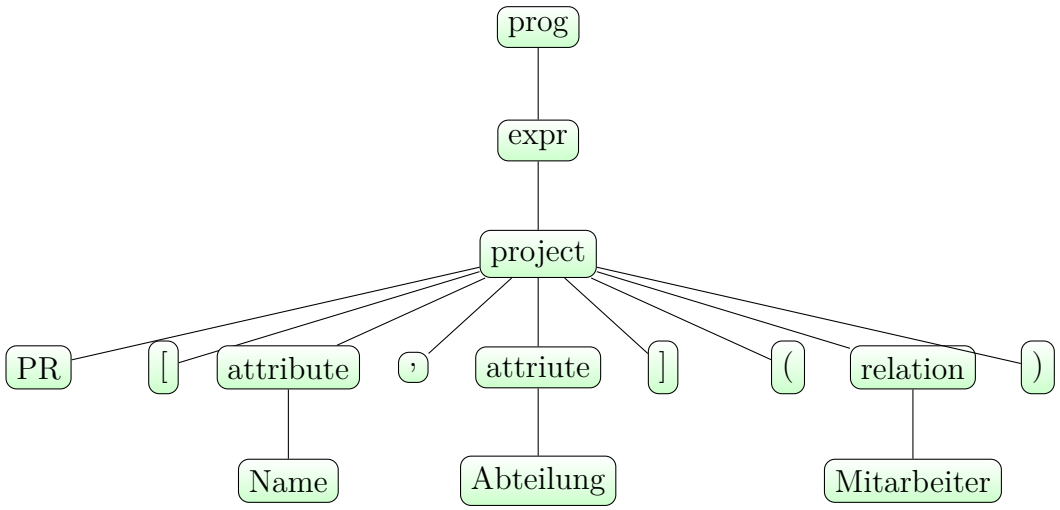


Abbildung 3.1.: AST-Baum für eine Projektion aus Mitarbeiter

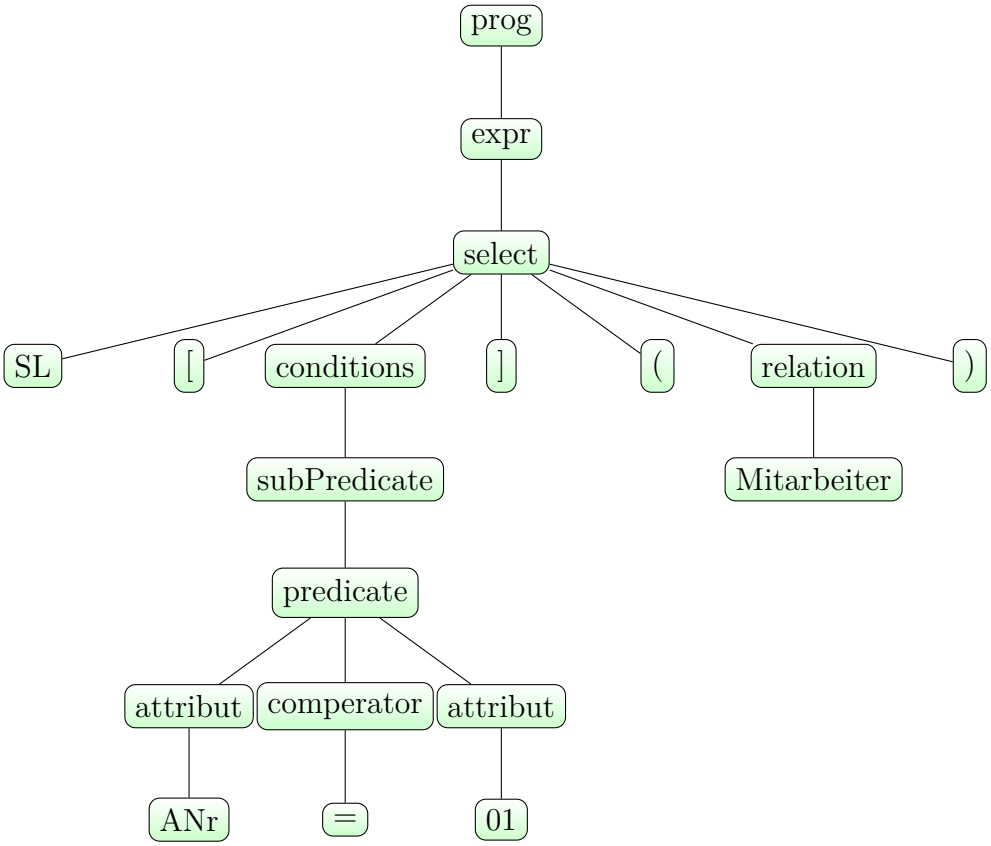


Abbildung 3.2.: AST-Baum für eine Selektion mit einem Prädikat ANr = 01 aus Mitarbeiter

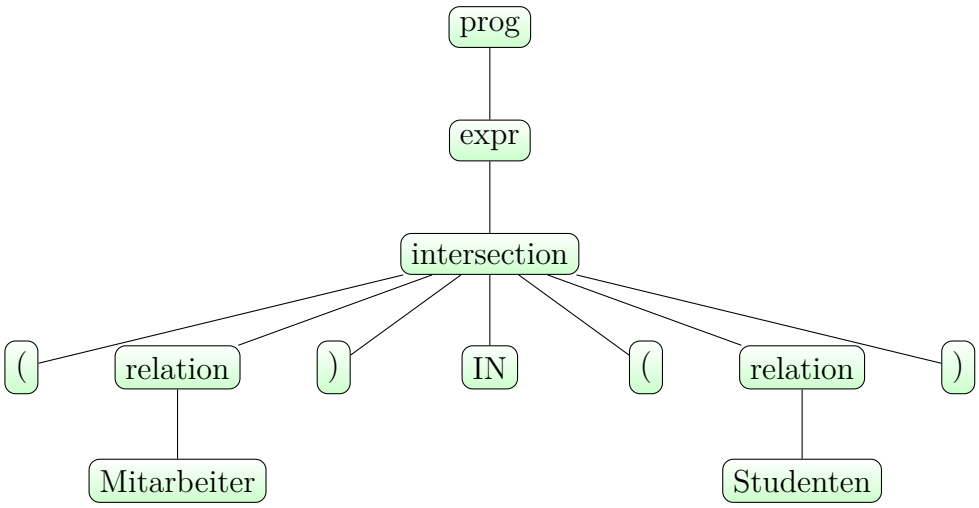


Abbildung 3.3.: AST-Baum für einen Durchschnitt

Komplexe Beispiele

Die folgenden Beispiele zeigen die Verschachtelung der relationalen Operationen unter Verwendung der entworfenen Syntax. Diese Beispiele beziehen sich auf die Relationen ANGEST, ANG_PRO, PROJKET (siehe Anhang A).

Die Lösung Nummer `1` zeigt die Lösung unter Verwendung der üblichen griechischen Buchstaben der relationalen Algebra. Während die Lösung Nummer `2` die gleiche Lösung unter Verwendung der entworfenen Syntax darstellt:

- Die Namen der Angestellten, die im Projekt 'Datawarehouse' arbeiten

1- $\pi_{name}((ANGEST \bowtie_{ANGEST.angnr=ANG\_PRO.angnr} ANG\_PRO) \bowtie_{PROJEKT.pnr=ANG\_PRO.pnr} (\sigma_{name="Datawarehouse"}(PROJEKT)))$

2- PR [name] ((ANGEST) JN [ANGEST.angnr = ANG_PRO.angnr] ((ANG_PRO) JN [ANG_PRO.pnr = PROJEKT.pnr] (SL [name= "Datawarehouse"] (PROJEKT))))

- Der Angestellte, der das Projekt 'Intranet' leitet und was er vom Beruf ist

1- $\pi_{ANGEST.name,beruf}(ANGEST \bowtie_{angnr=p\_leiter} (\sigma_{name="Intrane"}(PROEJKT)))$

2- PR [ANGEST.name, beruf] ((ANGEST) JN [angest.angnr = PROJEKT.P_leiter] (SL [name = "Intranet"] (PROJEKT)))

- Die Namen der Angestellten, die in Projekten arbeiten

1- $\pi_{ANGEST.name}(ANGEST \bowtie_{ANGEST.angnr=ANG\_PRO.angnr} (ANG\_PRO))$

2- PR [ANGEST.name] ((ANGEST)
JN [ANGEST.angnr = ANG_PRO.angnr] (ANG_PRO))

In Abbildung 3.4 ist die gesamte EBNF Grammatik dieser Arbeit abgebildet. Weiterhin werden die einzelnen Regeln dieser Grammatik diskutiert.

prog : expr
expr : select | project | cartesian | join | rename | union | intersection | difference
relation: ID #simple | expr #nested
rename : RENAME '[' ID ']' '(' relation ')'
select : SELECT '[' conditions ']' '(' relation ')'
project : PROJECT '[' attribut (',' attribut)+ ']' '(' relation ')'
cartesian: '(' relation ')' (CARTESIAN '(' relation ')')+
join : '(' relation ')' var JOIN '[' conditions ']' '(' relation ')'
union : '(' relation ')' | UNION | '(' relation ')'
intersection : '(' relation ')' | INTERSECTION | '(' relation ')'
difference : '(' relation ')' DIFFERENCE '(' relation ')'
predicate : attribut comparator attribut
conditions : subPredicate | subPredicate OR conditions
subPredicate : predicate | predicate AND subPredicate | '(' conditions ')' | ISNOT conditions
attribut : (ID '.')? ID
comparator : '=' | '<' | '>' | '>=' | '<=' | '!='
var : ('R' | 'L' | 'F')?
SELECT : 'SL'
PROJECT : 'PR'
JOIN : 'JN'
CARTESIAN : 'X'
RENAME : 'R'
UNION : 'UN'
INTERSECTION : 'IN'
DIFFERENCE : 'DI'
ID: [a-zA-Z0-9] [a-zA-Z0-9]*
AND : '&'
OR : '|'
NOT : '!'
WS : [\n \t \r]+ → *skip*

Abbildung 3.4.: Die entworfene Syntax dieser Arbeit in EBNF Form

Projektion: Die Eingabe für eine Projektion besteht aus 3 Komponenten. 'PR' ist der Bezeichner der Operation, gefolgt von einem einzigen Attribut oder einer Liste von Attributen. Diese werden in eckigen Klammern vom Komma getrennt eingegeben. Die dritte Komponente ist die Relation, auf die sich die Projektion bezieht. Dabei wird der Relationsname in runden Klammern eingegeben.

```
project : PROJECT '[' attribut ( ',' attribut )+ ']' '(' relation ')'

PROJECT : 'PR'
```

Selektion: Die Selektion besteht ebenfalls aus drei Komponenten. 'SL' ist der Befehlzeichen der Operation. Zu einer Selektion gehört ein Selektionsprädikat, über das die Dateneinträge ausgewertet werden. Dieses wird von eckigen Klammern umfasst.

```
select : SELECT '[' conditions ']' '(' relation ')'

SELECT : 'SL'
```

Prädikat: besteht aus folgenden Teilen:

- Linker Teil: Muss ein Attributname aus der eingegebenen Relation sein.
- Vergleichsoperator: Der Operator muss einem der folgenden Operationen entsprechen [=, >, <, >=, <=, ! =].
- Rechter Teil: Dieser Teil kann entweder aus einem Wert oder wiederum aus einem Attribut bestehen.

```
conditions : subPredicate | subPredicate OR conditions

subPredicate : predicate | predicate AND subPredicate | '('conditions')'
                         | ISNOT conditions

predicate : attribut comparator attribut

comparator : '=' | '<' | '>' | '>=' | '<=' | '!='

AND : '&'

OR : '|'

NOT : '!'
```

Join: Für die Join Bedingung gelten erneut die Regeln der Selektionsbedingung. Zusätzlich werden die Relationsnamen vor den Attributnamen getrennt durch einen Punkt eingegeben. Vor dem Bezeichner der Join Operation 'JN' wird die Variante der Join Operation wie in der Abbildung 3.2 dargestellt, eingegeben, um zwischen den unterschiedlichen Join Varianten zu unterscheiden:

```
join : '(' relation ')' var JOIN '[' conditions ']' '(' relation ')'

predicate : attribut comparator attribut

conditions : subPredicate | subPredicate OR conditions

var : ( 'R' | 'L' | 'F' )?

JOIN : 'JN'
```

Zeichen	Join Variante
F	Full Outer Join
R	Right Outer Join
L	Left Outer Join

Tabelle 3.2.: Join Varianten

Ein Inner Join wird automatisch gebildet, wenn keine der oben genannten Varianten eingegeben werden.

Kartesisches Produkt: Das kartesische Produkt repräsentiert den Cross Join und besteht aus zwei Relationen und dem Bezeichner 'X', der zwischen den beiden Relationsnamen eingegeben werden muss.

```
cartesian : '(' relation ')' (CARTESIAN '(' relation ')')+

CARTESIAN : 'X'
```

Umbenennung (Rename): Für die Umbenennungsoperation wird als Präfix das Befehlszeichen 'R' eingegeben. Der neue Name wird in eckigen Klammern umfasst. Nach der Ausführung dieser Umbenennungsoperation auf eine Relation kann es dann auf die Attribute dieser Relation durch die Eingabe des neuen Relationsnamen gefolgt vom Attributname zugegriffen werden:

```
neuer Relationsname.Attributname
```

```
rename : RENAME '[' ID ']' '(' relation ')'

RENAME : 'R'

ID: [ a-zA-Z0-9 ] [ a-zA-Z0-9 ]*
```

Mengen-orientierte Operationen (Vereinigung, Schnittmenge und Differenz): Bei Mengen-orientierten Operationen wird das Befehlszeichen nicht als Präfix, sondern zwischen zwei Relationen eingegeben. Hierbei müssen die Signaturen beider Relationen miteinander übereinstimmen. Außerdem müssen beide Relationen die gleiche Anzahl an Spalten und den gleichen Datentyp haben. Dies müssen ebenfalls in der gleichen Reihenfolge stehen. Anderweitig schlägt die Ausführung mit einer Fehlermeldung fehl.

```
union : '(' relation ')' | UNION | '(' relation ')'

intersection : '(' relation ')' | INTERSECTION | '(' relation ')'

difference : '(' relation ')' DIFFERENCE '(' relation ')'

UNION : 'UN'

INTERSECTION :'IN'

DIFFERENCE : 'DI'
```

3.3.3. Verschachtelung der relationalen Operationen

Da das System in der Lage sein soll, kombinierte relationale Ausdrücke in einer Eingabe zu verarbeiten. Es wurde zunächst ein Konzept entwickelt, wie die Operationen verschachtelt und verknüpft werden können. Das Ergebnis jeder einzelnen Operation der relationalen Algebra ist wiederum eine Relation, die die relevanten Daten enthält. Daher soll die Verknüpfung der Operationen über das Schlüsselwort Relation erfolgen. Zu diesem Zweck kann eine Relation entweder ein konkreter Relationsname (simple) oder wieder eine Expression (nested) sein, die wiederum eine neue Operation darstellt. Dabei dient das Ergebnis der inneren Expression als Relation der zuvor eingegebenen Expression. Die Abbildung 3.5 zeigt die zwei Eingabemöglichkeiten einer Relation:

```
relation : ID #simple | expr #nested

expr : select | project | cartesian | join | rename | union | intersection
     | difference

ID: [ a-zA-Z0-9 ] [ a-zA-Z0-9 ]*
```

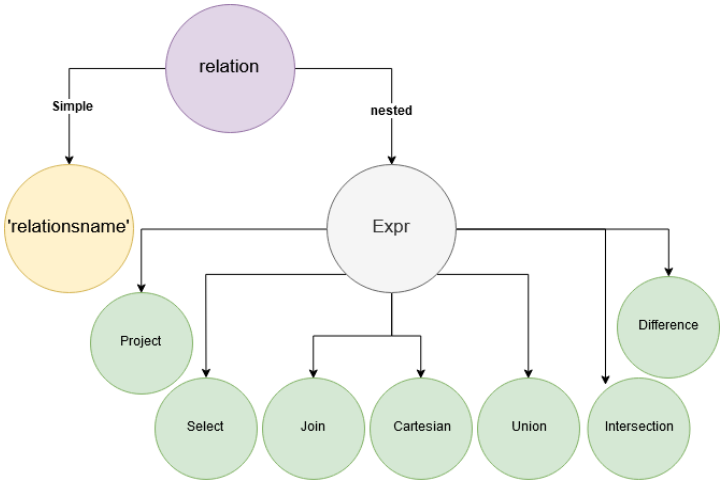


Abbildung 3.5.: Verknüpfung unterschiedlichen Operationen durch Relation

Bei syntaktisch inkorrekten Eingaben werden entsprechende Fehlermeldungen von ANTLR generiert und ausgegeben. Leerzeichen und Zeilenumbrüche werden in der Eingabe vernachlässigt.

3.4. Auswertungssystem der relationalen Operationen

In diesem Abschnitt wird die Implementierung des Auswertungssystems der Operationen der relationalen Algebra dargestellt. Es wird erklärt, wie die Eingabe unter Verwendung der entworfenen Syntax mit Hilfe des von ANTLR bereitgestellten Visitors und des generierten AST-Baums eine konkrete relationale Operation erstellt und zur Auswertung an das Auswertungssystem übertragen wird.

3.4.1. Implementierung des Auswertungssystems

Das System besteht grundlegend aus einer abstrakten Klasse (Abbildung 3.6), die die erforderlichen Methoden aller Operationen enthält.

- `getResult`: Die Methode liefert die Ergebnisrelation der Ausführung der Operation als Relation Objekt zurück. Dies wird von jeder Operation der relationalen Algebra überschrieben.
- `checkAttributesName`: Wird bei mengen-orientierten Operationen (Vereinigung, Durchschnitt und Differenz) verwendet, um die Schemata von zwei unterschiedlichen Relationen miteinander zu vergleichen. Diese Methode gibt eine Fehler-

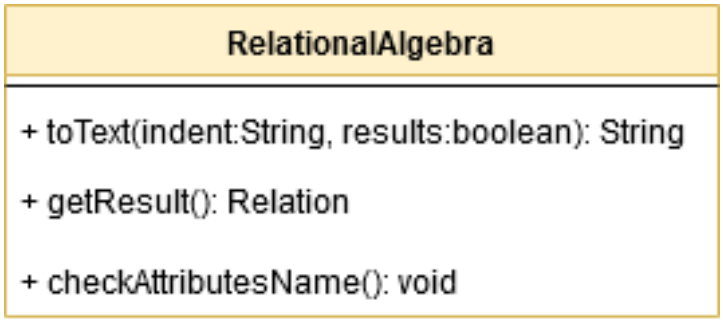


Abbildung 3.6.: Abstrakte Klasse aller relationalen Operationen

meldung aus, wenn die entsprechende Operation aufgrund der Unstimmigkeit der Schemata unausführbar ist.

- `toText`: Gibt den Inhalt der Ergebnisrelation als String zurück.

Für jede Operation der relationalen Algebra wurde eine konkrete Java-Klasse erstellt. Diese Klassen erben von der abstrakten Klasse RelationalAlgebra und überschreiben ihre Methoden. Dadurch ist es möglich, die verschiedenen Implementierungen der Operationen in separate Klassen auszulagern.

In der Abbildung 3.7 ist eine der grundlegenden Klassen des Auswertungssystems dargestellt. Die Klasse bildet eine Relation ab und besitzt die folgenden Komponenten:

- `name`: Name der Relation
- `attributes`: Diese Bilden zusammen das Schema einer Relation
- `Tupel`: Jedes Tupel repräsentiert einen Datensatz der Relation.

Im Folgenden werden einige Methoden der Klasse Relation erläutert:

- `addTuple`: Fügt der Relation ein neues Tupel hinzu
- `getResult`: Gibt die Relation als Relation-Objekt zurück
- `contentToText`: Gibt den gesamten Inhalt der Relation als String zurück
- `getAttributes`: Gibt eine Liste der Attribute zurück, die in der Relation vorhanden sind.
- `addAttribute`: Fügt der Relation ein neues Attribut hinzu
- `contains`: Prüft, ob ein gegebenes Tupel in der Relation enthalten ist
- `containsAttribute`: Prüft, ob ein bestimmtes Attribut in der Relation enthalten ist.

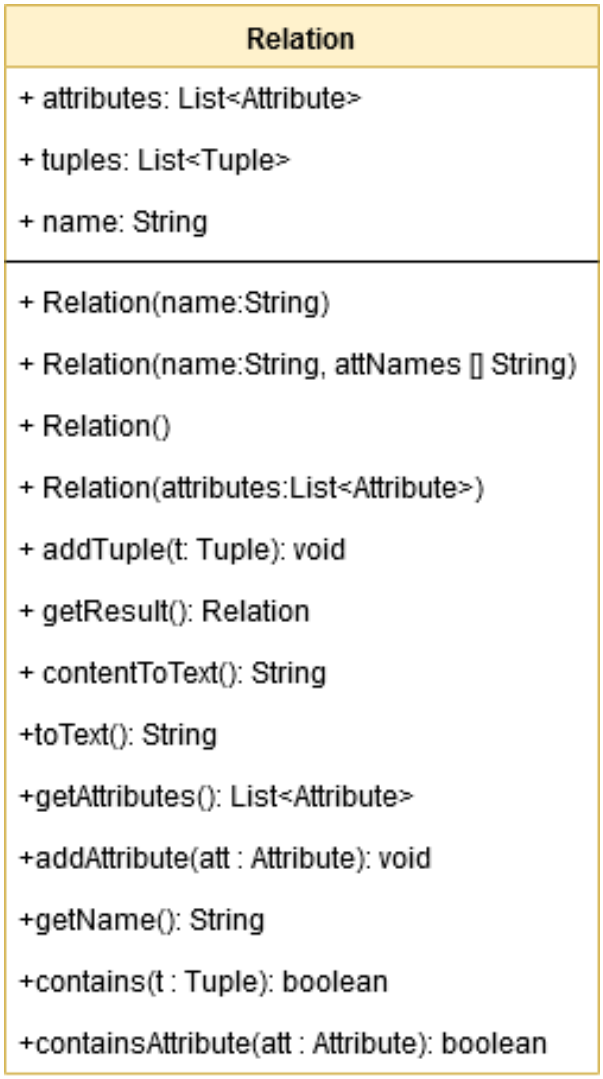


Abbildung 3.7.: Klasse: Relation

Im Folgenden werden die Klassen der relationalen Operationen einzeln erläutert:

Projection: Diese Klasse bekommt eine Relation, auf welche die Projektion ausgeführt wird, sowie die Attribute, die aus dieser Relation gefiltert werden sollen. Diese werden im eigenen Konstruktor initialisiert.

Selection: Diese Klasse bekommt ebenfalls eine Relation, sowie eine Bedingung, die auf die Attribute der Relation ausgewertet werden muss.

Join: Besitzt zwei Relationen und eine BooleanExpression, die die Join-Bedingung repräsentiert. Hierbei werden zwei Boolean Werte (left, right) gebraucht, um zwischen den

Join-Varianten (Left-Join, Right-Join, Full outer-Join) unterscheiden zu können. Die Unterscheidung kann über diese Boolean-Werte erfolgen:

- (left = true , right = false) Left Outer Join
- (left = false , right = true) Right Outer Join
- (left = true , right = true) Full Outer Join
- (left = false , right = false) Inner Join

Rename (Umbenennung): bekommt eine Relation und einen Namen, mit dem die Relation umbenannt wird.

SetMinus (Differenz), Union (Vereinigung), Intersection (Durchschnitt): bekommen jeweils zwei Relationen. Außerdem wird in diesen Klassen die Methode checkAttributesNames implementiert. Die den Vergleich der Schemen bei der Relationen durchführt. Dies gibt bei Unstimmigkeit der Schemen eine Fehlermeldung aus.

Implementierung des Prädikats:

Der linke sowie der rechte Teil eines Prädikates können anhand der folgenden Klassen repräsentiert werden:

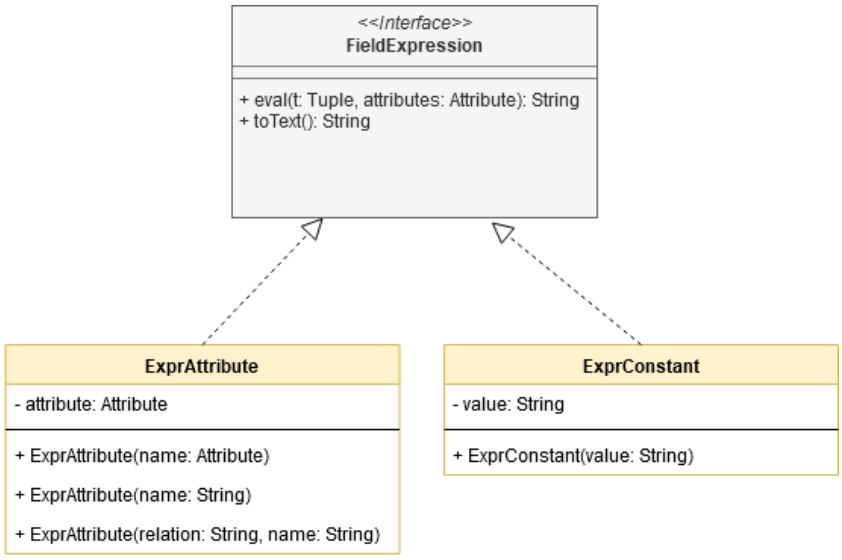


Abbildung 3.8.: FieldExpression Interface

- `ExprAttribut:` Stellt ein Attribut in einem Prädikat dar
- `ExprConstant:` Stellt einen Wert in einem Prädikat dar.

Beispiel: Sei Wohnort = "Hannover"eine Selektionsbedingung. Dies wird wie folgt repräsentiert:

```
ExprAttribut("Wohnort"), ExprConstant("Hannover")
```

Es wurde ebenfalls ein Interface mit dem Namen BooleanExpression erstellt. Dieses stellt die verschiedenen Vergleichsoperatoren (=, <, >, >=, <=, !=) sowie die logischen Operatoren (AND, OR, NOT) eines Prädikats dar. Das Interface in der Abbildung 3.9 verfügt über die folgenden Methoden:

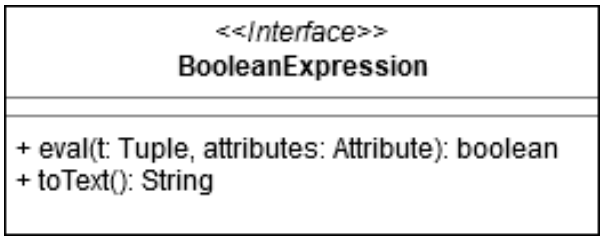


Abbildung 3.9.: BooleanExpression Interface

- `Eval` Methode erhält die Tupel und Attribute einer Relation als Parameter und kümmert sich um die Überprüfung der Bedingung für alle Tupel der Relation. Dies liefert bei den Tupel, die die Bedingung erfüllen, den Wert true zurück, ansonsten false.
- `toText` Methode gibt den Inhalt eines Prädikates zurück.

Die Bedingungen können mit den logischen Operatoren AND, OR und NOT kombiniert werden. Die logischen Operatoren AND und OR werden zum Filtern von Datensätzen verwendet, die auf mehr als einer Bedingung basieren:

- AND-Operatoren werden verwendet, um mehrere Bedingungen zu kombinieren. Die Ergebnismenge wird auf der Grundlage der Erfüllung beider Bedingungen gefiltert. Wenn also beide Bedingungen erfüllt sind, wird der Datensatz gefiltert. Es können mehrere AND Operatoren verwendet werden, um mehrere Bedingungen zu kombinieren.

  Bedingung A & Bedingung B

- OR-Operator wird ebenfalls verwendet, um mehrere Bedingungen zu kombinieren. Die Ergebnismenge wird auf der Grundlage der Erfüllung mindestens einer der Bedingungen gefiltert. Wurde eine der Bedingungen erfüllt, wird der Datensatz gefiltert. Um mehrere Bedingungen zu kombinieren, können mehrere OR-Operatoren als Teil des gesamten Prädikates verwendet werden.

Bedingung A | Bedingung B

- Der NOT-Operator wird verwendet, um die Ergebnismenge zu filtern, wenn die Bedingung nicht erfüllt ist.

! Bedingung A

Klassen der Operatoren: Es wurde für die Vergleichsoperatoren, sowie die logischen Operatoren jeweils eine Klasse erstellt, die die Schnittstelle BooleanExpression implementiert und deren Methoden anwendungsabhängig überschreibt. Das folgende Klassendiagramm stellt diese Klassen dar:

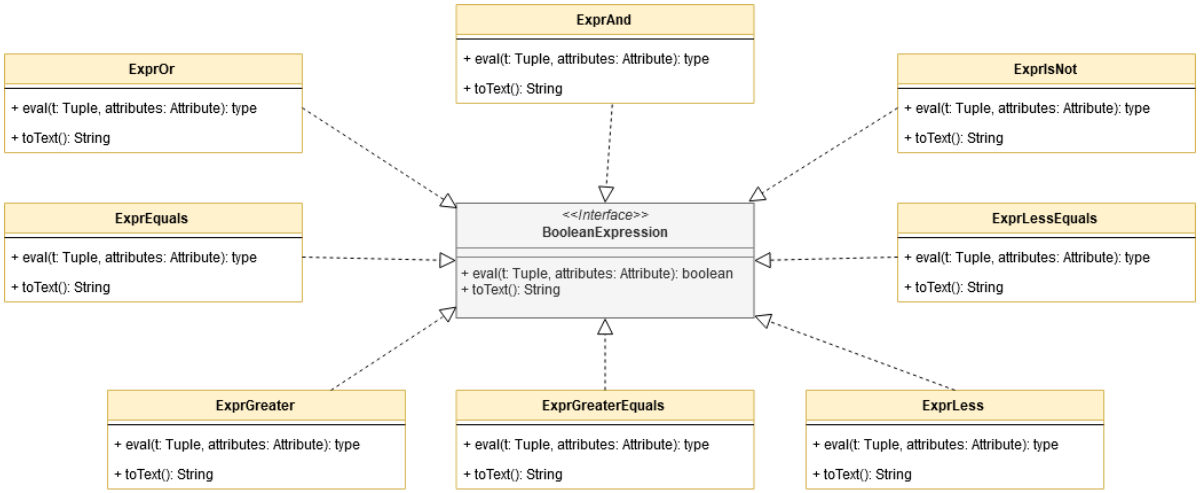


Abbildung 3.10.: BooleanExpression Klassen

Die Klassen lassen sich in zwei Typen unterteilen:

1- Klassen für logische Operatoren (And, Or, Not): Diese verknüpfen zwei BooleanExpression und bilden damit eine Kette von Bedingungen, die mit logischen Operatoren verbunden sind.

   – ExprOr

   – ExprAnd

   – ExprNot

2- Klassen für Vergleichsoperatoren: Diese verknüpfen zwei FieldExpressionen mit einem Vergleichsoperator und bilden damit eine einzige Bedingung.

Beispiele

- Sei `(Wohnort = "Hannover")` eine Bedingung einer Selektion, dann wird ein ExprEqulas Objekt erstellt, das den Vergleichsoperator (=) repräsentiert und zwischen einem Attribut und dem dazugehörigen Wert verknüpft.

```
ExprEquals(ExprAttribut("Wohnort"), ExprConstant("Hannover"))
```

- Sei `(Wohnort = "Hannover")` und (Gehalt = ”5000”) zwei Bedingungen einer Selektion, die mit dem AND Operator verknüpft sind

```
ExprAnd(ExprEquals(ExprAttribut("Wohnort"), ExprConstant("Hannover")),
        ExprGreater(ExprAttribute("Gehalt"), ExprConstant("5000")))
```

Wenn mehrere logische Operatoren in einem Ausdruck verwendet werden, werden OR-Operatoren nach AND-Operatoren ausgewertet. Hierbei kann jedoch die Reihenfolge der Auswertung geändert werden, in dem Klammern verwendet werden.

3.4.2. Implementierung des ANTLR Visitors

Um den Syntaxbaum einer Eingabe durchlaufen zu können, bietet ANTLR zwei verschiedene Baumlaufmechanismen:

- Listener
- Visitor

Zwischen beiden Mechanismen gibt es zwei Hauptunterschiede:

- Listener-Methoden werden automatisch von dem von ANTLR bereitgestellten Walker-Objekt aufgerufen, während Visitor-Methoden ihre Kinder mit expliziten visit-Aufrufen besuchen müssen.
- Listener-Methoden können keinen Wert zurückgeben, wohingegen Visitor-Methoden einen beliebigen benutzerdefinierten Typ zurückgeben können. Bei Listener müssen veränderbare Variablen verwendet werden, um Werte zu speichern, während dies bei Visitor nicht nötig ist.

Standardmäßig erzeugt ANTLR nur den Listener. Um einen Visitor generieren zu lassen, muss das Befehlszeilenoption '-visitor' verwendet werden. Mit der Befehlszeilenoption '-visitor' erstellt ANTLR aus der Grammatikdatei eine Standard Visitor-Klasse. Diese Visitor-Klasse enthält für jeden Regelnamen aus der Grammatik jeweils eine Visit-Methode. Beide Mechanismen können das gleiche Ergebnis darstellen aber in dieser Arbeit wurde sich für den Visitor-Mechanismus entschieden, da dies eine bessere Steuerung über die Traversierung der Knoten des Baums und direkte Nutzung der Parser-Ausgabe zur Interpretation bietet.

Funktionsweise des ANTLR-Visitors

ANTLR stellt eine Visitor-Schnittstelle und eine Klasse mit einer Standardimplementierung für die Visitor-Methoden bereit. Die Ausgaben der Grammatik in dieser Arbeit sind die Klasse RelAlgebraBaseVisitor und das Interface RelAlgebraVisitor. Es wurde zunächst ein neuer Visitor mit dem Namen ANTLRToExpression erstellt, der von der generierten Klasse RelAlgebraBaseVisitor erbt. Hierbei wurde der Rückgabewert des neues Visitor als Relation Objekt festgelegt. Da jede Visit-Methode des Visitors eine relationale Operation an jedem Knoten durchführt, muss das in jeder Visit-Methode entstehende Ergebnis als Relation-Objekt zurückgegeben werden. Die folgende Abbildung 3.11 zeigt einen Depth-First Walk des Parse-Baumes der Eingabe `'PR [Name] (Mitarbeiter)'` sowie die Reihenfolge der Aufrufe der Visit-Methoden des Visitors.

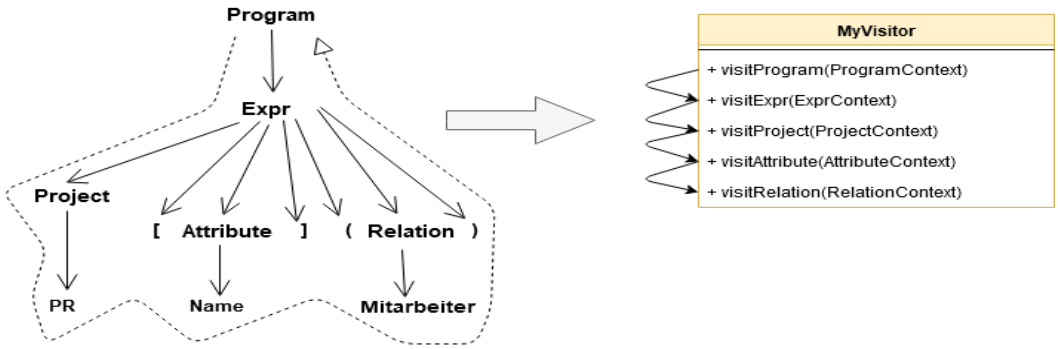


Abbildung 3.11.: Durchlauf eines ASTs

ANLTR erzeugt aus den gelesenen Token für jeden Eingang einen AST (Abstract Syntax Tree) und ruft die entsprechende Visit-Methode auf. Die Visit-Methoden lesen die Eingabe in jedem besuchten Knoten des Baums und speichern sie. Anschließend wird, je nach besuchter Methode, eine Instanz einer relationalen Operation erzeugt. Diese wird von der Auswertung der relationalen Operationen ausgeführt und die Ergebnisrelation wird als Objekt vom Typ Relation zurückgegeben.

In der Abbildung 3.12 ist zu sehen, welche Methoden von dem Visitor `ANTLRTOExpression` je nach gelesenen Token aufgerufen werden können.

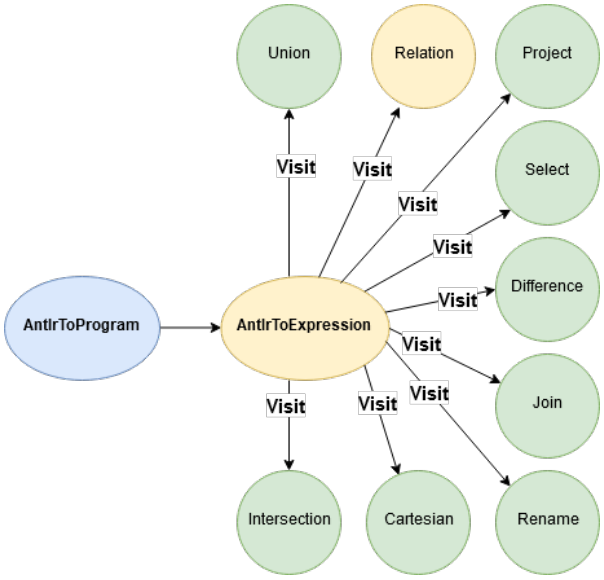


Abbildung 3.12.: ANTLRTOExpression Visitor

Prädikat Visitor

Es wurde in dieser Arbeit ein Visitor mit dem Namen `visitorPredicate` erstellt, der sowohl bei einer Selektion als auch bei einer Join Operation den Teilbaum des Prädikats durchläuft. Dieser Visitor ermöglicht das Einlesen und Zusammensetzen verschiedener Kombinationen von BooleanExpression, die in der Abbildung 3.10 dargestellt sind. Dabei werden die Eingaben an den Knoten des Baumes eingelesen und entsprechend BooleanExpression-Objekte erzeugt. Schließlich wird ein zusammengesetztes BooleanExpression-Objekt zurückgegeben, die aus verschiedenen Kombinationen von BooleanExpression-Objekten bestehen kann. Die Methoden dieses Visitors können sich je nach Eingabe rekursiv gegenseitig aufrufen. Das zurückgegebene Objekt einer Methode dient als Eingabe in der zuvor aufgerufenen Methode. In der Abbildung 3.13 ist die Klasse VisitorPredicate mit ihren Methoden dargestellt.

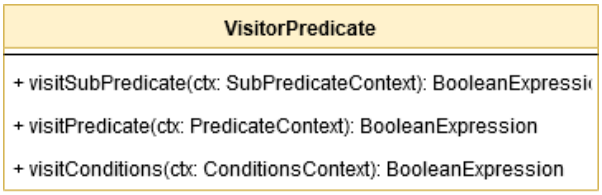


Abbildung 3.13.: Visitor für Prädikat in Selektion und Join Operationen

Beispiel:

Sei `'(Wohnort = Hannover AND Gehalt >= 5000)'` eine Bedingung einer Selektion Operation. Daraus wird der in der Abbildung 3.14 dargestellten Baum vom Parser erzeugt.

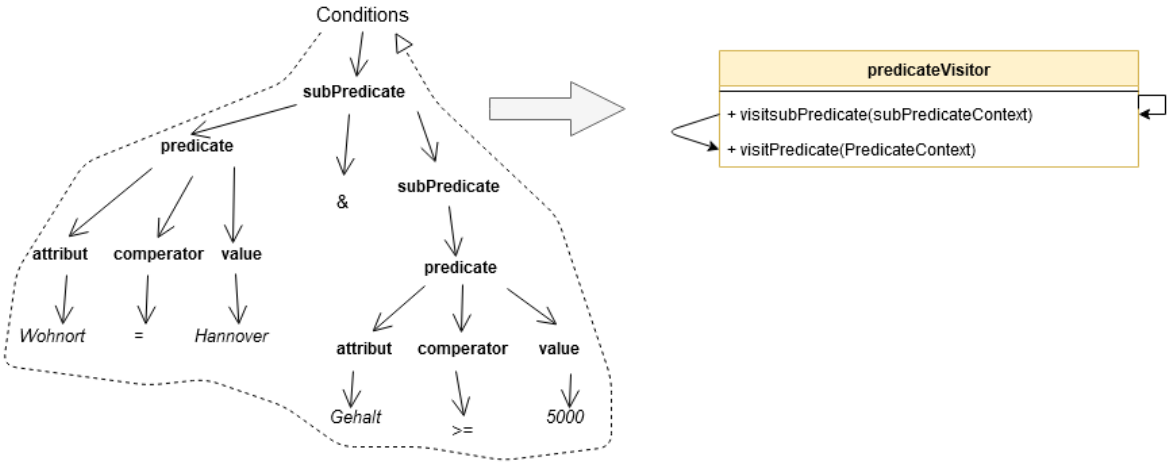


Abbildung 3.14.: Durchlauf einer Selektionsbedingung

Zunächst wird die Methode `visitSubPredicate` aufgerufen. Dies ruft im linken Teil des Baumes die Methode `visitPredicate` auf. Dies bildet das `ExprEquals` BooleanExpression-Objekt:

```
ExprEquals(ExprAttribut("Wohnort"), ExprConstant("Hannover"))
```

Im rechten Teil des Baumes wird zunächst die Methode `visitSubPredicate` wieder aufgerufen. Dies ruft die Methode visitPredicate, die ein `ExprGreaterEquals` BooleanExpression-Objekt bildet.

```
ExprGreaterEquals(ExprAttribut("Gehalt"), ExprConstant("5000"))
```

Die beiden gebildeten BooleanExpression-Objekte (`ExprEquals und ExprGreaterEquals`) werden an die zuerst aufgerufene Methode `visitPredicate` zurückgegeben, die aus den beiden Objekten ein weiteres `ExprAnd` BooleanExpression-Objekt bildet:

```
ExprAnd(ExprEquals(ExprAttribut("Wohnort"), ExprConstant("Hannover")),
        ExprGreater(ExprAttribute("Gehalt"), ExprConstant("5000")))
```

Das erzeugte BooleanExpression-Objekt steht dann bereit zur Auswertung und wird von dem Visitor als zusammengesetztes Objekt zurückgegeben.

ANTLRTOExpression Visitor

Im Folgenden werden die Methoden des entwickelten Visitors (`ANTLRTOExpression`) erklärt.

visitRelation: Hierbei wird zwischen zwei Fällen unterscheiden:

- **visitSimple:** Diese Methode wird automatisch von einer Visit-Methode einer Operation aufgerufen, wenn der besuchte Relationsknoten eine konkrete Relation ist und keine weitere Operation enthält. Diese Methode sorgt dafür, die konkreten Daten der Relation aus der Datenbank über eine DBFactory Klasse zurückzuliefern.
- **visitNested:** Diese Methode wird aufgerufen, wenn der besuchte Knoten wiederum eine weitere relationale Operation enthält. In dieser Methode wird dann die visitExpr Methode aufgerufen, um die weitere Operation des Baums auszuführen. Das Ergebnis der weiteren Ausführung wird durch visitNested als Ergebnisrelation zurückgeliefert.

Abhängig von dem vom Lexer zurückgegebenen Token kann ANTLR in der Regel automatisch erkennen, welche der Methoden visitSimple oder visitNested aufgerufen werden muss. Im Falle einer weiteren Operation in der Relation wird visitNested aufgerufen. Andernfalls wird visitSimple aufgerufen.

visitProjection: Liest die Attribute der Projektion ein. Die Relation wird über visitSimple oder visitNested wie oben erläutert zurückgeliefert. Anschließend wird eine Instanz der Projektion Operation erzeugt, die dann vom Auswertungssystem ausgeführt wird. Das Ergebnis wird als Relation-Objekt zurückgegeben.

visitJoin: In dieser Methode wird anhand der Variable `Var` vor dem Join-Befehlzeichen unterschieden, um welche Join-Variante es sich handelt und dementsprechend eine Join-Instanz erstellt. Das Ergebnis wird als neues Relation-Objekt zurückgegeben.

- L für `Left-Join`

- R für `Right-Join`
- F für `Full-Join`
- default für `Inner-Join`

visitRename: Liest den in eckigen Klammern eingegebenen Namen ein. Anschließend erstellt die Methode eine Rename Instanz mit den entsprechenden Parameter. Die zurückgegebene Relation trägt dann den neuen Namen.

visitSelection: Hierbei kommt der VisitorPredicte zum Einsatz, um alle Bedingungen sowie die Verknüpfungen dieser Bedingungen im Selektionsprädikat einzulesen. Nachher wird eine Selektion-Instanz mit einer Relation und den dazugehörigen Bedingungen (BooleanExpression-Objekt) erstellt.

visitDifference, visitUnion und visitIntersection: Diese Methoden haben einen ähnlichen Ablauf. Es werden zunächst die zwei Relationen über visitSimple bzw. visitNested besucht. Anschließend wird eine Instanz der Operation erstellt, die dann vom Auswertungssystem ausgeführt wird. Das Ergebnis wird als neues Relation-Objekt zurückgegeben.

4. Vergleich der Lösungen

In diesem Kapitel wird die studentische Lösung einer Aufgabe mit der vom Dozenten erstellten Musterlösung verglichen. Es wird erläutert, wie der Vergleich sowohl auf Basis der Struktur als auch auf Basis des Inhaltes der Ergebnisrelationen durchgeführt wird. Darüber hinaus wird in diesem Kapitel die Implementierung des Vergleichs behandelt.

Der grundsätzliche Ablauf ist: Der Dozent legt eine Aufgabe mit einer Musterlösung an. Die Erstellung der Musterlösung erfolgt nach der entworfenen Syntax. Zunächst werden beide Lösungen separat ausgewertet. Dies führt zu zwei Ergebnissen in Form von Relationen. Nun wird ein struktureller und inhaltlicher Vergleich zwischen den Relationen durchgeführt.

4.1. Vergleich

Der Test auf Korrektheit der Lösung wird über den Vergleich der Musterlösung und der studentischen Lösung erzielt. Im Rahmen dieses Vergleichs wird zunächst geprüft, ob die Struktur der Ergebnisrelation der Musterlösung gleich der Struktur der Ergebnisrelation der studentischen Lösung ist. Weiterhin wird inhaltlich verglichen, ob die beiden Ergebnisrelationen die gleiche Tupel bzw. Datensätze enthalten.

4.1.1. Struktureller Vergleich

Der strukturelle Vergleich befasst sich mit dem Vergleich auf Basis der Struktur der Ergebnisrelationen beider Lösungen. Dabei wird zunächst die Spaltenanzahl beiden Ergebnisrelationen miteinander verglichen. Mit dem Strukturvergleich kann festgestellt werden, ob ein Vergleich des Inhalts der Ergebnisrelationen durchgeführt werden kann.

4.1.2. Inhaltlicher Vergleich

Der inhaltliche Vergleich befasst sich mit dem Vergleich auf Basis des Inhaltes der Ergebnisrelationen, um Hinweise auf Abweichung geben zu können. Zunächst werden die

Attribute der Ergebnisrelationen auf Namensgleichheit und Reihenfolge geprüft. Anschließend wird anhand des in der Abbildung 4.1 dargestellten Statements geprüft, ob die Ergebnisrelationen die gleichen Tupel enthalten. So kann beispielsweise eine zu hohe oder niedrige Zeilenanzahl in dem studentischen Statement ein Hinweis auf vergessenen oder überflüssigen Teil in einem eingegebenen Prädikat helfen. [Car17]

```
(<studentisches Statement> Union < Musterloesung Statement>)
                            Minus
(<studentisches Statement> Intersection <Musterloesung Statement>)
```

Abbildung 4.1.: Vergleich der Tupel

Es ist jedoch zu beachten, dass die Korrektheitsprüfung auf der Identität des Ergebnisses beruht und kein sematischer Vergleich der Lösung mit der Musterlösung stattfindet. Das bedeutet, dass auch unterschiedliche Lösungen, die die korrekte Ergebnisrelation darstellen, als korrekt bewertet werden.

4.2. Implementierung des Vergleichs

Es wurde zunächst die Klasse Comperator erstellt. Dies ist für den strukturellen, sowie den inhaltlichen Vergleich zuständig. Das folgende Klassendiagramm 4.2 zeigt die erstellte Klasse und deren Komponenten.

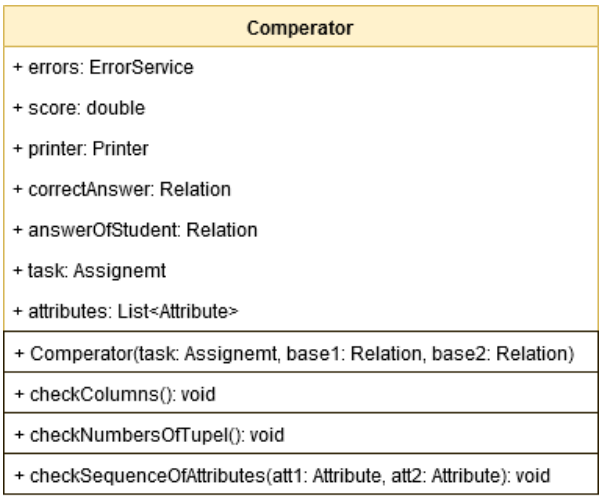


Abbildung 4.2.: Comperator Klasse

4.2.1. ErrorService

Service könnte verschiedene Fehlertypen, die im Weiteren dargestellt werden, in deren List enthalten. Es sollte sichergestellt werden, dass es nur eine einzige Instanz dieses Services erzeugt wird. Zu diesem Zweck wurde das Singleton pattern verwendet.

Singleton pattern: Gehört zur Kategorie der Erzeugungsmuster unter den Design pattern. Die Aufgabe des Singleton Pattern besteht darin, zu verhindern, dass von einer Klasse mehr als ein Objekt erstellt werden kann. Dies wird erreicht, indem das gewünschte Objekt in einer Klasse selbst als statische Instanz erzeugt wird. [Kle08] Die Klasse `Error Service` verfügt über zwei weitere Methoden: Die `addError` Methode fügt einen neuen Fehler zu der Fehlerliste hinzu. Die `getMistakes` Methode liefert die Fehlerliste eines ErrorServices zurück. Dadurch kann auf die Fehler des ErrorServices zugegriffen werden.

Abbildung 4.3.: ErrorService Klasse

Das System sollte in der Lage sein, verschiedene Arten von Fehlern zu behandeln und diese anzuzeigen. Auf diese Weise kann die Anzeige von Fehlern zu einer besseren Fehlersuche in der Lösung beitragen. Zu diesem Zweck wurde eine abstrakte Klasse mit dem Namen Mistake erstellt. Diese enthält die wichtigen Komponenten eines Fehlers. Die Abbildung 4.4 stellt die abstrakte Klasse Mistake dar.

Die Fehler, die bei der Korrektheitsprüfung auftreten können, wurden zunächst in verschiedenen Typen unterteilt. Für jeden dieser Fehler wurde eine eigene Klasse erstellt, die dann die Komponenten von der „Mistake“ Klasse erbt und deren Methoden überschreibt.

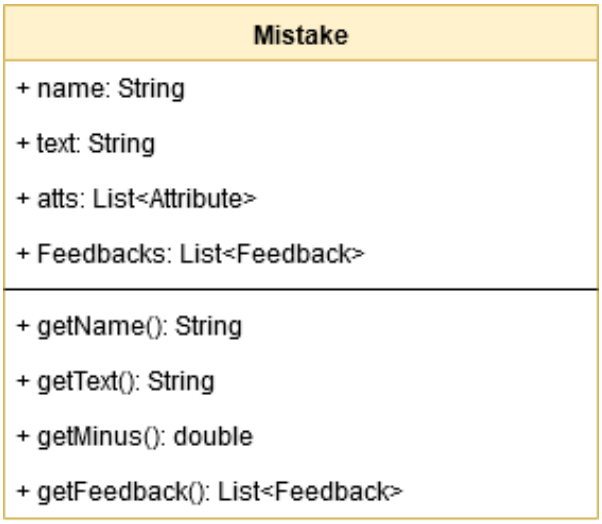


Abbildung 4.4.: Mistake Klasse

4.2.2. Fehlertypen

- Inequality (Ungleichheit): Dieser Fehler weist auf die Ungleichheit der Anzahl der Spalten zwischen der studentischen Lösung und der Musterlösung hin. (siehe Abbildung 4.5)

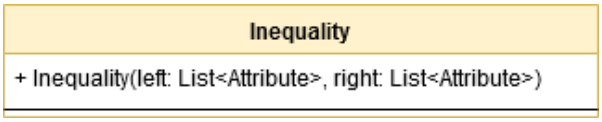


Abbildung 4.5.: Inequality Error Klasse

- Discrepancy (Unstimmigkeit): Weist auf die Unstimmigkeit der Attributnamen zwischen der studentischen Lösung und der Musterlösung hin. (siehe Abbildung 4.6)

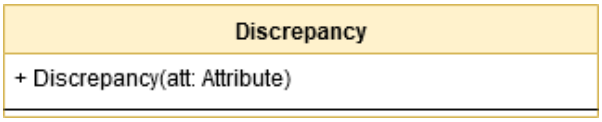


Abbildung 4.6.: Discrepancy Error Klasse

- Sequence (Reihenfolge): Dieser Fehler wird ausgelöst, wenn die Reihenfolge der Attribute inkonsistent ist (siehe Abbildung 4.7)

- Content Deviation (Abweichung des Inhaltes): Wird bei einer Abweichung der Tupel zwischen den Ergebnisrelationen ausgelöst. (siehe Abbildung 4.8)

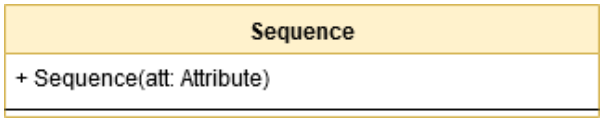


Abbildung 4.7.: Sequence Error Klasse

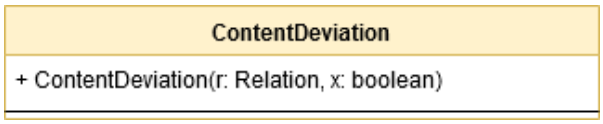


Abbildung 4.8.: Content Deviation Error Klasse

Komponenten eines Fehlers:

- Ein Name: bezeichnet einen Fehler
- Ein Text: gibt eine kurze Beschreibung über den Fehler an.
- Punktabzug: Punktabzüge werden anhand einer JSON-Datei konfiguriert.
- Eine Liste von Attributen: ergänzt die lernunterstützenden Feedbacks und weist darauf hin, wo welcher Fehler genau aufgetreten ist.
- Eine Liste von lernunterstützenden Feedbacks: Diese wird verwendet, um genauer auf Fehler hinzuweisen.

4.2.3. Ablauf

Ablauf des strukturellen Vergleichs:

Der Komparator führt zuvor den Strukturvergleich durch. Die Anzahl der Spalten der Ergebnisrelationen wird miteinander verglichen. Im Falle einer Ungleichheit wird ein Ungleichheitsfehler (Inequality) in die Liste des Error-Services aufgenommen und kein weiterer Vergleich durchgeführt. Wenn die Anzahl der Spalten in beiden Lösungen gleich ist, wird der inhaltliche Vergleich durchgeführt.

Ablauf des inhaltlichen Vergleichs:

Der Vergleich des Inhaltes wird erst durchgeführt, wenn der Strukturvergleich der Lösungen erfolgreich war. Bei diesem Vergleich werden zunächst die Namen der Attribute auf Existenz geprüft. Fehlt ein Attribut in der studentischen Lösung, wird ein Unstimmigkeitsfehler (Discrepancy) erzeugt. Sind alle erforderlichen Attribute in der studentischen

Lösung vorhanden, wird die Reihenfolge des Auftretens der Attribute geprüft. Bei einer Abweichung wird ein Fehler der Reihenfolge (Sequence) in das Error-Service aufgenommen. Wenn bei der Überprüfung der Reihenfolge und des Vorhandenseins der Attribute kein Fehler auftritt, wird das in der Abbildung 4.1 gezeigten Statement ausgeführt, um die Gleichheit der Tupel bzw. der Datensätze zu testen.

Bei der Durchführung des Statements können zwei Fälle auftreten:

- Leere Relation: Besagt, dass die Datensätze beider Lösungen identisch sind und somit die studentische Lösung als korrekt bewertet wird.
- Nicht leere Relation: Vermittelt, dass die studentische Lösung mehr oder weniger Datensätze enthält als die Musterlösung. Somit wird die studentische Lösung als inkorrekt bewertet. Hierbei wird ein Inhaltsabweichungsfehler (Content Deviation) erzeugt, der genaue Feedbacks auf die fehlenden bzw. überflüssigen Datensätze der studentischen Lösung hinweist.

5. Assessment

5.1. Punktevergabe

In der Regel wird für jede Aufgabe Punkte vergeben. Diese spielt bei der Endbewertung der Lösung eine Rolle. Die erreichte Note ist ein wichtiger Bestandteil des Lernens und hat einen großen Einfluss auf die Lernmotivation. Außerdem hilft sie dem Lernenden, einen Überblick darüber zu bekommen, wie gut der Lernstoff verstanden wurde. Eines der Ziele dieser Arbeit ist es, ein Punktesystem in Abhängigkeit von der Korrektheit der Lösung zu implementieren. Wie bereits im letzten Kapitel erwähnt, gibt es mehrere Arten von Fehlern, die beim Korrigieren einer Lösung auftreten können. Diese Fehler können je nach Aufgabe unterschiedlich gewichtet werden. Aus diesem Grund wurde beschlossen, eine JSON-Datei zu verwenden, die die Punktabzüge für jeden möglichen auftretenden Fehler enthält. Diese JSON-Datei kann von Dozierenden je nach Aufgabe bearbeitet oder erweitert werden.

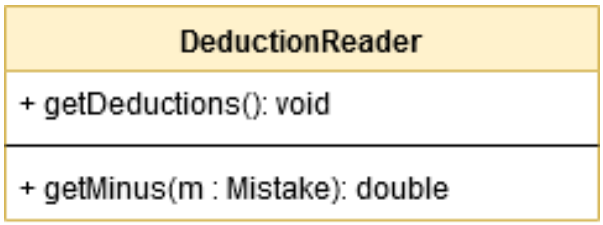


Abbildung 5.1.: Deduction Reader Klasse

Um die JSON-Datei einlesen zu können, wurde eine Klasse erstellt, die die Eingaben des JSON-Datei in einer HashMap in Form ('Key', 'Value') Paare speichert. Dabei ist Jede Fehler-Klasse dafür verantwortlich, ihren eigenen Punktabzug anhand der Methode getMinus() zu initialisieren. Diese sucht in der HashMap nach dem aufrufenden Fehler und gibt den zugehörigen Punktabzug zurück. Es sollte sichergestellt werden, dass keine Instanz der Klasse DeductionReader erzeugt wird. Daher ist diese Klasse abstrakt und hat nur statische Methoden.
Die Abbildung 5.2 zeigt die in dieser Arbeit verwendete JSON-Datei für Punktabzüge. Wie auf der Abbildung zu sehen ist, wird jeder Fehler durch einen Namen und den dazugehörigen Punktabzug repräsentiert:

```
{
    "Errors": [

            {
                    "Name": "Sequence",
                    "Minus": "0.3"
            },
            {
                    "Name": "Discrepancy",
                    "Minus": "0.5"
            },
            {
                    "Name": "Inequality",
                    "Minus": "0.2"
            },
            {
                    "Name": "ContentDeviation",
                    "Minus": "0.1"
            }
    ]
}
```

Abbildung 5.2.: JSON-Datei für Punktabzüge

Ablauf der Berechnung der erreichten Punktzahl:

Um die erreichte Note einer eingereichten Lösung berechnen zu können, wurde eine Klasse namens Calculator erstellt, die die Berechnung anhand der aufgetretenen Fehler sowie der gesamten Note einer Aufgabe berechnet. Die Abbildung 5.3 zeigt die Calculator Klasse

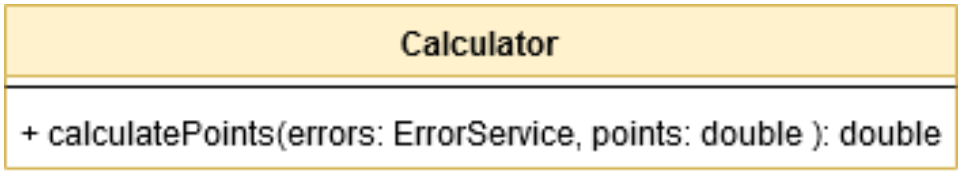


Abbildung 5.3.: Calculator Klasse

calculatePoints() ist die einzige Methode dieser Klasse. Diese bekommt eine Liste der aufgetretenen Fehler sowie eine Zahl, die der vollen Punktzahl einer Aufgabe entspricht.

Enthält die Liste keine Fehler, wird die volle Punktzahl der Aufgabe zurückgegeben. Andernfalls wird der Punktabzug für jeden aufgetretenen Fehler von der Gesamtpunktzahl der Aufgabe abgezogen und der Rest zurückgegeben. Es wurde sichergestellt, dass 0 zurückgegeben wird, wenn der Punktestand während des Abzugs Null erreicht.

Die Note wird nur berechnet, wenn der Strukturvergleich zwischen der studentischen Lösung und der Musterlösung nicht fehlschlägt. Wenn der Strukturvergleich fehlschlägt, wird die Note der eingereichten Lösung unmittelbar auf 0 gesetzt.

5.2. Generierung von lernunterstützenden Feedbacks

Feedback ist ein grundlegender Bestandteil des Lernens und hat einen enormen Einfluss auf das Lernen der Studierenden. Feedback lenkt die Aufmerksamkeit auf das, was wichtig ist. Es wirkt als Anreiz zum Lernen und es hat einen starken Einfluss darauf, was Studenten tun und wie sie es tun. Die Lernfortschrittskontrollen sind beim formativen Assessment in Form mehrerer kleiner Prüfungen in den Lernprozess integriert. Sie sollen dem Lehrenden und dem Lernenden einen fortwährenden Überblick über das Erreichen von Lernzielen geben.

Formatives Feedback dient dazu, den Aufbau und die Festigung des in der Vorlesung vermittelten Wissens zu unterstützen. Die regelmäßige, selbstständige und oft freiwillige Bearbeitung von Arbeitsaufträgen und die begleitenden Feedbacks können den Lernenden helfen, eigene Fehler zu erkennen und diese mit ihrem Lernverhalten in Verbindung zu bringen. Außerdem dienen sie der Lehrkraft dazu, Lehr- und Lernprozesse zu beobachten und unterrichtliche Maßnahmen zur besseren Kompetenzentwicklung einzuleiten. Beim formativen Assessment gibt es im Vergleich zum summativen Assessment mehrere Gelegenheiten der Evaluation unter realistischeren Bedingungen, wodurch ein kontinuierlicher Überblick über die Leistungen und Lernfortschritte der Lernenden gewonnen werden kann. [Sch18] [McC20]

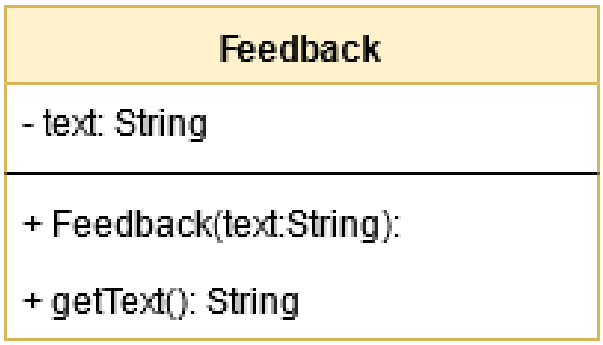


Abbildung 5.4.: Feedback Klasse

Die Generierung von Feedbacks wird von den aufgetretenen Fehlern übernommen. Bei der Erzeugung einen Fehler, wird im Konstruktor des Objekts Parameter übergeben. Dies helfen dabei, genaue und detaillierte Feedbacks zu generieren.

- Discrepancy Fehler (Unstimmigkeit): Ein Attribut wird übergeben, das den Fehler verursacht hat. Bei der Feedback-Generierung wird auf dieses Attribut hingewiesen.
- Sequence Fehler: Ebenfalls wird ein Attribut übergeben, das den Fehler verursacht hat. Damit können Feedbacks generiert werden, welche Attribute genau an falschen Stellen in der studentischen Lösung stehen.
- Inequality Fehler: Hierbei werden die Listen von Attributen der studentischen Lösung, sowie der Musterlösung übergeben. Damit wird geprüft, ob es sich um fehlende oder überflüssige Attribute in der studentischen Lösung handelt.
- Content Deviation Fehler: Bei der Erzeugung dieses Fehlers wird die Relation übergeben, die bei der Ausführung des Statements 4.1 für den Inhaltsvergleich entstanden ist. Mit einem booleschen Wert wird dann geprüft, ob in der studentischen Lösung fehlende oder überflüssige Datensätze vorhanden sind.

Die generierten Feedbacks werden in Listen gespeichert, die bei der Erzeugung des Endberichts aufgerufen werden, um die vorhandenen Feedbacks anzuzeigen.

5.3. Beispiele

5.3.1. Beispiel 1

Sei `'PR [Gehalt, Name] (ANGEST)'` die Musterlösung einer Aufgabe.

Versuch 1: Sei `'PR [Name, Gehalt] (ANGEST)'` eine studentische Lösung. Die Ausführung der Lösung wird eine Relation mit dem in der Abbildung 5.1 dargestellten Schema anzeigen:

Gehalt	**Name**

Tabelle 5.1.: ANGEST Relationschema

Nach der Durchführung des Vergleichs, werden folgende Feedbacks generiert, die auf die Abweichung der Reihenfolge der Attribute hinweisen.

Feedback: Das Attribut 'Name' steht an der falschen Stelle

Feedback: Das Attribut 'Gehalt' steht an der falschen Stelle

Versuch 2: `'PR [name] (ANGEST)'` hier wird der Strukturvergleich aufgrund Ungleichheit der Spaltennzahl einen Fehler auslösen. Das für diesen Fehler erzeugte Feedback lautet:

Feedback: Das Attribut 'Gehalt' fehlt.

Versuch 3: `'PR[Name, Gehalt] ((ANGEST) JN [angest.angnr = ang_pro.angnr] (ANG_PRO))'`

Hier wird nur der Inhaltlsvergleich fehlschlagen. Denn die Ergebnisrelation durch den Join in der studentischen Lösung enthält weniger Tupel als die Musterlösung. Dafür werden folgende Feedbacks generiert:

Feedback: Ein Prädikat könnte falsch sein oder die ausgeführte Operation passt nicht zur Aufgabe

Feedback: In der Ergebnisrelation fehlen folgende Zeilen.

ANGEST.NAME	ANGEST.GEHALT
...	...
...	...

Tabelle 5.2.: Fehlende Datensätze einer studentischen Lösung

Versuch 4: `PR [Gehalt, Name] (ANGEST)`

Mit dieser Lösung wird sowohl der Struktur- als auch der Inhaltsvergleich keinen Fehler auslösen. Daher wird die Lösung als korrekt bewertet und es werden keine Feedbacks erzeugt.

5.3.2. Beispiel 2

Sei `'SL[Wohnort = Hannover & Gehalt > 5000] (ANGEST)'` die Musterlösung einer Aufgabe.

Versuch 1: Sei `'SL[Wohnort = Hannover & Gehalt >= 5000] (ANGEST)'` eine studentische Lösung

Die Ergebnisrelation der studentischen Lösung hat mehr Datensätze als die Ergebnisrelation der Musterlösung. Hierbei löst der Inhaltsvergleich einen Fehler auf überflüssige Datensätze aus und es werden die folgenden Feedbacks ausgegeben:

Feedback: Ein Prädikat könnte falsch sein oder die ausgeführte Operation passt nicht zur Aufgabe

Feedback: Folgende Datensätze sind überflüssig:

ANGNR	NAME	WOHNORT	BERUF	GEHALT	ABTNR
199	Huber	Hannover	Administrator	5000	3

Tabelle 5.3.: Überflüssige Datensätze einer studentischen Lösung

Versuch 2: Sei `'SL[name = Schmidt] (ANGEST)'` eine studentische Lösung.

Die studentische Lösung im Versuch 2 bildet die gleiche Ergebnisrelation der Musterlösung. Daher wird diese Lösung als richtig bewertet und es wird keine weitere Feedbacks generiert.

Zum besseren Verständnis der Fehler und um eventuell eine schnelle Fehlersuche zu ermöglichen, werden zusätzlich die Ergebnisrelation der Musterlösung sowie der studentischen Lösung angezeigt.

5.4. Generierung des Endberichts

Um einen Bericht zu erstellen, wurde eine Klasse Namens „ReportGenerator“ erstellt. Sie ist für das Erstellen und Ausgeben des Endberichts einer Lösung zuständig. 5.5

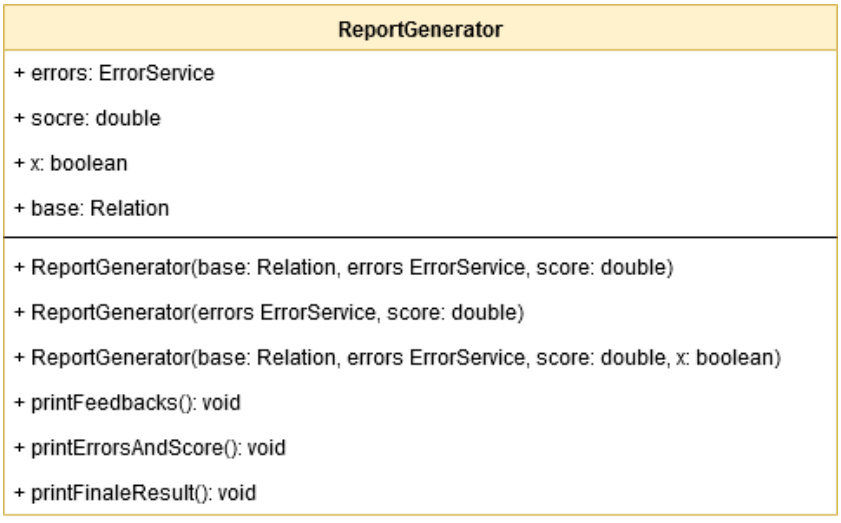


Abbildung 5.5.: ReportGenerator Klasse

Der Endbericht könnte folgende Komponenten enthalten:

- Fehlernamen, die bei dem Struktur sowie Inhaltsvergleich aufgetreten sind, und die dazugehörige Fehlerbeschreibung.
- Lernunterstützende Feedbacks zu den aufgetretenen Fehlern
- Die erreichte Note
- Das Endergebnis einer Lösung. Dies zeigt, ob die von Studierenden eingereichte Lösung als korrekt oder inkorrekt bewertet wurde.

Die folgenden Abbildungen 5.4, 5.5, 5.6 zeigen den Aufbau des Endberichts und seiner Bestandteile.

Errors	**Text**	**Minus**
1.Fehler	1.Fehlerbeschreibung	1.Punktabzug
2.Fehler	2.Fehlerbeschreibung	2.Punktabzug
...	....	...

Tabelle 5.4.: Muster eines Endberichts

Score	**Anzahl der erreichten Punkte**

Tabelle 5.5.: Muster für erreichte Note

Resultat	**korrekt oder inkorrekt**

Tabelle 5.6.: Muster des Endergebnis

6. Auswertung

In diesem Kapitel wird das Testen des Systems mit der Test-Driven-Development (TDD) Methode erklärt. Darüber hinaus werden einige Unit-Tests erklärt, die in dieser Arbeit implementiert wurden. Anschließend werden einige Testfälle beschrieben, die die Funktionsweise des Systems anhand einer studentischen und einer Musterlösung darstellen.

6.1. Test-Driven Development

"Test Driven Development (TDD) ist ein Softwareentwicklungs- und Designparadigma, das das Testen von Programmkomponenten nutzt, um den gesamten Softwareentwicklungsprozess zu steuern. Test Driven Development ist eine Entwurfsstrategie, die das Testen vor die Erstellung des Quellcodes stellt und im Hinblick auf die Prozesse priorisiert. Ziel ist es, die Qualität der Software deutlich zu erhöhen und den Wartungsaufwand im Nachhinein zu reduzieren. TDD wird meist im Kontext agiler Methoden und insbesondere bei Extreme Programmierung eingesetzt. Andere Begriffe für TDD sind testgetriebene Softwareentwicklung, testgeleitete Programmierung oder Test-First Design"[Rou17]

6.1.1. Funktionsweise von TDD

Test Driven Development läuft inkrementell ab. Nachdem die ersten Testfälle geschrieben wurden, wird die Software Schritt für Schritt erweitert. Bei jedem Schritt wird die Software mit teilweise minimalen Funktionen angereichert und erneut getestet. Jeder fehlerhafte Test führt zum Umschreiben von Quellcode, jeder bestandene Test erweitert den Funktionsumfang bzw. sichert die Funktionalität der Software. Einzelne Testfälle, auch Unit-Tests genannt, nehmen in der Regel wenig Zeit in Anspruch, so dass der Fortschritt in der Softwareentwicklung sofort sichtbar ist. Dieses Vorgehen führt zu einem sogenannten Red-Green-Refactor Zyklus, der die Abläufe strukturiert. Dieser Zyklus ist in der Abbildung 6.1 dargestellt.

- Test schreiben, der fehlschlägt und Rot markiert wird
- Produktivcode schreiben, der diesen Test besteht

- Refactoring: Der Produktivcode wird erweitert, ergänzt und neu umstrukturiert. Dazu werden Testfälle und Codebstandteile geschrieben. So läuft der Zyklus wieder ab, bis die Software aus Entwicklersicht einfach und verständlich ist.

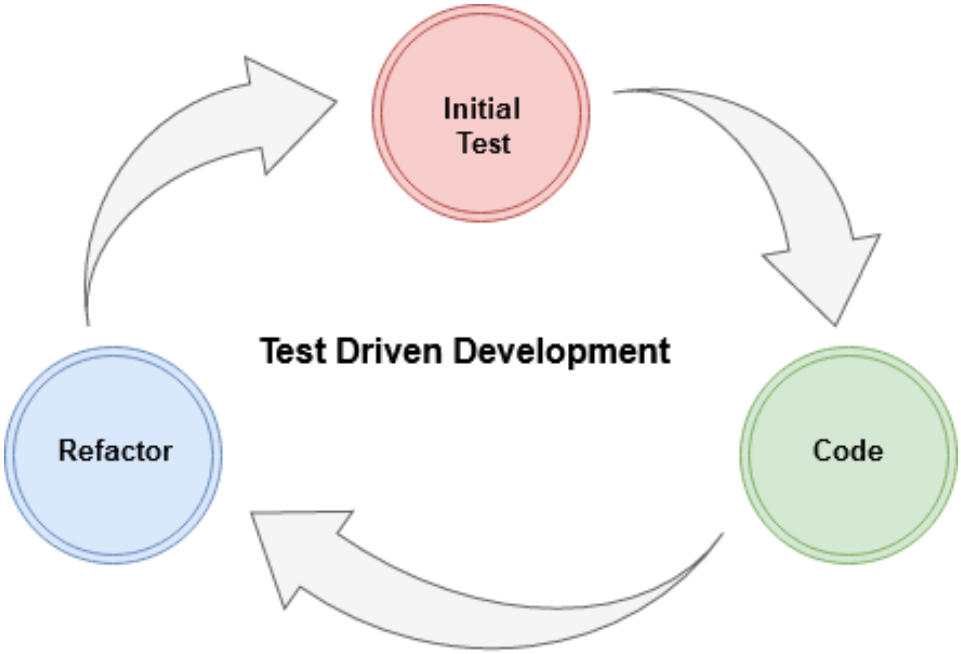


Abbildung 6.1.: Red Green Refactor Zyklus von TDD

6.1.2. Vorteile von TDD

- Das Ergebnis von TDD ist eine Software auf einem hohen Qualitätsniveau
- Die Fehleranalyse und eventuelle Wartungsarbeiten sind einfacher und schneller
- Sowohl der Code als auch die Systemarchitektur sind sauber strukturiert und klar verständlich
- Redundanzen und nicht benötigte Codebestandteile werden effektiv vermieden.

6.1.3. Unit-Tests

Unit-Tests sind eine Art des Software-Testings, bei denen die kleinsten Einheiten einer Anwendung, Units genannt, unabhängig und isoliert getestet werden. Das Ziel ist zu verifizieren, dass jede Einheit des Softwarecodes die erwartete Leistung erbringt. Unit-Tests isolieren einen Abschnitt des Codes und untersuchen dessen Korrektheit. Eine Unit kann eine einzelne Funktion, eine Methode, ein Modul oder ein Objekt sein [And03]

In dieser Arbeit wurden automatisierte Unit-Tests mit Hilfe des Junit-Frameworks geschrieben. Diese wurden verwendet, um die Funktionalität der Kernmodule zu untersuchen. Darunter befinden sich die Klassen des Auswertungssystems für die relationale Algebra. Der Punkte-Rechner wurde auf die Korrektheit der Berechnung der erreichten Note getestet. Darüber hinaus wurden einige Klassen des Systems isoliert auf ihre Funktionalität getestet, wie z. B. die Fehlerklassen, die Feedback-Klasse, die Relation-Klasse und die Reporter-Generator-Klasse. [Tam13] [Kle08]

6.2. Definierte Testfälle

In diesem Abschnitt werden einige Testfälle zum Testen des entwickelten Systems vorgestellt. Das System wird anhand einer studentischen Lösung und einer Musterlösung einer Aufgabe getestet. Ziel dieser Beispiele ist es, die Funktionsweise des Systems zu zeigen. Es ist zu beachten, dass die Punktabzüge der Fehler bei den folgenden Testfällen nur zum Testen konfiguriert wurden. Bei den Testfällen werden die Ergebnisrelationen der beiden Lösungen vorgestellt. Außerdem wird der Endbericht, der vom System erstellt wird, mit dem Endergebnis und den generierten Feedbacks gezeigt. Die Beispiele beziehen sich auf die Relationen, die im Anhang dieser Arbeit zu finden sind. A

6.2.1. Testfall 1

Wie heißen die Angestellte, die in Projekten arbeiten?

Die Musterlösung dieser Aufgabe ist:

```
PR[Name] ((ANGEST) JN [ANGEST.angnr = ANG_PRO.angnr](ANG_PRO))
```

Name
Müller
Winter
Matthäus
Schmidt

Tabelle 6.1.: Ergebnisrelation der Musterlösung beim 1. Testfall

Versuch 1: Sei die folgende Lösung eine studentische Lösung:

```
PR[Name, Gehalt]((ANGEST) JN [ANGEST.angnr = ANG_PRO.angnr](ANG_PRO))
```

Name	Gehalt
Müller	4500
Winter	5600
Matthäus	8000
Schmidt	6400

Tabelle 6.2.: Ergebnisrelation des 1. Versuchs beim 1. Testfall

Erwartete Ausgabe beim 1. Versuch:

Bei dieser Lösung wird der Strukturvergleich der Ergebnisrelationen fehlschlagen. Denn die studentische Lösung hat mehr Spalten als die Musterlösung. In diesem Fall wird die Punktzahl dieser Lösung direkt auf 0 gesetzt und es wird kein weiterer Vergleich durchgeführt. Die Abbildung 6.2 zeigt den Endbericht dieser Lösung.

Errors	Text	Minus
Ungleichheit	Die Spaltenzahl stimmt nicht überein	0
Score	0.0	
Result	not correct	

Feedback: *Das Attribut Gehalt ist überflüssig.*

Abbildung 6.2.: Ausgabe des 1. Versuchs beim 1. Testfall

Versuch 2: Beim zweiten Versuch wird das überflüssige Attribut 'Gehalt' gelöscht. Die neue Lösung lautet:

```
PR[Name]((ANGEST) JN [ANGEST.angnr = ANG_PRO.angnr](ANG_PRO))
```

Erwartete Ausgabe beim 2. Versuch:

Die dargestellte Lösung im zweiten Versuch stellt die korrekte Ergebnisrelation dar. Das heißt, die Lösung wird als richtig mit voller Punktzahl bewertet und es wird kein Feedback generiert. Die Abbildung 6.3 zeigt den Endbericht dieser Lösung.

Errors	Text	Minus
Score	5.0	
Result	correct	

Abbildung 6.3.: Ausgabe des 2. Versuchs beim 1. Testfall

6.2.2. Testfall 2

Wie heißt der Angestellte, der das Projekt 'Intranet' leitet und was ist er vom Beruf?

Die Musterlösung dieser Aufgabe lautet:

```
PR[ANGEST.Name, Beruf]( (ANGEST) JN [ANGEST.angnr = PROJEKT.P_leiter]
      (SL[name = Intranet] (PROJEKT)) )
```

Name	Beruf
Matthäus	Werbefachfrau

Tabelle 6.3.: Ergebnisrelation der Musterlösung beim 2. Testfall

Versuch 1: Sei die folgende Lösung eine studentische Lösung:

```
PR[ANGEST.Name, Beruf]((ANGEST) JN [ANGEST.angnr = PROJEKT.P_leiter](PROJEKT))
```

Name	Beruf
Matthäus	Werbefachfrau
Winter	Analytikerin
Schmidt	Ingenieur

Tabelle 6.4.: Ergebnisrelation des 1. Versuchs beim 2.Testfall

Erwartete Ausgabe beim 1. Versuch:

Bei diesem Versuch wurde ein Prädikat vergessen, das der Datensatz mit dem Projektname 'Intranet' selektiert. Daher werden irrelevante Datensätze in der Ergebnisrelation der studentischen Lösung entstehen. Die Abbildung 6.4 zeigt die Ergebnisrelation dieser Lösung sowie einen Hinweis auf die irrelevante Datensätze.

Errors	Text	Minus
Inhalt	Abweichung des Inhaltes	2.9
Score	2.1	
Result	Not correct	

Feedback: *Ein Prädikat könnte falsch sein oder die ausgeführte Operation passt nicht zur Aufgabe*

Feedback: *Folgende Datensätze sind überflüssig:*

ANGEST.NAME	*ANGEST.BERUF*
Winter	*Analytikerin*
Schmidt	*Ingenieur*

Abbildung 6.4.: Ausgabe des 1. Versuchs beim 2. Testfall

Versuch 2: Eine korrekte Alternativlösung zur Aufgabe ist:

```
PR[ANGEST.Name, Beruf]( SL[Projekt.name = Intranet]
     ((ANGEST)JN [ANGEST.angnr = PROJEKT.P_leiter](PROJEKT)))
```

Erwartete Ausgabe beim 2. Versuch:

Diese Lösung in Versuch 2 hat ein anderes Format als die Musterlösung, stellt aber dennoch die korrekte Ergebnisrelation dar. Die Alternativlösung wird deshalb als korrekt bewertet und es werden keine Feedbacks erzeugt.

6.2.3. Testfall 3

Wie heißen die Angestellte aus Hannover, die Projekte leiten und ab 6000 Euro verdienen, was ist Ihr Beruf und wie viel verdienen Sie?

Die Musterlösung dieser Aufgabe lautet:

```
PR[ANGEST.Name, Beruf, Gehalt] SL[Gehalt >= 6000 & Wohnort = Hannover]
   ((ANGEST) JN [ANGEST.angnr = PROJEKT.P_leiter](PROJEKT))
```

Die Ergebnisrelation der Musterlösung ist:

Name	Beruf	Gehalt
Schmidt	Ingenieur	6400
Schulze	Daten Analytiker	7000

Tabelle 6.5.: Ergebnisrelation der Musterlösung beim 3. Testfall

Versuch 1: Sei

```
PR[ANGEST.Name, Gehalt, Wohnort] ( SL[Gehalt >= 6000]
   ((ANGEST) JN [ANGEST.angnr = PROJEKT.P_leiter](PROJEKT)))
```

eine studentische Lösung.

Erwartete Ausgabe beim 1. Versuch:

Bei dieser Lösung werden zwei Fehler geworfen. Erstens ist die Unstimmigkeit der Schemata auf das überflüssige Attribut 'Wohnort' zurückzuführen. Zweitens befindet sich das Attribut 'Gehalt' in der studentischen Lösung an der falschen Stelle. In der Abbildung 6.5 wird der Endbericht dieser Lösung dargestellt.

Errors	Text	Minus
Reihenfolge	Abweichung der Reihenfolge	0.8
Unstimmigkeit	Schemata stimmen nicht überein	0.5
Score	3.7	
Result	Not correct	

Feedback: *Das Attribut ‚Gehalt' steht an der falschen Stelle*

Feedback: *Das Attribut Wohnort gehört nicht zur Lösung*

Abbildung 6.5.: Ausgabe des 1. Versuchs beim 3. Testfall

Versuch 2: Die im ersten Versuch geworfenen Fehler werden im zweiten Versuch korrigiert. Sei

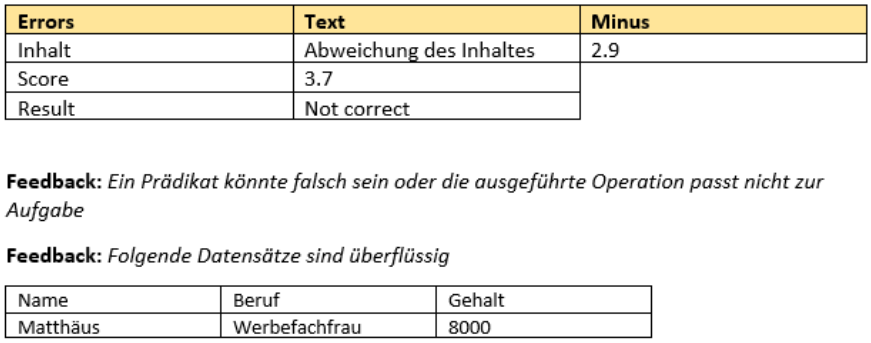
```
PR[ANGEST.Name, Beruf, Gehalt]( SL[Gehalt >= 6000]
   ((ANGEST) JN [ANGEST.angnr = PROJEKT.P_leiter](PROJEKT)))
```

die neue studentische Lösung.

Erwartete Ausgabe beim 2. Versuch:

In der neuen Lösung wurde das Attribut `'Wohnort'` hinzugefügt. Außerdem wurde das Attribut 'Gehalt' an die richtige Stelle gesetzt. Aber trotzdem ist die Lösung nicht ganz korrekt. Denn das Prädikat `(Wohnort = Hannover)` wurde vergessen. Daher wird

im Endbericht angezeigt, dass ein Prädikat nicht korrekt ist. Außerdem werden die überflüssigen Datensätze in der Ergebnisrelation ausgegeben. Die Abbildung 6.6 stellt den Endbericht der Lösung dar.



Errors	Text	Minus
Inhalt	Abweichung des Inhaltes	2.9
Score	3.7	
Result	Not correct	

Feedback: *Ein Prädikat könnte falsch sein oder die ausgeführte Operation passt nicht zur Aufgabe*

Feedback: *Folgende Datensätze sind überflüssig*

Name	Beruf	Gehalt
Matthäus	Werbefachfrau	8000

Abbildung 6.6.: Ausgabe des 2. Versuchs beim 3. Testfall

7. Fazit

7.1. Zusammenfassung

In dieser Bachelorarbeit wurde ein Lernunterstützungs- und Bewertungssystem für relationale Algebra für die Hochschule Hannover entwickelt. Zunächst wurde mit dem Werkzeug ANTLR4 eine EBNF-Syntax entworfen, um Ausdrücke der relationalen Algebra in das System eingeben zu können. Mit dieser Syntax war es möglich, relationale Ausdrücke anstelle von griechischen Buchstaben auf einer Standardtastatur einzugeben. Für die Auswertung der Operationen der relationalen Algebra wurde ein Auswertungssystem weiterentwickelt, das diese auf einer Datenbank auswertet und das Ergebnis in Form einer Relation zur Verfügung stellt. Auf Basis dieses Auswertungssystems war es möglich, einen Vergleich zwischen der studentischen Lösung und der Musterlösung zu realisieren. Anhand bestimmter Vergleichskriterien wurden verschiedene Fehlertypen definiert, die beim Vergleich der Lösungen auftreten können. So war das System in der Lage, verschiedene Fehlertypen zu erkennen und darauf basierend ein lernunterstützende Feedbacks zu generieren. Darüber hinaus wurde ein Scoring-System entwickelt, um die erreichte Punktzahl einer studentischen Lösung zu berechnen. Da das System als Konsolenanwendung entwickelt wurde, wurde ein einheitliches Design für den Abschlussbericht erstellt, um die erreichte Note, die generierten Feedbacks und das Endergebnis einer studentischen Lösung auf der Konsole (CLI) anzuzeigen. Dieser Abschlussbericht wird in Form einer Tabelle ausgegeben.

7.2. Ausblick

Um das System zu verfeinern, könnte in Zukunft eine grafische Benutzeroberfläche oder ein Web-Interface zur Benutzerfreundlichkeit entwickelt werden.

Ein System könnte entwickelt werden, das alle Versuche bzw. eingegebenen Lösungen der Studierenden zu einer Aufgabe speichert und anzeigt. Dieses System könnte sowohl den Studenten als auch den Dozenten helfen, einerseits herauszufinden, wo die Schwachstellen in dem in der Vorlesung vermittelten Wissen liegen. Andererseits dient es den Dozenten dazu, den Lehr- und Lernprozess zu beobachten und Maßnahmen zur besseren Kompetenzentwicklung einzuleiten.

Darüber hinaus könnte ein System zur Erkennung von partiellen Musterlösungen in der studentischen Lösung entwickelt werden. Dies könnte auf der Basis der relationalen Ausdrücke realisiert werden. Sodass Ausdrücke der studentischen Lösung und der Musterlösung verglichen werden, um partielle Musterlösungsausdrück zu erkennen. Oder auf der Basis der abstrakten Syntaxbäume. Sodass die vom Parser generierten abstrakten Syntaxbäume für die Musterlösung und die studentische Lösung verglichen werden, um Teilbäume der Musterlösung zu erkennen.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[AJA10] A.J.Admiraal. *Automated ANTLR Tree walker Generation*. Eingesehen am 02.12.2020. 2010. URL: https://fmt.ewi.utwente.nl/media/13.pdf.

[And03] David Thomas Andrew Hunt. *Pragmatic Unit Testing*. 2003.

[Car17] Felix Heine und Carsten Kleiner. *Automatisierte Bewertung in der Programmierausbildung*. Eingesehen am 15.12.2020. 2017. URL: https://www.waxmann.com/?eID=texte&pdf=3606Kapitel12.pdf&typ=zusatztext.

[DrM] Stefan Klauck Dr.Matthias Uflacker. *hpi: Datenbanken: Relationales Modell und SQL*. Eingesehen am 18.11.2020. URL: https://hpi.de/fileadmin/user_upload/fachgebiete/naumann/folien/SS11/DBS_I/DBS1_05_RelationaleAlgebra.pdf.

[Hei18] Prof. Dr. Felix Heine. *Datenbanksysteme 1 Skript*. 2018.

[JBo17] Robert Garmann Peter Fricke Paul Reiser Christopher Bersuch2 Oliver J.Bott. "Moodle Grappa aSQLg Graja neue Entwicklung bei der Grading-Software der Hochschule Hannover". In: (2017).

[Kle08] Stephan Kleuker. *Grundkurs Software-Engineering mit UML*. 2008.

[McC20] Teresa McConlogue. *Assessment and feedback in higher education*. 2020.

[Mic12] Günter Matthiessen Michael Unterstein. *Relationale Datenbanken und SQL in Theorie und Praxis 5.Auflage*. 2012.

[Par89] Terence Parr. *ANTLR*. 1989. URL: https://www.antlr.org/.

[Rou17] Margaret Rouse. *test-driven development (TDD)*. 2017.

[Sch18] Christin Schmidt. *formatives Assessment in der Grundschule*. 2018.

[Tam13] Michael Tamm. *JUnit Profiwissen*. 2013.

A. Anhang

ANGNR	NAME	WOHNORT	BERUF	GEHALT	ABTNR
112	Müller	Karlsruhe	Programmieren	4500	3
205	Winter	Hamburg	Analytikerin	5600	3
117	Matthäus	Osnabrück	Werbefachfrau	8000	5
198	Schmidt	Hannover	Ingenieur	6400	4
199	Huber	Hannover	Administrator	5000	3
200	Schulze	Hannover	Daten Analytiker	7000	6
201	Dammann	Hannover	Programmierer	4800	6

Tabelle A.1.: ANGEST Relation

PNR	NAME	P_BESCHR	P_LEITER
12	Datawarehouse	....	205
18	Intranet	....	117
17	Projekt DBMigration	....	198
33	VU	....	198
27	Klima Analyse	....	200
29	Green Software	....	201

Tabelle A.2.: PROJEKT Relation

PNR	ANGNR	PROZ_ARB
12	112	100
18	205	20
17	117	70
17	198	30
18	198	80
33	198	50
27	200	20
29	201	22

Tabelle A.3.: ANG_PRO Relation