eTutor

Modularchitektur und Moduldokumentation

Relational Database Design

Michael Karlinger

Version 1.0

Stand Februar 2006

INSTITUT FÜR WIRTSCHAFTSINFORMATIK

dke

Data & Knowledge Engineering

Inhaltsverzeichnis

1 Einleitung

Das Modul Relational Database Design (RDBD) realisiert die vom eTutor Core geforderten Funktionalitäten für verschiedene Beispieltypen aus dem Bereich des relationalen Datenbankentwurfs. Die Vorgehensweise im RDBD Modul nicht nur einen Beispieltyp, so wie in anderen Modulen des eTutor Systems, zu unterstützen sondern mehrere, ist durch die Nutzung von Synergieeffekten bei der Implementierung motiviert. Diese Synergieeffekte können aufgrund der Gemeinsamkeiten der Beispieltypen erzielt werden. Für das Verständnis der folgenden Erläuterungen ist es empfehlenswert, mit der Thematik des relationalen Datenbankentwurfs vertraut zu sein. Eine kurze und übersichtliche Darstellung dieser Thematik findet sich in den Ungsunterlagen zur Vorlesung Datenmodellierung.

Das RDBD Modul ermöglicht die Bearbeitung von Beispielen durch Studenten sowie die Administration von Beispielen der folgenden Beispieltypen:

- *RBR Algorithmus*: Der RBR Algorithmus dient dazu, Funktionalen Abhängigkeiten die in einer Teilrelation einer Basisrelation eingebettet sind, zu berechnen. Bei diesem Beispieltyp berechnet der Student ausgehend von einer Basisrelation und einer Teilrelation die, in dieser Teilrelation eingebetteten, Funktionalen Abhängigkeiten.

- *Allgemeine Normalisierung*: Eine Relation (Basisrelation) zu normalisieren bedeutet, diese Basisrelation so in Teilrelationen zu zerlegen, dass sich die Teilrelationen in einer gewünschten Normalform befinden. Bei diesem Beispieltyp berechnet der Student ausgehend von einer Basisrelation und der gewünschten Normalform genau diese normalisierten Teilrelationen.

- *Decompose Algorithmus*: Der Decompose Algorithmus stellt eine strukturierter Art und Weise dar, eine Basisrelation zu normalisieren. Der Unterschied zur Allgemeinen Normalisierung besteht genau darin, dass der Weg von der Basisrelation zu den normalisierten Teilrelationen vorgegeben ist. Bei diesem Beispieltyp berechnet der Student ausgehend von einer Basisrelation und der gewünschten Normalform unter Anwendung des Decompose Algorithmus die normalisierten Teilrelationen.

- *Minimale Überdeckung*: Bei diesem Beispieltyp berechnet der Student ausgehend von einer Menge an Funktionalen Abhängigkeiten die minimale Überdeckung dieser Funktionale Abhängigkeiten.

- *Attribut Hülle*: Bei diesem Beispieltyp berechnet der Student ausgehend von einer Menge an Attributen, im folgenden als Basisattribute bezeichnet, und einer Menge an Funktionalen Abhängigkeiten die Hülle der Basisattribute unter Berücksichtigung dieser Funktionalen Abhängigkeiten berechnen.

- *Schlüssel Bestimmen*: Bei diesem Beispieltyp bestimmt der Student die Schlüssel einer Relation unter Berücksichtigung der Funktionalen Abhängigkeiten dieser Relation.

- *Finden von Normalform Verletzungen*: Bei diesem Beispieltyp soll der Student für jede Funktionale Abhängigkeit einer Relation feststellen, ob die jeweilige Funktionale Abhängigkeit eine Verletzung einer bestimmten Normalform darstellt und falls dies zutrifft welche Normalform verletzt ist. Zusätzlich soll der Student feststellen, in welcher Normalform sich die Relation befindet.

Die vorliegende Dokumentation beschreibt die vom RDBD Modul zur Verfügung gestellten Funktionalitäten und adressiert sowohl Benutzer als auch Entwickler des RDBD Moduls. Als Benutzer des RDBD Moduls werden jene Personen erachtet, die primär an der zur Verfügung gestellten Funktionalität des Moduls interessiert sind. Die Anwendung dieser Funktionalitäten ist nicht Gegenstand dieser Dokumentation. Benutzer des RDBD Moduls die primär an der Verwendung des RDBD Moduls interessiert sind, werden auf das allgemeine Benutzerhandbuch des eTutor Systems verwiesen. Als Entwickler des RDBD Moduls werden jene Personen erachtet die über die ledigliche Auflistung der Funktionalitäten des RDBD Moduls hinaus auch an der Realisierung dieser Funktionalitäten interessiert sind. Die an Entwickler gerichteten Teile dieser Dokumentation sind realisierungsnahe Zusätze zu den an Benutzer gerichteten Teilen. Die gesamte vorliegende Dokumentation ist an Entwickler des RDBD Moduls gerichtet. Spezielle Teile daraus aber auch für Benutzer. Um eine Verdopplung der für beide Lesergruppen gleichermaßen konzipierten Teile der Dokumentation zu verhindern, ist diese Dokumentation nicht primär nach den Anforderungen der Lesergruppen aufgebaut sondern nach den beinhalteten Themen. Im Folgenden wird der Aufbau dieser Dokumentation beschrieben und darauf hingewiesen ob und inwiefern das jeweilige Thema auch für Benutzer und nicht nur für Entwickler konzipiert ist.

Kapitel 2 gibt einen Überblick über die Realisierung des RDBD Moduls. Dieses Kapitel ist speziell für Entwickler konzipiert. Darin werden das Design des RDBD Moduls, dessen Konfigurationsmöglichkeiten und Einbettung in das eTutor System beschrieben. Weiters wird die Strukturierung des Source Codes veranschaulicht und die verwendeten Bibliotheken erläutert.

Kapitel 3 beschreibt die für die Ausarbeitung eines Beispiels durch Studenten zur Verfügung gestellte Funktionalität des RDBD Moduls. In den darin enthaltenen Abschnitten wird die Funktionalität zur Beispiel Analyse, Beispiel Bewertung und dem Berichterstellen (Erzeugen von Feedback) des RDBD Moduls beschrieben. Jede dieser Beschreibungen ist nach Beispieltypen gegliedert. In jeder dieser Beschreibungen findet sich zu Beginn eine, an Benutzer des RDBD Moduls gerichtete, allgemeine Beschreibung und darauf aufbauend eine speziell an Entwickler gerichtete Beschreibung der Realisierung der Funktionalität.

Kapitel 4 beschreibt die für die Administration von Beispielen durch LVA Leiter zur Verfügung gestellte Funktionalität des RDBD Moduls. Eine Besonderheit des RDBD Moduls ist dabei, dass es erlaubt, den für Studenten bei der Beispielausarbeitung angezeigten Angabetext automatisiert zu erzeugen. Diese Funktionalität wird in Abschnitt 4.2 beschrieben und ist für Benutzer des RDBD Moduls konzipiert. In den restlichen Abschnitten dieses Kapitels werden Funktionalitäten zum Erzeugen, Löschen und Ändern eines Beispiels beschrieben. Diese Beschreibungen sind für Entwickler des RDBD Moduls gedacht.

Bei den an Entwickler gerichteten Teilen der Dokumentation wird davon ausgegangen, dass der Leser mit den Zusammenhängen zwischen dem eTutor Core und den einzelnen Modulen des eTutor Systems vertraut ist. Eine Dokumentation dieser Zusammenhänge findet sich in der Systemdokumentation des eTutors.

2 Realisierung im Überblick

Dieses Kapitel gibt einen Überblick über die Realisierung des RDBD Moduls und ist für Entwickler des RDBD Moduls konzipiert. Das Design des RDBD Moduls wird im Abschnitt 2.1 erläutert. Es wird ausdrücklich empfohlen dieses Kapitel zu lesen, bevor man sich den Details der Realisierung der Funktionalitäten für die Beispiel Ausarbeitung bzw. Beispiel Administration widmet. Im Abschnitt 2.2 werden die Konfigurationsmöglichkeiten des RDBD Moduls beschrieben. Abschnitt 2.3 widmet sich der Einbettung des RDBD Moduls in das eTutor System. Dort sind auch die am eTutor Core vorzunehmenden Konfigurationen für das Verfügbarmachen der Funktionalitäten des RDBD Moduls beschrieben. In den Abschnitten 2.4 und 2.5 werden abschließend die Strukturierung des Source Codes und die, zur Realisierung des RDBD Moduls benötigten, Bibliotheken beschrieben.

2.1 Design

Abbildung 1 zeigt die Komponenten des RDBD Moduls im Überblick. Die Zusammenhänge zwischen den Komponenten sind aus Gründen der Übersichtlichkeit nicht in Abbildung 1 vermerkt.

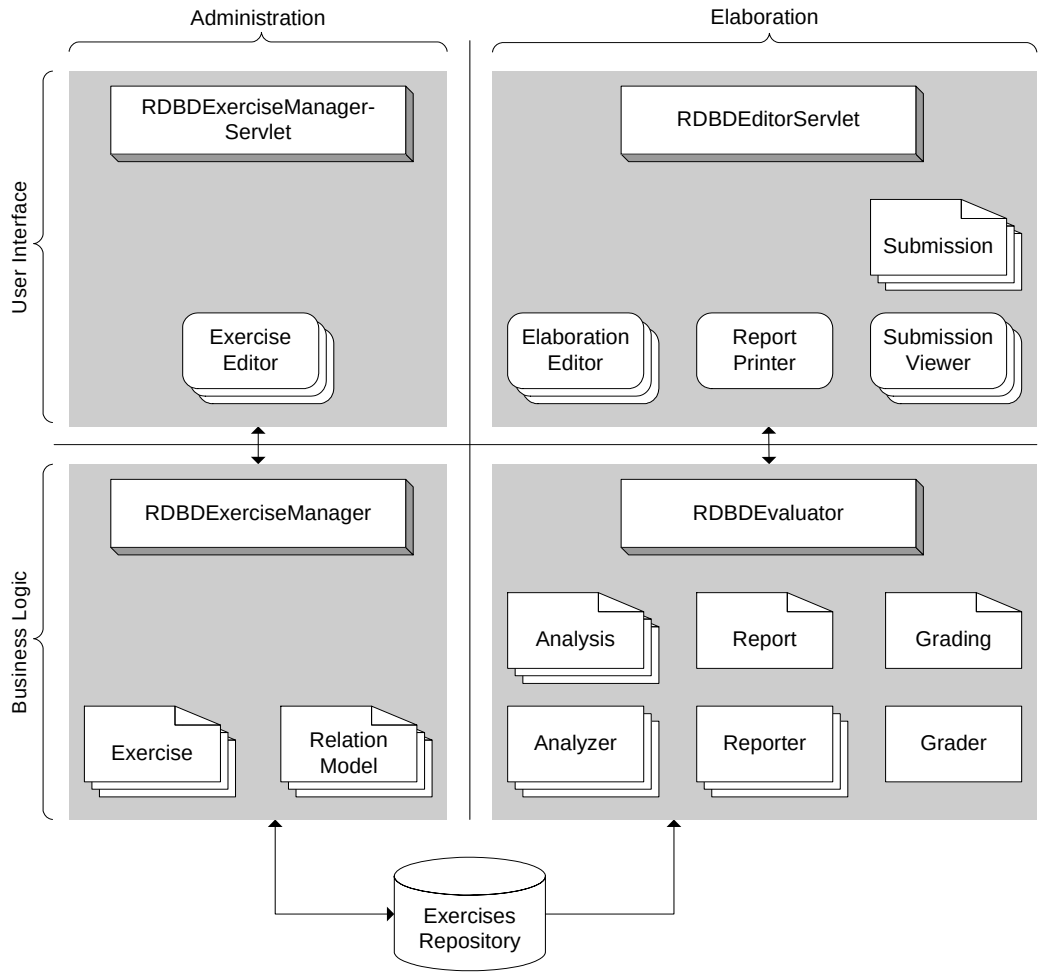

Abbildung 1: Komponenten des RDBD Moduls.

Diese Zusammenhänge sind der folgenden textuellen Beschreibung der Komponenten zu entnehmen. Die Komponenten des RDBD Moduls sind primär nach deren Verwendungszweck gegliedert. Dabei werden Komponenten die der Ausarbeitung von Beispielen dienen von denen unterschieden, die für die Administration von Beispielen vorgesehen sind. Sowohl die Komponenten der Beispiel Ausarbeitung als auch die Komponenten der Beispiel Administration sind in Komponenten der Benutzerschnittstelle und solche zu unterscheiden, die die zugrundeliegende Logik realisieren.

2.1.1 Komponenten der Beispiel Administration

Die Funktionalitäten der Beispiel Administration werden vom `RDBDExerciseManager` realisiert. Der `RDBDExerciseManager` implementiert das vom eTutor Core für die Bereitstellung der Funktionalitäten der Beispiel Administration vorgesehene Interface (`etutor.core.manager.ModuleExerciseManager`). Der `RDBDExerciseManager` stellt Funktionalitäten für das Anlegen, Löschen und Ändern von Beispielen (Exercise) zur Verfügung. Beispiele werden im Beispiel Repository (Exercises Repository) persistiert. Beispiele werden im Beispiel Repository als serialisierte JAVA Objekte (BLOB) gemeinsam mit einer Beispiel ID gespeichert.

Ein Beispiel kapselt die Information, die von den Komponenten der Beispiel Ausarbeitung benötigt wird, um ein Beispiel dem Studenten anzuzeigen bzw. zu evaluieren. Aufgrund der Charakteristika der im RDBD Modul unterstützten Beispieltypen ist ein wesentlicher Bestandteil dieser Information eine Relation bzw. Teile einer Relation. Für den Beispieltyp "Minimale Überdeckung" zum Beispiel ist die notwendige Information die Menge an Funktionalen Abhängigkeiten, für die eine Minimale Überdeckung berechnet werden soll. Für den Beispieltyp "Attribut Hülle" als ein weiters Beispiel sind die Basisattribute sowie eine Menge an Funktionalen Abhängigkeiten als Information notwendig.

Aufgrund dieser starken Gemeinsamkeiten zwischen den einzelnen Beispieltypen im RDBD Modul wurde ein Modell einer Relation implementiert. Sowohl die Komponenten der Beispiel Administration als auch die Komponenten der Beispiel Ausarbeitung arbeiten mit diesem Modell. Abbildung 2 zeigt den Aufbau dieses Modells. Eine Relation (Relation) kann einen Namen (name) haben und enthält eine Menge an Attributen (attributes). Attribute werden als Zeichenketten (String) repräsentiert. Eine Relation kann weiters beliebig viele Schlüssel (Key) haben. Ein Schlüssel besteht selbst wieder aus einer Menge an Attributen. Schließlich hat eine Relation auch eine Menge an Funktionalen Abhängigkeiten (FunctionalDependency). Eine Funktionale Abhängigkeit besteht aus Attributen der linken (lhsAttributes) und der rechten (rhsAttributes) Seite. Constraints, die aussagen, dass ein Attribut eines Schlüssels oder einer Funktionalen Abhängigkeit in den Attributen der jeweiligen Relation enthalten sein müssen wurden nicht umgesetzt, da solche Constraints bei der Verwendung der Modell Elemente oft hinderlich sind.

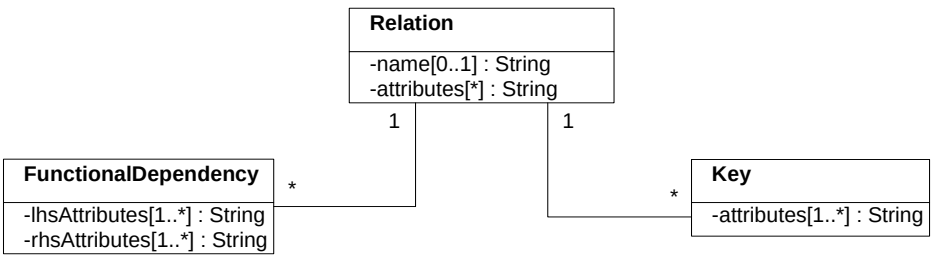

Abbildung 2: Modell einer Relation.

Die Beispiele selbst werden im RDBD Modul abhängig vom Beispieltyp entweder direkt in Form einer Relation dargestellt oder, sofern die, für die Evaluierung eines Beispiels benötigte Information über die in einer Relation darstellbare Information hinausgeht, durch eigene Klassen dargestellt. Genaueres zur Repräsentation von Beispielen finden Sie im Abschnitt 4.1.

Die zentrale Komponente der Benutzerschnittstelle für die Administration von Beispielen im RDBD Modul ist das `RDBDExerciseManagerServlet`. Das `RDBDExerciseManagerServlet` implementiert die Funktionalitäten der Benutzerschnittstelle für die Administration von Beispielen. Eine wesentliche Funktionalität ist das Anzeigen eines Editors für die Beispiel Administration (Exercise Editor). Da diese Beispiel Editoren abhängig vom Beispieltyp sind, wurde für jeden Beispieltyp ein eigener Beispiel Editor in Form einer JSP Seite implementiert.

2.1.2 Komponenten der Beispiel Ausarbeitung

Der zentrale Einstiegspunkt für die Beispiel Evaluation ist der `RDBDEvaluator`. Der `RDBDEvaluator` implementiert das vom eTutor Core für die Evaluierung von Beispielen vorgesehene Interface (`etutor.core.evaluation.Evaluator`). Das Analysieren, Bewerten und Berichterstellen wird vom eTutor Core zentral über den `RDBDEvaluator` veranlasst. Intern sind im RDBD Modul für jeden unterstützten Beispieltyp eigene Komponenten für das Analysieren und Berichterstellen implementiert. Die Bewertung ist von einer einzelnen Komponente für alle unterstützten Beispieltypen gemeinsam realisiert. Näheres zur Bewertung finden Sie im Abschnitt 3.2. Der `RDBDEvaluator` fungiert als Dispatcher der Kommandos des eTutor Cores auf Beispieltyp spezifische Komponenten.

Das Analysieren einer Beispiel Ausarbeitung (Submission) folgt im RDBD Modul dem algorithmischen Ansatz. Dieser Ansatz ist wie folgt charakterisiert: Für das Analysieren einer Beispiel Ausarbeitung sind zwei wesentliche Inputs erforderlich sind. Einerseits wird die, vom Studenten erstellte, Beispiel Ausarbeitung an sich und andererseits die Beispiel Angabe benötigt. Das Analysieren erfolgt durch Berechnen der korrekten Lösung auf Basis der Beispiel Angabe und Vergleichen dieser korrekten Lösung mit der Beispiel Ausarbeitung.

Sofern eine Beispiel Ausarbeitung durch Klassen des Relationen Modells darstellbar ist, werden diese Klassen direkt verwendet. Dies trifft zum Beispiel für die Ausarbeitung eines Beispiels vom Typ "Minimal Cover" zu. In diesem Fall besteht die Beispiel Ausarbeitung aus einer Menge an Funktionalen Abhängigkeiten, die die, vom Studenten berechnete, Minimale Überdeckung darstellen. Ist eine Beispiel Ausarbeitung nicht durch eine Klasse des Relationen Modells darstellbar, werden dafür eigens implementierte Klassen verwendet. Dies trifft zum Beispiel für die Ausarbeitung eines Beispiels vom Typ "Finden von Normalform Verletzungen" zu, da keine Klasse des Relationen Modells die Möglichkeit bietet, eine Zuordnung von Funktionalen Abhängigkeiten zu dadurch verletzten Normalformen zu repräsentieren.

Das Berechnen der korrekten Lösung ist abhängig vom Typ des jeweiligen Beispiels. Deshalb wurde im RDBD Modul für jeden Beispieltyp ein eigener Analysator (Analyzer) implementiert, der die Berechnung der korrekten Lösung übernimmt. Wird vom eTutor Core das Analysieren einer Beispiel Ausarbeitung über den `RDBDEvaluator` veranlasst, so lädt dieser die zugehörige Beispiel Angabe aus dem Beispiel Repository, identifiziert den passenden Analysator und übergibt diesem Analysator die benötigte Information, das heißt die Beispiel Angabe und Beispiel Abgabe in aufbereiteter Form. Die beim Analysieren festgestellten Abweichungen werden in sogenannten Analysen (Analysis) dargestellt. Da die feststellbaren Abweichungen ebenfalls abhängig vom Typ des jeweiligen Beispiels sind, wurde im RDBD Modul für jeden Beispieltyp eine eigene Klasse zur Repräsentation einer Analyse (Analysis) implementiert, die die jeweiligen Abweichungen

festhält. Detaillierte Informationen zu den einzelnen Analysatoren und Analysen im RDBD Modul finden Sie in Abschnitt 3.1.

Das Berichterstellen im RDBD Modul basiert auf der Analyse einer Beispiel Ausarbeitung eines Studenten. Da die Analysen abhängig vom Typ des jeweiligen Beispiels sind, sind auch die zu berichtenden Abweichungen abhängig vom Beispieltyp. Deshalb wurde im RDBD Modul für jeden Beispieltyp ein eigener Berichtersteller (Reporter) implementiert. Ein Berichtersteller erstellt auf Basis einer Analyse einen Bericht (Report). Obwohl der Inhalt eines Berichts abhängig vom Beispieltyp ist wird im RDBD Modul dennoch eine gemeinsame Struktur für Berichte verwendet. Genaueres zu den einzelnen Berichterstellern und dem Inhalt der einzelnen Berichte finden Sie in Abschnitt 3.3.

Den zentralen Einstiegspunkt für die Benutzerschnittstelle der Beispiel Ausarbeitung stellt das `RDBDEditorServlet` dar. Fordert der eTutor Core das `RDBDEditorServlet` auf, einen Editor (Elaboration Editor) bereitzustellen, der es dem Studenten ermöglicht ein bestimmtes Beispiel auszuarbeiten, so eruiert das `RDBDEditorServlet` den Typ des Beispiels und veranlasst den entsprechenden Editor, die Benutzerschnittstelle zu erzeugen. Diese Beispieltyp spezifischen Editoren sind in Form von JSP Seiten implementiert.

Die Funktionalität für das Anzeigen eines Berichts in der Benutzerschnittstelle der Beispiel Ausarbeitung ist gemeinsam für alle Berichte vom Report Printer realisiert, der ebenfalls in Form einer JSP Seite implementiert ist. Diese Vorgehensweise konnte im RDBD Modul eben dadurch ermöglicht werden, dass ein für alle Berichte gemeinsames Modell verwendet wird (vgl. Abschnitt 3.3).

Die Funktionalität für das Anzeigen einer Beispiel Ausarbeitung in der Benutzerschnittstelle der Beispiel Ausarbeitung ist vom Submission Viewer realisiert. Da Beispiel Ausarbeitungen abhängig vom Typ des jeweiligen Beispiels sind, wurde für jeden Beispieltyp ein eigener Submission Viewer in Form einer JSP Seite implementiert, der die Anzeige von Beispiel Ausarbeitungen übernimmt.

2.2 Konfigurationsmöglichkeiten

Die Konfigurationsmöglichkeiten des RDBD Moduls lassen sich in die Bereiche Datenbank-, Benutzerschnittstellen Parameter- und Komponenten Konfiguration aufteilen. Alle Konfigurationen werden zentral in der Klasse `etutor.modules.rdbd.RDBDConstants` verwaltet. Im Folgenden finden Sie eine Beschreibung der einzelnen Konfigurationsmöglichkeiten.

2.2.1 Datenbank Konfiguration

Das RDBD Modul benötigt eine Datenbank die es ermöglicht, serialisierte JAVA Objekte zu speichern, um Beispiel Angaben zu persistieren (Beispiel Repository). Um eine Verbindung zum Beispiel Repository herzustellen, verwendet das RDBD Modul einen Connection Pool, der über ein JNDI-Service verfügbar sein muss. Der JNDI-Kontext innerhalb des JNDI-Service, der diesen Connection Pool zugänglich macht, kann über das Property NAMING_DATASOURCE der Klasse `RDBDConstants` gesetzt werden.

In der momentanen Konfiguration wird der vom eTutor Core zur Verfügung gestellte JNDI-Service verwendet. Die am eTutor Core zur Verfügung gestellte Datenbank für die Verwaltung

der Beispiel Angaben des RDBD Moduls ist über den URL "jdbc:oracle:thin:@mond:1521:etutor" mit dem Benutzer "rdbd" und dem Passwort "rdbd" erreichbar.

2.2.2 Komponenten Konfiguration

Die konfigurierbaren Komponenten des RDBD Moduls sind die Editoren für die Beispiel Ausarbeitung (Elaboration Editors). In der Klasse `RDBDConstants` sind in entsprechenden Properties die Pfade zu den JSP Seiten, die die einzelnen Editoren implementieren, vermerkt.

2.2.3 Konfiguration der Parameter der Benutzerschnittstelle

Die Namen aller Parameter und Attribute die innerhalb von HTTP Requests, HTTP Responses und in der Session verwendet werden, sind in der Klasse `RDBDConstants` konfigurierbar. Durch diese Vorgehensweise wird es ermöglicht, dass im Falle von Namenskonflikten zwischen Attributen oder Parametern, diese einfach dadurch verhindert werden können, dass der Name der betroffenen Attribute bzw. Parameter in der Klasse `RDBDConstants` verändert wird. Eine weitere Adaption des Source Codes ist dann nicht mehr notwendig.

2.3 Einbettung in das eTutor System

Da das RDBD Modul in der derzeitigen Ausbaustufe die Kommunikation mit dem eTutor Core mittels entfernter Methodenaufrufe über RMI nicht unterstützt, muss das RDBD Modul im Web-Container des eTutor Cores installiert werden. Diese Installation wird vom zentralen deployment Skript des eTutor Cores (ANT Build File) durchgeführt. Für diese Installation ist es erforderlich, die vom RDBD Modul benötigten Servlets der Benutzerschnittstelle im Web-Container des eTutor Cores zu konfigurieren. Die dafür vorgesehene Konfigurationsdatei (web.xml) befindet sich im deployment Ordner des eTutor Cores und enthält die in Tabelle 1 angegebenen, RDBD Modul spezifischen Servlet Konfigurationen. Das URL Pattern dieser Servlet Konfigurationen ergibt sich aus "/modules/rdbd/" gefolgt vom Servlet Namen. Bei den implementierenden Klassen wurde das Package nicht explizit angegeben, da sich jede dieser Klassen im Package `etutor.modules.rdbd.ui` befindet.

Servlet Name	*Implementierende Klasse*
RDBDEditor	RDBDEditorServlet
RDBDExerciseManager	RDBDExerciseManagerServlet
AttributeClosureExerciseManager	AttributeClosureExerciseManagerServlet
DecomposeExerciseManager	DecomposeExerciseManagerServlet
KeysDeterminationExerciseManager	KeysDeterminationExerciseManagerServlet
MinimalCoverExerciseManager	MinimalCoverExerciseManagerServlet
NormalformDeterminationExerciseManager	NormalformDeterminationExerciseManagerServlet
NormalizationExerciseManager	NormalizationExerciseManagerServlet
RBRExerciseManager	RBRExerciseManagerServlet

Tabelle 1: RDBD Modul spezifische Servlet Konfigurationen am eTutor Core.

Um das RDBD Modul im eTutor Core verfügbar zu machen, sind zusätzliche Konfigurationen in der Tabelle Tasktype der eTutor Core Datenbank notwendig. Tabelle 2 zeigt die Konfigurationen in der Tabelle Tasktype, die für alle unterstützten Beispieltypen im RDBD Modul gleich sind.

Beschreibung	*Spalte*	*Wert*
Evaluator Klasse	classfile	etutor.modules.rdbd.RDBDEvaluator
Editor für Beispiel Ausarbeitung	submissionfile	/modules/rdbd/RDBDEditor
Benutzerschnittstellen Komponente für das Anzeigen eines Berichts	reporter	/modules/rdbd/printReport.jsp

Tabelle 2: Beispieltyp übergreifende Konfigurationen in der eTutor Core Datenbank.

Die Konfiguration der Editoren für die Beispiel Ausarbeitung sind dabei Referenzen auf das `RDBDEditorServlet`.

Tabelle 3 zeigt die Konfigurationen in der Tabelle Tasktype, die für jeden Beispieltyp im RDBD Modul individuell sind.

Beschreibung	*Spalte*	*Wert*
Exercise Manager Klasse	exercise_manager	etutor.modules.rdbd.exercises.<SpecificExerciseManager>
Editor für Beispiel Administration	create_exercise	/modules/rdbd/<SpecificExerciseManagerServlet>
Benutzerschnittstellen Komponente für das Anzeigen einer Beispiel Abgabe	submission_viewer	/modules/rdbd/<Beispieltyp>/showSubmission.jsp

Tabelle 3: Beispieltyp spezifische Konfigurationen in der eTutor Core Datenbank.

Die Konfigurationen der Exercise Manager Klassen sind in der Spalte exercise_manager anzugeben. Diese Exercise Manager Klassen befinden sich im Package `etutor.modules.rdbd.exercises`. Die einzelnen Klassen sind in Tabelle 3 nicht explizit angegeben. Die Namen dieser Klassen ergeben sich aus <Beispieltyp> gefolgt von "ExerciseManager".

Die Konfigurationen der Editoren für die Beispiel Administration sind in der Spalte create_exercise anzugeben. Die darin angegebenen Servlets sind in Tabelle 1 aufgelistet. Die Zuordnung zwischen Beispieltyp und Servlet sind offensichtlich.

Die Konfigurationen der Komponenten für das Anzeigen einer Beispiel Abgabe sind in der Spalte submission_viewer anzugeben.

2.4 Strukturierung des Source Codes

Der gesamte Source Code des RDBD Moduls ist im CVS des eTutor Projekts verwaltet. Die gewählte Ordnerstruktur entspricht der in der Systemdokumentation vorgegebenen Ordnerstruktur für Module des eTutor Systems. In diesem Abschnitt werden die Struktur und der Inhalt des Ordners für die JAVA Source Files und die Struktur und der Inhalt des Ordners für die Komponenten der Benutzerschnittstelle erläutert.

Tabelle 4 listet die Packages des RDBD Moduls auf und gibt eine kurze Erläuterung des Inhalts der einzelnen Packages.

Package	*Inhalt*
etutor.modules.rdbd	Klassen zur Repräsentation von Beispiel Angaben, Beispiel Ausarbeitungen und die Implementierung des `RDBDEvaluators`.
etutor.modules.rdbd.algorithms	Implementierungen von Algorithmen, wie zum Beispiel Member, Cover oder Closure, die beim Analysieren von Beispiel Ausarbeitungen häufig verwendet werden.
etutor.modules.rdbd.analysis	Klassen zur Repräsentation von Analysen und Implementierungen der einzelnen Analysatoren.
etutor.modules.rdbd.exercises	Die Exercise Manager Klassen.
etutor.modules.rdbd.model	Klassen zur Repräsentation des im RDBD Modul verwendeten Modells von Relationen.
etutor.modules.rdbd.report	Klassen zur Erstellung von Berichten.
etutor.modules.rdbd.ui	Die in der Benutzerschnittstelle des RDBD Moduls benötigten Servlets.

Tabelle 4: Packages im RDBD Modul.

Tabelle 5 zeigt die interne Struktur des Ordners für die Komponenten der Benutzerschnittstelle. Für jeden Beispieltyp wurde ein (gleichnamiger) Ordner erstellt, der die Benutzerschnittstellen Komponenten des jeweiligen Beispieltyps beinhaltet. Diese Komponenten sind für jeden Beispieltyp gleichermaßen:

- *showEditor.jsp*: Diese JSP Seiten realisieren die jeweiligen Editoren für die Beispiel Ausarbeitung.
- *showSubmission.jsp*: Diese JSP Seiten realisieren die jeweilige Funktionalität für das Anzeigen einer Beispiel Ausarbeitung.
- *specifyExercise.jsp*: Diese JSP Seiten realisieren die jeweiligen Editoren für die Beispiel Administration.

Ordner	*Inhalt*
attributeClosure	Komponenten der Benutzerschnittstelle für den Beispieltyp Attribut Hülle.
css	Cascading Stylesheets die in den einzelnen Komponenten der Benutzerschnittstelle verwendet werden.

decompose	Komponenten der Benutzerschnittstelle für den Beispieltyp Decompose Algorithmus.
images	Bilder, die in der Benutzerschnittstelle verwendet werden.
js	JavaScript Dateien, die am Client durchführbare Teile der Funktionalität der Benutzerschnittstelle realisieren.
keysDetermination	Komponenten der Benutzerschnittstelle für den Beispieltyp Schlüssel Bestimmen.
minimalCover	Komponenten der Benutzerschnittstelle für den Beispieltyp Minimale Überdeckung.
normalform	Komponenten der Benutzerschnittstelle für den Beispieltyp Finden von Normalform Verletzungen.
normalization	Komponenten der Benutzerschnittstelle für den Beispieltyp Allgemeine Normalisierung.
reductionByResolution	Komponenten der Benutzerschnittstelle für den Beispieltyp RBR Algorithmus.

Tabelle 5: Stukturierung der Komponenten der Benutzerschnittstelle.

2.5 Verwendete Bibliotheken

Für die Realisierung des RDBD Moduls wurden keine externen Gesamtsysteme verwendet. Bibliotheken werden daher nur für die Verbindung zum Beispiel Repository und für die JSP Seiten und Servlets der Benutzerschnittstelle benötigt.

Bibliothek	*Zweck*
ojdbc.jar	Für den Verbindungsaufbau zum Beispiel Repository.
servlet.jar	Für die JSP Seiten und Servlets der Benutzerschnittstelle.

Tabelle 6: Verwendete Bibliotheken im RDBD Modul.

Tabelle 6 listet die im RDBD Modul verwendeten Bibliotheken auf.

3 Beispiel Ausarbeitung

In diesem Kapitel werden die vom RDBD Modul zur Verfügung gestellten Funktionalitäten für die Beispiel Ausarbeitung beschrieben. Abschnitt 3.1 erläutert die Analyse Funktionalität des RDBD Moduls. Diese Beschreibung ist nach Beispieltypen gegliedert. Für Benutzer des RDBD Moduls ist in diesem Abschnitt ersichtlich, welche Information für das Analysieren einer Beispiel Ausarbeitung benötigt wird und welche Abweichungen in der Beispiel Ausarbeitung von der korrekten Lösung feststellt werden können. Entwickler des RDBD Moduls finden in diesem Abschnitt darüber hinaus eine Beschreibung der Realisierung der Analyse Funktionalität. Im Abschnitt 3.2 wird die Bewertungs-Funktionalität des RDBD Moduls beschrieben. Diese Funktionalität ist im RDBD Modul rudimentär realisiert. Im Abschnitt 3.3 wird auf die Funktionalität der Berichterstellung eingegangen. Benutzer des RDBD Moduls können in diesem Abschnitt einen Überblick über die vom RDBD Modul erzeugten Rückmeldungen in den verschiedenen Evaluationsarten und Diagnosestufen erhalten. Für Entwickler des RDBD Moduls ist darüber hinaus die Realisierung dieser Funktionalität beschrieben.

3.1 Analyse

Die algorithmische Natur der vom RDBD Modul unterstützten Beispieltypen erlaubt es, sehr feine Abweichungen einer Beispiel Ausarbeitung von der korrekten Lösung festzustellen. Für jede Art von Abweichungen wird im RDBD Modul ein eigener Analysator und eine eigene Analyse zur Verfügung gestellt. Das Vorhandensein trivialer Funktionaler Abhängigkeiten in einer Minimalen Überdeckung ist zum Beispiel eine konkrete Art von Abweichungen. Typischer Weise können beim Analysieren einer Beispiel Ausarbeitung abhängig vom Beispieltyp verschiedene Arten von Abweichungen festgestellt werden. Beim Analysieren einer Minimalen Überdeckung zum Beispiel können nicht nur triviale Funktionale Abhängigkeiten festgestellt werden, sondern auch redundante Funktionale Abhängigkeiten oder Funktionale Abhängigkeiten die sich nicht in kanonischer Darstellung befinden.

Abbildung 3 veranschaulicht den Zusammenhang zwischen Beispieltypen, festzustellenden Abweichungen, Analysen und Analysatoren. Für eine Ausarbeitung eines Beispiels vom Typ T1 wird vom entsprechenden Analysator eine Analyse erstellt. In einer Ausarbeitung eines Beispiels vom Typ T1 können Abweichungen der Arten A1, A2 und A3 festgestellt werden. Für jede dieser Arten von Abweichungen gibt es wiederum eine eigene Analyse und einen eigenen Analysator. Aus dieser Vorgehensweise ergibt sich, dass die Analyse T1 aus Analysen der Abweichungen der Arten A1, A2 und A3 besteht. Diese Verschachtelung von Analysen ermöglicht die Wiederverwendung von Klassen zur Repräsentation von Analysen. Gleichermaßen kann sich der Analysator T1 für das Analysieren einer Ausarbeitung eines Beispiel vom Typ T1 der Analysatoren A1, A2 und A3 bedienen. Durch diese Vorgehensweise wird auch bei den Implementierungen der Analysatoren eine hohe Wiederverwendung erreicht. Die Möglichkeiten zur Wiederverwendung von Analysen und Analysatoren ist vor allem auch Beispieltyp übergreifend möglich. Die Analyse T2 in Abbildung 3 kann auf die bereits in der Analyse T1 verwendete Analyse A1 zurückgreifen. Analog gilt dies für den Analysator T2.

Die Bewertung und das Berichterstellen basiert, entsprechend den Anforderungen des eTutor Cores auf den, beim Analysieren einer Beispiel Abgabe erstellten Analysen. Für diese beiden Funktionalitäten ist es notwendig anhand einer Analyse entscheiden zu können, ob die Beispiel Ausarbeitung korrekt ist oder nicht. Eine Beispielabgabe ist korrekt, sofern keine Abweichungen in der Beispiel Ausarbeitung von der korrekten Lösung festgestellt wurden. Um eine generische Möglichkeit zu bieten, anhand einer Analyse entscheiden zu können, ob eine Beispiel Ausarbeitung korrekt ist oder nicht, weist jede Analyse ein boolean Flag

`submissionSuitsSolution` auf. Hat dieses boolean Flag den Wert false, so sind in der jeweiligen Analyse Abweichungen festgehalten und die Beispiel Ausarbeitung ist nicht korrekt. Hat dieses boolean Flag den Wert true, so sind in der jeweiligen Analyse keine Abweichungen festgehalten und die Beispiel Ausarbeitung ist korrekt.

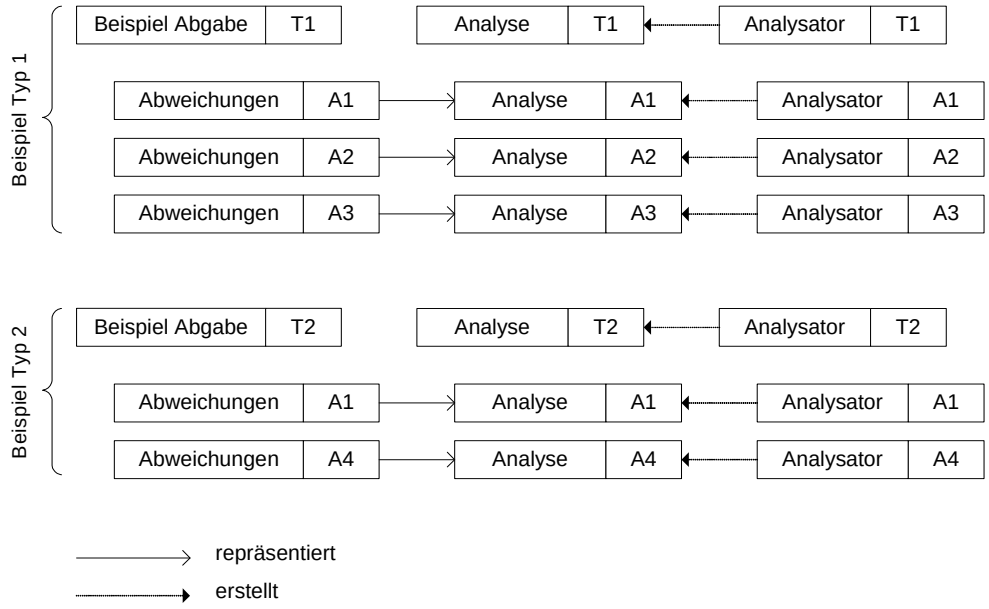

Abbildung 3: Zusammenhang von Beispieltypen, Abweichungen, Analysen und Analysatoren.

Beim Analysieren von Beispiel Ausarbeitungen werden verschiedene Algorithmen von diversen Analysatoren häufig verwendet. In den folgenden Beschreibungen der einzelnen Analysatoren wird, sofern einer dieser Algorithmen verwendet wird, nur mehr auf die Verwendung hingewiesen darüber hinaus aber nicht mehr auf die einzelnen Algorithmen eingegangen. Tabelle 7 listet diese Algorithmen auf. Diese Algorithmen sind im Package `etutor.modules.rdbd.algorithms` implementiert. Jeder Algorithmus ist in einer eigenen Klasse (benannt nach dem jeweiligen Algorithmus) implementiert. Für die Veranlassung der Ausführung eines solchen Algorithmus stellt jede dieser Klassen eine Methode `execute` zur Verfügung, die entsprechend den Anforderungen des Algorithmus parametrisiert ist. Die Implementierung dieser Algorithmen folgt exakt den Angaben im Skriptum zur Vorlesung Datenmodellierung.

Algorithmus	*Beschreibung*
Cover	Berechnet, ob zwei Mengen an Funktionale Abhängigkeiten die gleiche Hülle haben.
Closure	Berechnet die Hülle einer Menge an Attributen innerhalb einer Menge an Funktionalen Abhängigkeiten.
Minimal Cover	Berechnet die Minimale Überdeckung einer Menge an Funktionalen Abhängigkeiten.
Reduction By Resolution	Berechnet die in einer Teilrelation eingebetteten Funktionale Abhängigkeiten.

Tabelle 7: Häufig verwendete Algorithmen beim Analysieren von Beispiel Abgaben.

In den folgenden Abschnitten werden die einzelnen Analyse Vorgänge im RDBD Modul detailliert beschrieben. Aufgrund der Wiederverwendung von Analysatoren und Analysen werden

Analyse Vorgänge, deren Ergebnis die Gesamtanalyse einer Beispiel Ausarbeitung ist von solchen Analyse Vorgängen unterschieden, deren Ergebnis eine Teilanalyse ist. Leser, die sich lediglich einen Überblick über die Gesamtanalysen verschaffen wollen können aus Tabelle 8 entnehmen in welchem Abschnitt die jeweiligen Gesamtanalysen behandelt werden.

Gesamtanalyse für Beispieltyp	*Erläuterung*
RBR Algorithmus	Abschnitt 3.1.1
Decompose Algorithmus	Abschnitt 3.1.2
Allgemeine Normalisierung	Abschnitt 3.1.4
Attribut Hülle	Abschnitt 3.1.15
Schlüssel Bestimmen	Abschnitt 3.1.16
Finden von Normalform Verletzungen	Abschnitt 3.1.14
Minimale Überdeckung	Abschnitt 3.1.8

Tabelle 8: Überblick über die Erläuterungen zu Gesamtanalysen.

3.1.1 Gesamtanalyse - RBR Algorithmus

Bei diesem Analysevorgang wird die Gesamtanalyse der Abgabe eines Beispiels vom Typ RBR Algorithmus erstellt. Der RBR Algorithmus dient dazu, festzustellen welche Funktionalen Abhängigkeiten in einer Teilrelation einer Relation, im folgenden als Basisrelation bezeichnet, eingebettet sind.

Benötigte Information

- *Funktionale Abhängigkeiten der Basisrelation*: Die für die Berechnung der eingebetteten Funktionalen Abhängigkeiten zugrundeliegenden Funktionalen Abhängigkeiten.
- *Teilrelation*: Für die, in dieser Relation enthaltenen, Attribute werden die eingebetteten Funktionalen Abhängigkeiten berechnet.
- *Eingebettete Funktionale Abhängigkeiten*: Die vom Studenten gefundenen, eingebetteten Funktionalen Abhängigkeiten.

Festgestellte Abweichungen

- *Fehlende eingebettete Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die in der korrekten Lösung enthalten sind, nicht aber in der Lösung des Studenten.
- *Falsche eingebettete Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die in der Lösung des Studenten enthalten sind, nicht aber in der korrekten Lösung.

3.1.1.1 Realisierung

- *Analysators*: RBRAnalyzer
- *Aktivierende Methode*: analyze
- *Analyse*: RBRAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwartet die für den Analyse Vorgang notwendige Information in den beiden folgenden Parametern:

- `Relation baseRelation`: Die Funktionalen Abhängigkeiten dieser Relation repräsentieren die Funktionalen Abhängigkeiten der Basisrelation. Die Attribute und Schlüssel dieser Relation müssen nicht gesetzt sein.
- `Relation subRelation`: Die Attribute dieser Relation repräsentieren die Teilrelation, für die die eingebetteten Funktionalen Abhängigkeiten vom Studenten berechnet wurden. Die Funktionalen Abhängigkeiten dieser Relation repräsentieren die vom Studenten gefundenen eingebetteten Funktionalen Abhängigkeiten.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `RBRAnalysis` repräsentiert. Diese Klasse stellt zwei Listen `missingFunctionalDependencies` und `additionalFunctionalDependencies` zur Verfügung, in denen fehlende bzw. falsche eingebettete Funktionale Abhängigkeiten vermerkt werden können.

Beschreibung des Analyse Vorgangs

Die korrekten eingebetteten Funktionale Abhängigkeiten werden unter Verwendung des Algorithmus `ReductionByResolution` berechnet. Anschließend werden die korrekten eingebetteten Funktionalen Abhängigkeiten mit den Funktionalen Abhängigkeiten der Lösung des Studenten verglichen. Unter Zuhilfenahme des `Member` Algorithmus wird festgestellt, ob eine Funktionale Abhängigkeit der korrekten Lösung auch in der Lösung des Studenten enthalten ist und vice versa. Ist eine Funktionale Abhängigkeit in der korrekten Lösung enthalten, nicht aber in der Lösung des Studenten, wird diese als fehlende Funktionale Abhängigkeit in der Analyse des RBR Algorithmus vermerkt. Ist eine Funktionale Abhängigkeit in der Lösung des Studenten vorhanden, nicht aber in der korrekten Lösung, wird diese als falsche Funktionale Abhängigkeit in der Analyse des RBR Algorithmus vermerkt. Wurde eine fehlende oder eine falsche eingebettete Funktionale Abhängigkeit gefunden, wird in der Analyse des RBR Algorithmus vermerkt, dass die Lösung des Studenten nicht korrekt ist.

3.1.2 Gesamtanalyse - Decompose Algorithmus

Bei diesem Analysevorgang wird die Gesamtanalyse der Abgabe eines Beispiels vom Typ Decompose Algorithmus erstellt.

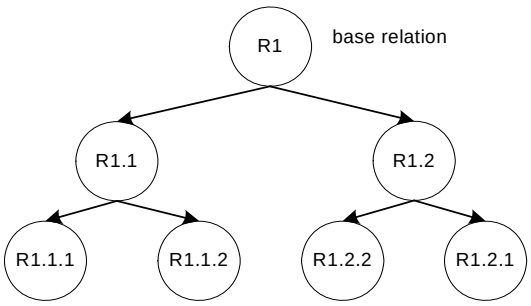

Abbildung 4: Ergebnis einer Normalisierung entsprechend dem Decompose Algorithmus.

Der Decompose Algorithmus dient zur Normalisierung einer Relation. Diese Relation wird im folgenden als Ausgangsrelation bezeichnet. Beim Decompose Algorithmus wird die Ausgangsrelation solange rekursiv in genau zwei Teilrelationen zerlegt, bis die entstandenen Teilrelationen die gewünschte Normalform aufweisen. Durch diese Vorgehensweise entsteht ein Baum aus Teilrelationen, wie in Abbildung 4 veranschaulicht. Dieser Baum wird im folgenden als Zerlegungsbaum bezeichnet. Jede Zerlegung einer Relation wird dabei als Zerlegungsschritt bezeichnet. Ein Zerlegungsschritt in Abbildung 4 ist zum Beispiel die Zerlegung der Relation R1.1 in die Teilrelationen R1.1.1 und R1.1.2. Das Ergebnis der Zerlegung stellen die Relationen in den Blattknoten des Zerlegungsbaums dar. Diese Relationen werden im folgenden als Ergebnisrelationen bezeichnet.

Benötigte Information

- *Ausgangsrelation*: Die zu normalisierende Relation. Die Attribute und Funktionalen Abhängigkeiten der Ausgangsrelation müssen bekannt sein.
- *Zerlegungsbaum*: Der vom Studenten erstellte Zerlegungsbaum. Für eine umfassende Analyse ist es nicht ausreichend, nur die Ergebnisrelationen zu berücksichtigen. Es ist notwendig, jeden einzelnen Zerlegungsschritt zu analysieren. Deshalb werden nicht nur die Ergebnisrelationen sondern der gesamte Zerlegungsbaums benötigt. Von jeder Relation im Zerlegungsbaum müssen die Attribute, Schlüssel und Funktionalen Abhängigkeiten bekannt sein.
- *Gewünschte Normalform*: Die Normalform in der sich die Ergebnisrelationen befinden müssen.
- *Maximale Anzahl an verlorenen Funktionalen Abhängigkeiten*: Bei der Normalisierung einer Ausgangsrelation nach dem Decompose Algorithmus kann es vorkommen, dass keine Abhängigkeitstreue Zerlegung gefunden werden kann. Für diesen Fall ist es notwendig, zu spezifizieren, wie viele Funktionale Abhängigkeiten höchstens verloren werden dürfen.

Festgestellte Abweichungen

- *Verlorene Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die in der Ausgangsrelation enthalten sind, aber aus den Funktionalen Abhängigkeiten der Ergebnisrelationen nicht mehr ableitbar sind.
- *Abweichungen in einzelnen Zerlegungsschritten*: Für jeden Zerlegungsschritt werden bestimmte Abweichungen festgehalten. Für diesen Analyse Vorgang ist ein eigener Analysator zuständig. Näheres zu diesem Analyse Vorgang finden Sie im Abschnitt 3.1.3.

3.1.2.1 Realisierung

- *Analysators*: DecomposeAnalyzer
- *Aktivierende Methode*: analyze
- *Analyse*: DecomposeAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwartet die für den Analyse Vorgang notwendige Information in einem Objekt der Klasse `DecomposeAnalyzerConfig`. Abbildung 5 veranschaulicht den Aufbau dieser Klasse.

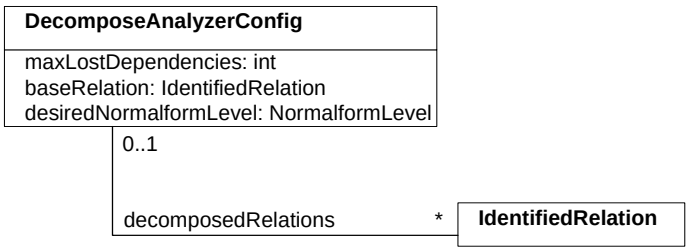

Abbildung 5: Aufbau der Klasse DecomposeAnalyzerConfig.

Im Property `maxLostDependencies` wird die maximal erlaubte Anzahl an verlorenen Funktionalen Abhängigkeiten repräsentiert. Soll spezifiziert werden, dass beliebig viele Funktionale Abhängigkeiten verloren werden dürfen, so muss die angegebene Anzahl größer als die Anzahl an Funktionalen Abhängigkeiten der Ausgangsrelation sein. Im Property `baseRealtion` wird die Ausgangsrelation repräsentiert. Die Attribute und Funktionalen Abhängigkeiten dieser Relation müssen gesetzt sein. Im Property `desiredNormalformLevel` wird die Normalform repräsentiert, in der sich die Ergebnisrelationen befinden müssen. Im Property `decomposedRelations` wird der, vom Studenten erstellte Zerlegungsbaum repräsentiert. Es ist erforderlich, dass die darin enthaltenen Teil-/Ergebnisrelationen in Objekten der Klasse `IdentifiedRelation` repräsentiert werden. Die Klasse `IdentifiedRelation` erbt von der Klasse `Relation` und erweitert diese um ein Property `ID`. Diese `ID` muss entsprechend der IDs in Abbildung 4 gestaltet sein. Durch diese `IDs` wird es ermöglicht, aus den einzelnen Teil-/Ergebnisrelationen den Zerlegungsbaum zu rekonstruieren. Bei jeder dieser Relationen müssen Attribute, Funktionale Abhängigkeiten und Schlüssel gesetzt sein.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `DecomposeAnalysis` repräsentiert. Abbildung 6 veranschaulicht den Aufbau dieser Klasse.

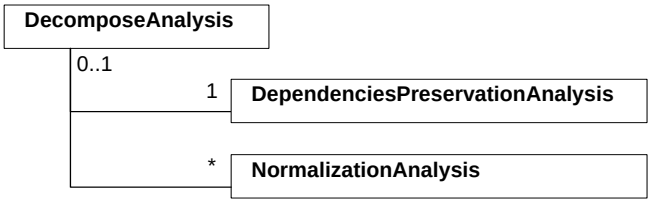

Abbildung 6: Aufbau der Klasse DecomposeAnalysis.

Bei der Zerlegung verlorene Funktionale Abhängigkeiten werden in einer enthaltenen Analyse der Abhängigkeitstreue einer Zerlegung repräsentiert (siehe `DependenciesPreservationAnalysis` in Abbildung 6). Abweichungen in den einzelnen Zerlegungsschritten werden in enthaltenen Analysen einer Allgemeinen Normalisierung repräsentiert (siehe `NormalizationAnalysis` in Abbildung 6). Da in der Analyse eines Zerlegungsschritts im Decompose Algorithmus die gleichen Abweichungen festgehalten werden, wie in der Analyse einer Allgemeinen Normalisierung wurde die Klasse `NormalizationAnalysis` zum Zwecke der Repräsentation der Abweichungen wiederverwendet.

Beschreibung des Analyse Vorgangs

Die Analyse der Abhängigkeitstreue wird unter Verwendung des Analysators der Abhängigkeitstreue einer Zerlegung erstellt (siehe Abschnitt 3.1.7). Dem Analysator der

Abhängigkeitstreue einer Zerlegung werden die Funktionalen Abhängigkeiten der Ausgangsrelation und die Funktionalen Abhängigkeiten der Ergebnisrelationen übergeben. Die Funktionalen Abhängigkeiten der Teilrelationen des Zerlegungsbaums werden nicht berücksichtigt, da nur ausschlaggebend ist, ob die Funktionalen Abhängigkeiten der Ausgangsrelation aus den Funktionalen Abhängigkeiten der Ergebnisrelationen ableitbar sind. Ist die Anzahl an nicht mehr ableitbaren (verlorenen) Funktionalen Abhängigkeiten in der Analyse der Abhängigkeitstreue einer Zerlegung größer als die maximal erlaubte Anzahl an verlorenen Funktionalen Abhängigkeiten, wird in der Analyse des Decompose Algorithmus vermerkt, dass die Lösung des Studenten nicht korrekt ist.

Um Abweichungen in den einzelnen Zerlegungsschritten festzustellen wird der Analysator für Zerlegungsschritte im Decompose Algorithmus für jeden Zerlegungsschritt aktiviert (siehe Abschnitt 3.1.3). Ist in einer der resultierenden Analysen vermerkt, dass der Zerlegungsschritt nicht korrekt durchgeführt wurde, wird in der Analyse des Decompose Algorithmus vermerkt, dass die Lösung des Studenten nicht korrekt ist.

3.1.3 Teilanalyse - Zerlegungsschritt im Decompose Algorithmus

Bei diesem Analysevorgang wird die Analyse eines Zerlegungsschritts im Decompose Algorithmus erstellt. Ein Zerlegungsschritt besteht dabei aus einer Ausgangsrelation und genau zwei Teilrelationen, die die Zerlegung der Ausgangsrelation darstellen.

Benötigte Information

- *Ausgangsrelation*: Zu dieser Relation müssen die Attribute und Funktionalen Abhängigkeiten bekannt sein.
- *Teilrelationen*: Die beiden Teilrelationen des Zerlegungsschritts. Zu jeder dieser Relationen müssen die Attribute, Funktionalen Abhängigkeiten und Schlüssel bekannt sein.
- *Geforderte Normalform*: Die Normalform die die Ergebnisrelationen aufweisen müssen. Diese Information wird nur für solche Zerlegungsschritte benötigt, bei denen zumindest eine der Teilrelationen auch eine Ergebnisrelation ist.

Festgestellte Abweichungen

Folgende Abweichungen werden für die beiden Teilrelationen der Lösung des Studenten gemeinsam festgestellt:

- *Fehlende Attribute in der Zerlegung*: Attribute der Ausgangsrelation, die in keiner der beiden Teilrelationen enthalten sind. Trifft dies zu, handelt es sich um eine ungültige Zerlegung.
- *Verlustfreiheit der Zerlegung*: Es wird vermerkt, ob die Teilrelationen eine verlustfreie Zerlegung der Ausgangsrelation darstellen.

Folgende Abweichungen werden für jede der beiden Teilrelationen der Lösung des Studenten einzeln festgestellt:

- *Nicht kanonisch dargestellte Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten der Teilrelation die sich nicht in kanonischer Darstellung befinden.

- *Triviale Funktionale Abhängigkeiten*: Triviale Funktionale Abhängigkeiten die in der Teilrelation enthalten sind.
- *Überflüssige Attribute*: Attribute die in einzelnen Funktionalen Abhängigkeiten der Teilrelation überflüssig sind.
- *Redundante Funktionale Abhängigkeiten*: Redundante Funktionale Abhängigkeiten die in der Teilrelation enthalten sind.
- *Falsche eingebettete Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die in der Teilrelation enthalten sind, sich aber nicht aus den Funktionalen Abhängigkeiten der Ausgangsrelation ableiten lassen.
- *Fehlende eingebettete Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die sich aus den Funktionalen Abhängigkeiten der Ausgangsrelation ableiten liesen, aber nicht in den Funktionalen Abhängigkeiten der Teilrelation enthalten sind.
- *Fehlende Schlüssel*: Schlüssel, die in der Teilrelation enthalten sein sollten dies aber nicht sind.
- *Falsche Schlüssel*: Schlüssel die in der Teilrelation angegeben wurden, aber keine Schlüssel der Teilrelation sind.
- *Unzureichende Normalform*: Teilrelationen die auch Ergebnisrelationen sind aber nicht die gewünschte Normalform aufweisen.

3.1.3.1 Realisierung

- *Analysator*: DecomposeStepAnalyzer
- *Aktivierende Methode*: analyze
- *Analyse*: DecomposeStepAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwartet die für den Analyse Vorgang notwendige Information in einem Objekt der Klasse `DecomposeStepAnalyzerConfig`. Abbildung 7 zeigt den Aufbau dieser Klasse.

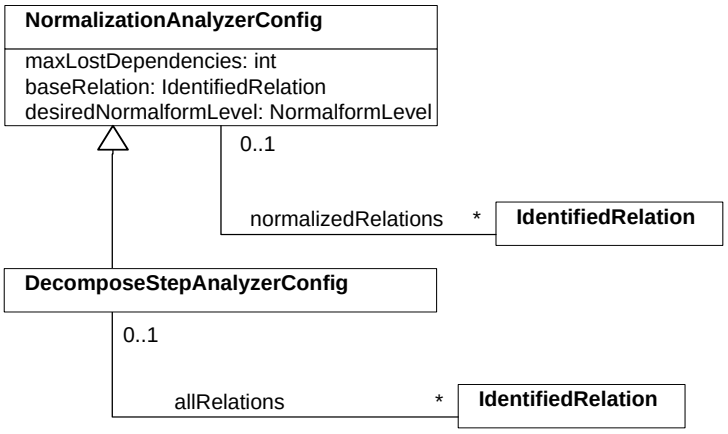

Abbildung 7: Aufbau der Klasse DecomposeStepAnalyzerConfig.

Die Klasse `DecomposeStepAnalyzerConfig` erbt die Eigenschaften der Klasse `NormalizationAnalyzerConfig`. Im Property `baseRelation` wird die Ausgangsrelation des

Zerlegungsschritts repräsentiert. Die Attribute und Funktionalen Abhängigkeiten dieser Relation müssen gesetzt sein. Im Property `desiredNormalformLevel` wird die Normalform, in der sich Ergebnisrelationen befinden müssen, repräsentiert. Im Property `normalizedRelations` werden die beiden Teilrelationen des Zerlegungsschritts repräsentiert. Bei diesen Relationen müssen Attribute, Funktionale Abhängigkeiten und Schlüssel gesetzt sein. Die bisher erläuterten Properties der Klasse `DecomposeStepAnalyerConfig` ermöglichen grundsätzlich die zuvor beschriebene, benötigte Information für den Analyse Vorgang zu repräsentieren. Allerdings wäre es nicht möglich zu entscheiden, ob es sich bei einer Teilrelation um eine Ergebnisrelation handelt oder nicht, da diese Entscheidung auf Basis der IDs der Teilrelationen getroffen wird. Deshalb ist es notwendig zusätzlich alle Relationen des Zerlegungsbaums zu kennen. Diese Relationen werden im Property `allRelations` repräsentiert. Sowohl die Relationen im Property `normalizedRelations` als auch die Relationen im Property `allRelations` müssen in Objekten der Klasse `IdentifiedRelation` repräsentiert werden.

Repräsentation der festgestellten Abweichungen

Da die, beim Analysieren eines Zerlegungsschritts im Decompose Algorithmus festgestellten, Abweichungen denen gleichen, die beim Analysieren einer Allgemeinen Normalisierung festgestellt werden, wird die Klasse `NormalizationAnalysis` für die Repräsentation der festgestellten Abweichungen wiederverwendet (siehe Abschnitt 3.1.4).

Beschreibung des Analyse Vorgangs

Die Vorgehensweise beim Analysieren eines Zerlegungsschritts im Decompose Algorithmus gleicht der beim Analysieren einer Allgemeinen Normalisierung (siehe Abschnitt 3.1.4) bis auf die beiden folgenden Unterschiede:

- *Analysieren der Abhängigkeitstreue*: Es wird zwar analog zum Analysieren einer Allgemeinen Normalisierung festgestellt, welche Funktionalen Abhängigkeiten in den Teilrelationen verloren wurden, es wird aber in der Analyse des Zerlegungsschritts nicht vermerkt, dass die Lösung des Studenten nicht korrekt ist, sofern verlorene Funktionale Abhängigkeiten festgestellt werden. Der Grund dafür ist, dass die Abhängigkeitstreue im Beispieltyp Decompose Algorithmus nur zwischen der Ausgangsrelation und den Ergebnisrelationen festgestellt wird und nicht jede Teilrelation eines Zerlegungsschritt auch eine Ergebnisrelation ist. Die insgesamte Abhängigkeitstreue der Zerlegung wird, wie im Abschnitt 3.1.2 erläutert, vor der Analyse der einzelnen Zerlegungsschritte bestimmt.
- *Analysieren der Normalform*: Beim Analysieren der Normalform einer Teilrelation wird unterschieden, ob es sich um eine Ergebnisrelation handelt oder nicht. Wird eine Abweichung in der Normalform einer Teilrelation von der gewünschten Normalform festgestellt und handelt es sich dabei um eine Ergebnisrelation, so wird in der Analyse des Zerlegungsschritts vermerkt, dass die Lösung des Studenten nicht korrekt ist. Wird eine Abweichung in der Normalform einer Teilrelation von der gewünschten Normalform festgestellt, die keine Ergebnisrelation ist, so wird in der Analyse des Zerlegungsschritts nicht vermerkt, dass die Beispiel Abgabe nicht korrekt ist. Der Grund dafür ist wiederum, dass nur die Ergebnisrelationen die gewünschte Normalform erfüllen müssen.

3.1.4 Gesamtanalyse - Allgemeine Normalisierung

Bei diesem Analysevorgang wird die Gesamtanalyse der Abgabe eines Beispiels vom Typ Allgemeine Normalisierung erstellt.

Bei der Allgemeinen Normalisierung wird eine Relation, im folgenden als Basisrelation bezeichnet, so in Teilrelationen zerlegt, dass diese Teilrelationen eine gewünschte Normalform aufweisen. Diese Teilrelationen werden im Folgenden als Ergebnisrelationen bezeichnet. Im Unterschied zur Normalisierung mittels Decompose wird bei der Allgemeinen Normalisierung kein bestimmter Lösungsweg vorgegeben.

Benötigte Information

- *Basisrelation*: Die zu normalisierende Relation. Die Attribute und Funktionalen Abhängigkeiten dieser Relation müssen bekannt sein.
- *Ergebnisrelationen*: Die vom Studenten berechneten Ergebnisrelationen. Die Attribute, Funktionalen Abhängigkeiten und Schlüssel der Ergebnisrelationen müssen bekannt sein.
- *Geforderte Normalform*: Die Normalform in der sich die Ergebnisrelationen befinden müssen.
- *Maximale Anzahl an verlorenen Funktionalen Abhängigkeiten*: Bei der Allgemeinen Normalisierung einer Ausgangsrelation kann es vorkommen, dass keine Abhängigkeitstreue Zerlegung gefunden werden kann. Für diesen Fall ist es notwendig, zu spezifizieren, wie viele Funktionale Abhängigkeiten höchstens verloren werden dürfen.

Festgestellte Abweichungen

Folgende Abweichungen werden für alle Ergebnisrelationen gemeinsam, das bedeutet für die Normalisierung als Ganzes, festgehalten:

- *Fehlende Attribute in der Zerlegung*: Attribute der Ausgangsrelation, die in keiner der Ergebnisrelationen enthalten sind. Wird ein solches Attribut identifiziert, handelt es sich um eine ungültige Zerlegung.
- *Verlustfreiheit der Zerlegung*: Es wird vermerkt ob die Ergebnisrelationen eine verlustfreie Zerlegung der Basisrelation darstellen.
- *Fehlende Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten der Basisrelation die aus den Funktionalen Abhängigkeiten der Ergebnisrelationen nicht ableitbar sind.

Folgende Abweichungen werden für jede Ergebnisrelation einzeln festgestellt:

- *Nicht kanonisch dargestellte Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten der Ergebnisrelation die sich nicht in kanonischer Darstellung befinden.
- *Triviale Funktionale Abhängigkeiten*: Triviale Funktionale Abhängigkeiten die in der Ergebnisrelation enthalten sind.
- *Überflüssige Attribute*: Attribute die in einzelnen Funktionalen Abhängigkeiten der Ergebnisrelation überflüssig sind.
- *Redundante Funktionale Abhängigkeiten*: Redundante Funktionale Abhängigkeiten die in der Ergebnisrelation enthalten sind.
- *Falsche eingebettete Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die in der Ergebnisrelation enthalten sind, sich aber nicht aus den Funktionalen Abhängigkeiten der Ausgangsrelation ableiten lassen.
- *Fehlende eingebettete Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die sich aus den Funktionalen Abhängigkeiten der Ausgangsrelation ableiten liesen aber nicht in den Funktionalen Abhängigkeiten der Ergebnisrelation enthalten sind.

- *Fehlende Schlüssel*: Schlüssel, die in der Ergebnisrelation enthalten sein sollten dies aber nicht sind.
- *Falsche Schlüssel*: Schlüssel die in der Ergebnisrelation angegeben wurden, aber keine Schlüssel der Ergebnisrelation sind.
- *Unzureichende Normalform*: Ergebnisrelationen die sich nicht in der gewünschten Normalform befinden.

3.1.4.1 Realisierung

- *Analysator*: NormalizationAnalyzer
- *Aktivierende Methode*: analyze
- *Analyse*: NormalizationAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwatet die für den Analyse Vorgang benötigte Information in einem Objekt der Klasse `NormalizationAnalyzerConfig`. Abbildung 8 veranschaulicht den Aufbau dieser Klasse.

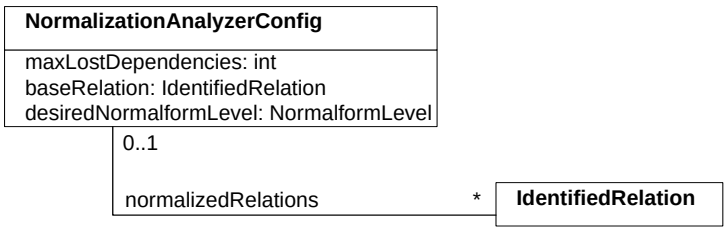

Abbildung 8: Aufbau der Klasse NormalizationAnalyzerConfig.

Im Property `maxLostDependencies` wird die maximal erlaubte Anzahl an verlorenen Funktionalen Abhängigkeiten repräsentiert. Soll spezifiziert werden, dass beliebig viele Funktionale Abhängigkeiten verloren werden dürfen, so muss die angegebene Anzahl größer als die Anzahl an Funktionalen Abhängigkeiten der Ausgangsrelation sein. Im Property `baseRealtion` wird die Ausgangsrelation repräsentiert. Die Attribute und Funktionalen Abhängigkeiten dieser Relation müssen gesetzt sein. Im Property `desiredNormalformLevel` wird die Normalform repräsentiert, in der sich die Ergebnisrelationen befinden müssen. Im Property `normalizedRelations` werden die vom Studenten berechneten Ergebnisrelationen repräsentiert. Bei jeder dieser Relationen müssen Attribute, Funktionale Abhängigkeiten und Schlüssel gesetzt sein.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `NormalizationAnalysis` repräsentiert. Abbildung 9 veranschaulicht den Aufbau dieser Klasse.

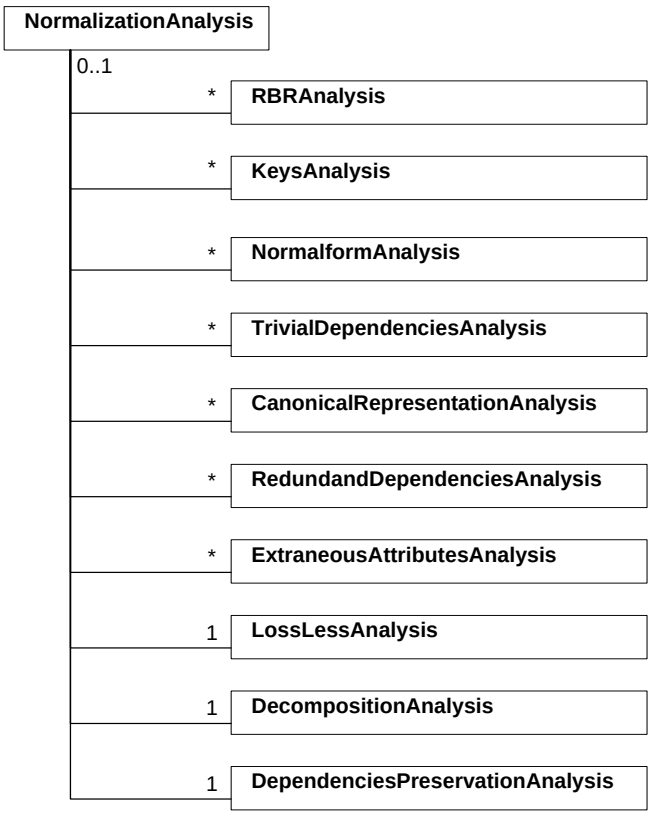

Abbildung 9: Aufbau der Klasse NormalizationAnalysis.

Fehlende bzw. falsche eingebettete Funktionale Abhängigkeiten einer Ergebnisrelation werden in Analysen des RBR Algorithmus festgehalten (vgl. `RBRAnalysis` in Abbildung 9). Fehlende bzw. falsche Schlüssel einer Ergebnisrelation werden in Analysen der Schlüssel Bestimmung festgehalten (vgl. `KeysAnalysis` in Abbildung 9). Ergebnisrelationen die sich nicht in der gewünschten Normalform befinden, werden in Analysen der Normalform einer Relation festgehalten (vgl. `NormalformAnalysis` in Abbildung 9). Triviale Funktionale Abhängigkeiten einer Ergebnisrelation werden in Analysen Trivialer Funktionaler Abhängigkeiten vermerkt (vgl. `TrivialDependenciesAnalysis` in Abbildung 9). Funktionale Abhängigkeiten einer Ergebnisrelationen, die sich nicht in kanonischer Darstellung befinden werden in Analysen für Nicht kanonisch dargestellte Funktionale Abhängigkeiten vermerkt (vgl. `CanonicalRepresentationAnalysis` in Abbildung 9). Redundante Funktionale Abhängigkeiten einer Ergebnisrelation, werden in Analysen Redundanter Funktionaler Abhängigkeiten vermerkt (vgl. `RedundandDependenciesAnalysis` in Abbildung 9). Überflüssige Attribute in Funktionalen Abhängigkeiten einer Ergebnisrelationen, werden in Analysen Überflüssiger Attribute vermerkt (vgl. `ExtranuousAttributesAnalysis` in Abbildung 9). Fehlende Attribute in der Zerlegung werden in einer Analyse der Gültigkeit einer Zerlegung vermerkt (vgl. `DecompositionAnalysis` in Abbildung 9). Falls die Verlustfreiheit der Zerlegung nicht gegeben ist, wird dies in der Analyse der Verlustfreiheit einer Zerlegung vermerkt (vgl. `LossLessAnalysis` in Abbildung 9). Fehlende Funktionale Abhängigkeiten in den Ergebnisrelationen, also solche, die nicht mehr aus den Funktionalen Abhängigkeiten der Basisrelation ableitbar sind, werden in einer Analyse der Abhängigkeitstreue einer Zerlegung vermerkt (vlg. `DependenciesPreservationAnalysis` in Abbildung 9).

Beschreibung des Analyse Vorgangs

Um zu überprüfen, ob die Ergebnisrelationen eine gültige Zerlegung der Basisrelation darstellen, wird der Analysator für die Gültigkeit einer Zerlegung aktiviert (siehe Abschnitt 3.1.5). Diesem

Analysator werden die Basisrelation und die Ergebnisrelationen übergeben. Die resultierende Analyse der Gültigkeit einer Zerlegung wird der Analyse der Allgemeinen Normalisierung hinzugefügt. Wurde festgestellt, dass es sich um keine gültige Zerlegung handelt, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analyse Vorgang wird abgebrochen. Dieser Vorgehensweise liegt die Überlegung zu Grunde, dass jede Zerlegung zumindest gültig sein muss.

Um zu überprüfen, ob die Ergebnisrelationen eine Verlustfreie Zerlegung der Basisrelation darstellen, wird der Analysator für die Verlustfreiheit einer Zerlegung aktiviert (siehe Abschnitt 3.1.6). Die resultierende Analyse der Verlustfreiheit einer Zerlegung wird der Analyse der Allgemeinen Normalisierung hinzugefügt. Wird erkannt, dass es sich um keine verlustfreie Zerlegung handelt, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen. Dieser Vorgehensweise liegt die Überlegung zu Grunde, dass jede Zerlegung zumindest verlustfrei sein muss.

Um fehlende Funktionale Abhängigkeiten in den Ergebnisrelationen zu identifizieren, wird der Analysator für die Abhängigkeitstreue einer Zerlegung aktiviert (siehe Abschnitt 3.1.7). Die resultierende Analyse der Abhängigkeitstreue einer Zerlegung wird der Analyse der Allgemeinen Normalisierung hinzugefügt. Wird erkannt, dass mehr Funktionale Abhängigkeiten bei der Zerlegung verloren wurden als erlaubt, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

Da der Analysator für die Abhängigkeitstreue einer Zerlegung als Input Relationen erwartet, die keine trivialen bzw. redundanten Funktionalen Abhängigkeiten enthalten und deren Funktionale Abhängigkeiten keine überflüssigen Attribute aufweisen und sich in kanonischer Darstellung befinden, werden vor der Aktivierung dieses Analysators noch folgende Schritte ausgeführt.

1. Es wird für jede Ergebnisrelation überprüft, ob sich die darin enthaltenen Funktionalen Abhängigkeiten in kanonischer Darstellung befinden. Zu diesem Zweck wird für jede Ergebnisrelation der Analysator der Kanonischen Darstellung von Funktionalen Abhängigkeiten aktiviert (siehe Abschnitt 3.1.12). Die resultierenden Analysen der Kanonischen Darstellung von Funktionalen Abhängigkeiten werden der Analyse der Allgemeinen Normalisierung hinzugefügt. Wird erkannt, dass eine Funktionale Abhängigkeit einer Ergebnisrelation sich nicht in kanonischer Darstellung befindet, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

2. Es wird für jede Ergebnisrelation überprüft, ob darin triviale Funktionale Abhängigkeiten enthaltenen sind. Zu diesem Zweck wird für jede Ergebnisrelation der Analysator für Triviale Funktionale Abhängigkeiten aktiviert (siehe Abschnitt 3.1.11). Die resultierenden Analysen Trivialer Funktionaler Abhängigkeiten werden der Analyse der Allgemeinen Normalisierung hinzugefügt. Wird eine triviale Funktionale Abhängigkeit erkannt, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

3. Es wird für jede Ergebnisrelation überprüft, ob darin redundante Funktionale Abhängigkeiten enthaltenen sind. Zu diesem Zweck wird für jede Ergebnisrelation der Analysator für Redundante Funktionale Abhängigkeiten aktiviert (siehe Abschnitt 3.1.10). Die resultierenden Analysen Redundanter Funktionaler Abhängigkeiten werden der Analyse der Allgemeinen Normalisierung hinzugefügt. Wird eine redundante Funktionale Abhängigkeit erkannt, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

4. Es wird für jede Ergebnisrelation überprüft, ob darin Funktionale Abhängigkeiten enthaltenen sind, die überflüssige Attribute aufweisen. Zu diesem Zweck wird für jede Funktionale Abhängigkeit jeder Ergebnisrelation der Analysator für Überflüssige Attribute in Funktionalen Abhängigkeiten aktiviert (siehe Abschnitt 3.1.9). Die resultierenden Analysen Überflüssiger Attribute in Funktionalen Abhängigkeiten werden der Analyse der Allgemeinen Normalisierung hinzugefügt. Wird eine Funktionale Abhängigkeit erkannt in der zumindest ein überflüssiges Attribut enthalten ist, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

Um zu überprüfen, ob in den Ergebnisrelationen der Lösung des Studenten korrekte Schlüssel angegeben wurden, wird für jede dieser Teilrelationen der Analysator für das Bestimmen von Schlüsseln aktiviert (siehe Abschnitt 3.1.16). Dieser Analysator benötigt als Input einerseits die korrekten Schlüssel der jeweiligen Ergebnisrelation und andererseits die zu überprüfenden Schlüssel. Deshalb werden unter Verwendung der Klasse `KeysDeterminator` für jede Teilrelation der Lösung des Studenten die korrekten Schlüssel berechnet. Die Berechnung der korrekten Schlüssel einer Relation wird im Abschnitt 3.1.16 näher erläutert. Die resultierenden Analysen der Schlüssel einer Relation werden in der Analyse der Allgemeinen Normalisierung vermerkt. Wird erkannt, dass in einer der Ergebnisrelationen ein falscher Schlüssel angegeben wurde, oder dass ein korrekter Schlüssel nicht angegeben wurde, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

Um zu überprüfen, ob in den Ergebnisrelationen korrekte eingebettete Funktionale Abhängigkeiten nicht angegeben wurden bzw. falsche eingebettete Funktionale Abhängigkeiten angegeben wurden, wird für jede Ergebnisrelation der Analysator für den RBR Algorithmus aktiviert (siehe Abschnitt 3.1.1). Dieser Analysator stellt Abweichungen in eingebetteten Funktionalen Abhängigkeiten fest. Die resultierenden Analysen werden der Analyse der Allgemeinen Normalisierung hinzugefügt. Wird erkannt, dass eine eingebettete Funktionale Abhängigkeit nicht angegeben wurde bzw. eine Funktionale Abhängigkeit fälschlicher Weise als eingebettet angegeben wurde, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

Um zu überprüfen, ob die Ergebnisrelationen auch der geforderten Normalform entsprechen, wird für jede Ergebnisrelation der Analysator der Normalform einer Relation aktiviert (siehe Abschnitt 3.1.14). Die resultierenden Analysen der Normalformen von Relationen werden der Analyse der Allgemeinen Normalisierung hinzugefügt. Wird erkannt, dass sich eine der Ergebnisrelationen nicht in der gewünschten Normalform befindet, wird in der Analyse der Allgemeinen Normalisierung festgehalten, dass die Lösung des Studenten nicht korrekt ist.

3.1.5 Teilanalyse - Gültigkeit einer Zerlegung

In diesem Analyse Vorgang wird die Gültigkeit einer Zerlegung festgestellt. Es wird festgestellt, ob jedes Attribut einer Basisrelation in zumindest einer der Teilrelationen der Zerlegung enthalten ist.

Benötigte Information

- *Basisrelation*: Die der Zerlegung zugrundeliegende Relation. Nur die Attribute dieser Relation müssen bekannt sein.
- *Teilrelationen*: Die vom Studenten gebildeten Teilrelationen. Auch von jeder Teilrelation müssen nur die Attribute bekannt sein.

Festgestellte Abweichungen

- *Fehlende Attribute*: Attribute der Basisrelation, die in keiner der Teilrelationen der Lösung des Studenten enthalten sind.

3.1.5.1 Realisierung

- *Analysator*: NormalizationAnalyzer
- *Aktivierende Methode*: analyzeDecomposition
- *Analyse*: DecompositionAnalysis

Repräsentation der benötigten Information

Die Methode `analyzeDecomposition` erwartet die für den Analyse Vorgang benötigte Information in den beiden folgenden Parametern:

- `Relation baseRelation`: Die der Zerlegung zugrundeliegende Relation. In dieser Relation müssen lediglich die Attribute gesetzt sein. Funktionale Abhängigkeiten und Schlüssel werden nicht benötigt.
- `Collection decomposedRelations`: Die vom Studenten gebildeten Teilrelationen. Bei diesen Teilrelationen müssen ebenfalls nur die Attribute gesetzt sein. Funktionale Abhängigkeiten und Schlüssel werden nicht benötigt.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der klasse `DecompositionAnalysis` repräsentiert. Diese Klasse stellt eine Liste `missingAttributes` zur Verfügung, in der fehlende Attribute vermerkt werden können.

Beschreibung des Analysevorgangs

Um die fehlenden Attribute einer Zerlegung zu identifizieren, werden die Teilrelationen der Lösung des Studenten iteriert und die Attribute der jeweiligen Teilrelation von den Attributen der Basisrelation (gedanklich) gestrichen. Enthält die Basisrelation nach diesem Vorgang noch Attribute, werden diese als fehlende Attribute in der Analyse der Gültigkeit einer Zerlegung hinzugefügt. Weiters wird in der Analyse vermerkt, dass die Lösung des Studenten nicht korrekt ist.

Anmerkung: Prinzipiell könnte in einer der Teilrelationen auch ein Attribut enthalten sein, dass nicht in der Basisrelation enthalten ist. Eine solche Teilrelation würde auch zu einer ungültigen Zerlegung führen. Die Editoren für die Beispielausarbeitung der Benutzerschnittstelle verhindern es allerdings, dass in einer Teilrelation ein Attribut enthalten ist, dass nicht auch in der Basisrelation enthalten ist. Deshalb werden solche zusätzlichen Attribute bei der Analyse der Gültigkeit einer Zerlegung nicht berücksichtigt.

3.1.6 Teilanalyse - Verlustfreiheit einer Zerlegung

In diesem Analyse Vorgang wird festgestellt, ob durch einen Verbund der vom Studenten gefundenen Teilrelationen einer Zerlegung wieder die zugrundeliegende Basisrelation entsteht. Trifft dies nicht zu, liegt keine verlustfreie Zerlegung vor.

Benötigte Information

- *Basisrelation*: Die der Zerlegung zugrundeliegende Relation. Die Attribute und Funktionalen Abhängigkeiten dieser Relation müssen bekannt sein.
- *Teilrelationen*: Die vom Studenten gebildeten Teilrelationen. Auch von diesen Relationen müssen die Attribute und Funktionalen Abhängigkeiten bekannt sein.

Festgestellte Abweichungen

- *Verlustfreie Zerlegung*: Angabe ob Zerlegung verlustfrei ist oder eben nicht. Ist eine Zerlegung nicht verlustfrei, so kann dies lediglich festgestellt werden, es kann aber keine spezifischere Begründung eruiert werden. Es handelt sich genau dann um eine nicht verlustfreie Zerlegung, wenn in keiner der Hüllen der Attribute der Teilrelationen, gebildet in der Vereinigung aller Funktionalen Abhängigkeiten der Teilrelationen, alle Attribute der Basisrelation enthalten sind. Somit kann lediglich festgehalten werden, dass eben keine Teilrelation diesen Umstand erfüllt, was gleichermaßen für jede nicht verlustfreie Zerlegung gilt.

3.1.6.1 Realisierung

- *Analysator*: NormalizationAnalyzer
- *Aktivierende Methode*: analyzeLossLess
- *Analyse*: LossLessAnalysis

Repräsentation der benötigten Information

Die Methode `analyzeLossLess` erwartet die für den Analyse Vorgang benötigte Information in den beiden folgenden Input Parametern:

- `Relation baseRelation`: Die der Zerlegung zugrundeliegende Relation. Die Attribute und Funktionalen Abhängigkeiten dieser Relation müssen gesetzt sein. Die Schlüssel dieser Relation müssen nicht gesetzt sein.
- `Collection decomposedRelations`: Die vom Studenten gebildeten Teilrelationen. Die Attribute und Funktionalen Abhängigkeiten dieser Teilrelationen müssen gesetzt sein. Die Schlüssel dieser Teilrelationen müssen nicht gesetzt sein.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `LossLessAnalysis` dargestellt. Diese Klasse hat keine über die Eigenschaften einer allgemeinen Analyse (`etutor.core.evaluation.Analysis`) hinausgehenden Eigenschaften. Es wird lediglich im Property `submissionSuitsSolution` vermerkt, ob die Zerlegung der Lösung des Studenten verlustfrei ist oder nicht.

Beschreibung des Analyse Vorgangs

Dem Analyse Vorgang liegt die Überlegung zu Grunde, dass eine Zerlegung genau dann verlustfrei ist, sofern die Attribut Hülle von zumindest einer Teilrelation berechnet in der Menge der Funktionalen Abhängigkeiten aller Teilrelationen alle Attribute der Basisrelation enthält. Diese Grundüberlegung ist im Analysator der Verlustfreiheit einer Zerlegung folgendermaßen umgesetzt:

1. Die Teilrelationen werden iteriert und die jeweiligen Funktionalen Abhängigkeiten werden temporär in einer Liste abgelegt.

2. Die Teilrelationen werden ein zweites mal iteriert. Für jede Teilrelation wird der Algorithmus `Closure` aufgerufen. Diesem Algorithmus werden einerseits die Attribute der jeweiligen Teilrelation und andererseits die zuvor berechneten Funktionalen Abhängigkeiten mitgegeben. Sind im Ergebnis dieses Algorithmus alle Attribute der Basisrelation enthalten, wird der Analysevorgang abgebrochen und in der Analyse der Verlustfreiheit einer Zerlegung vermerkt, dass die Lösung des Studenten der berechneten Lösung entspricht.

3. Wurden alle Teilrelationen durchiteriert und keine Teilrelation gefunden, die das entsprechende Kriterium erfüllt, wird in der Analyse der Verlustfreiheit einer Zerlegung vermerkt, dass die Lösung des Studenten nicht der berechneten Lösung entspricht.

3.1.7 Teilanalyse - Abhängigkeitstreue einer Zerlegung

Bei diesem Analyse Vorgang wird festgestellt, ob aus den Funktionalen Abhängigkeiten der Teilrelationen einer Zerlegung alle Funktionalen Abhängigkeiten der zugrundeliegenden Basisrelation ableitbar sind.

Benötigte Information

- *Basisrelation*: Die der Zerlegung zugrundeliegende Relation. Die Funktionalen Abhängigkeiten dieser Relation müssen bekannt sein.
- *Normalisierte Teilrelationen*: Die vom Studenten gebildeten Teilrelationen. Die Funktionalen Abhängigkeiten dieser Relation müssen bekannt sein.

Festgestellte Abweichungen

- *Nicht ableitbare Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten der Basisrelation, die nicht mehr aus den Funktionalen Abhängigkeiten der Teilrelationen der Lösung des Studenten ableitbar sind.

3.1.7.1 Realisierung

- *Analysator*: NormalizationAnalyzer
- *Aktivierende Methode*: analyzeDependenciesPreservation
- *Analyse*: DependenciesPreservationAnalysis

Repräsentation der benötigten Information

Die Methode `analyzeDependenciesPreservation` erwartet die für den Analyse Vorgang benötigte Information in den beiden folgenden Parametern:

- `Relation baseRelation`: Die der Zerlegung zugrundeliegende Relation. In dieser Relation müssen nur die Funktionalen Abhängigkeiten gesetzt sein. Die Attribute und Schlüssel werden nicht benötigt.
- `Collection decomposedRelations`: Die vom Studenten gebildeten Teilrelationen. In diesen Teilrelationen müssen ebenfalls nur die Funktionalen Abhängigkeiten gesetzt sein. Die Attribute und Schlüssel werden nicht benötigt.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `DependenciesPreservationAnalysis` repräsentiert. Diese Klasse stellt eine Liste `lostDependencies` zur Verfügung, in der nicht ableitbare Funktionale Abhängigkeiten vermerkt werden können.

Beschreibung des Analyse Vorgangs

Um die aus den Teilrelationen der Lösung des Studenten nicht ableitbaren Funktionalen Abhängigkeiten zu identifizieren, werden folgende Schritte durchgeführt:

1. Die Teilrelationen werden iteriert und die jeweiligen Funktionalen Abhängigkeiten werden temporär in einer Liste abgelegt.
2. Die Funktionalen Abhängigkeiten der Basisrelation werden iteriert. Für jede dieser Funktionalen Abhängigkeiten wird überprüft, ob sie aus den zuvor berechneten Funktionalen Abhängigkeiten ableitbar ist. Zu diesem Zweck wird der Algorithmus `Member` verwendet. Diesem Algorithmus wird einerseits die momentane Funktionale Abhängigkeit der Basisrelation und die Funktionalen Abhängigkeiten der Teilrelationen übergeben. Liefert der Algorithmus `Member` false bedeutet das, dass die momentane Funktionale Abhängigkeit der Basisrelation nicht ableitbar ist. In diesem Fall wird die jeweilige Funktionale Abhängigkeit der Basisrelation in der Analyse der Abhängigkeitstreue einer Zerlegung als nicht ableitbare Funktionale Abhängigkeit vermerkt und es wird festgehalten, dass die Lösung des Studenten nicht der berechneten Lösung entspricht. Liefert der Algorithmus `Member` true, bedeutet das, dass die momentane Funktionale Abhängigkeit der Basisrelation ableitbar ist und die Funktionalen Abhängigkeiten der Basisrelation werden weiter iteriert.

3.1.8 Gesamtanalyse - Minimale Überdeckung

Bei diesem Analysevorgang wird die Gesamtanalyse der Abgabe eines Beispiels vom Typ Minimale Überdeckung erstellt.

Benötigte Information

- *Basis Menge an Funktionale Abhängigkeiten*: Die Funktionalen Abhängigkeiten für die eine Minimale Überdeckung berechnet werden soll.
- *Funktionale Abhängigkeiten der Minimalen Überdeckung*: Die Funktionalen Abhängigkeiten, die der Student als Minimale Überdeckung der Basis Menge an Funktionalen Abhängigkeiten berechnet hat.

Festgestellte Abweichungen

- *Triviale Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten der Minimalen Überdeckung die trivial sind.
- *Nicht kanonisch dargestellte Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten der Minimalen Überdeckung die sich nicht in kanonischer Darstellung befinden.
- *Redundante Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten der Minimalen Überdeckung die redundant sind.
- *Überflüssige Attribute in Funktionale Abhängigkeiten*: Attribut in Funktionalen Abhängigkeiten der Minimalen Überdeckung die überflüssig sind.
- *Falsche Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten der Minimalen Überdeckung, die nicht aus der Basis Menge an Funktionalen Abhängigkeiten ableitbar sind. Würden falsche Funktionale Abhängigkeiten nicht festgestellt werden, könnte es sein, dass in den Funktionalen Abhängigkeiten der Minimalen Überdeckung zwar keine trivialen und redundanten Funktionale Abhängigkeiten enthalten sind, dass sich diese Funktionalen Abhängigkeiten weiters in kanonischer Darstellung befinden und sie keine überflüssigen Attribute aufweisen und es sich dennoch um keine Minimale Überdeckung der Basis Menge an Funktionalen Abhängigkeiten handelt.
- *Fehlende Funktionale Abhängigkeiten:* Funktionale Abhängigkeiten der Basis Menge, die aus den Funktionalen Abhängigkeiten der Minimalen Überdeckung nicht mehr ableitbar sind werden festgestellt. Würden fehlende Funktionale Abhängigkeiten nicht festgestellt werden, könnte es sein, dass in den Funktionalen Abhängigkeiten der Minimalen Überdeckung zwar keine trivialen und redundanten Funktionale Abhängigkeiten enthalten sind, dass sich diese Funktionale Abhängigkeiten weiters in kanonischer Darstellung befinden und sie keine überflüssigen Attribute aufweisen und es sich dennoch um keine Minimale Überdeckung der Funktionalen Abhängigkeiten der Basis Menge handelt.

3.1.8.1 Realisierung

- *Analysator*: MinimalCoverAnalyzer
- *Aktivierende Methode*: analyze
- *Analyse*: MinimalCoverAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwarte die für den Analyse Vorgang benötigte Information in den beiden folgenden Parametern:

- `Relation specification`: Die Funktionalen Abhängigkeiten dieser Relation repräsentieren die Basis Menge an Funktionalen Abhängigkeiten. Die Attribute und Schlüssel dieser Relation müssen nicht gesetzt sein.
- `Relation relation`: Die Funktionalen Abhängigkeiten dieser Relation stellen die Funktionalen Abhängigkeiten der Minimalen Überdeckung der Lösung des Studenten dar. Die Attribute und Schlüssel dieser Relation müssen nicht gesetzt sein.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `MinimalCoverAnalysis` repräsentiert. Abbildung 10 veranschaulicht den Aufbau dieser Klasse.

Funktionale Abhängigkeiten der Minimalen Überdeckung die trivial sind werden in einer Analyse Trivialer Funktionaler Abhängigkeiten repräsentiert (siehe `TrivialDependenciesAnalysis` in Abbildung 10). Funktionale Abhängigkeiten der Minimalen Überdeckung die sich nicht in kanonischer Darstellung befinden werden in einer Analyse der Kanonischen Darstellung von Funktionale Abhängigkeiten repräsentiert (siehe `CanonicalRepresentationAnalysis` in Abbildung 10).

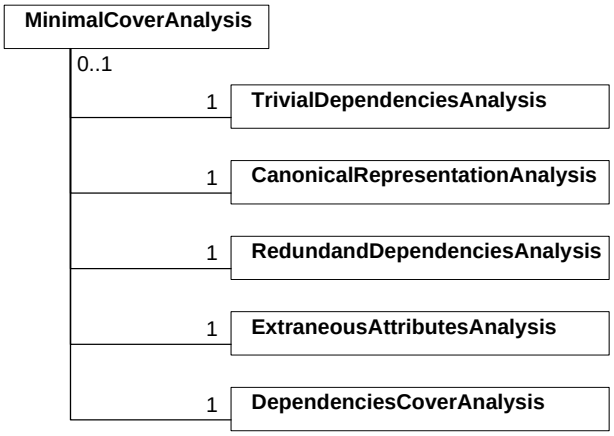

Abbildung 10: Aufbau der Klasse MinimalCoverAnalysis.

Funktionale Abhängigkeiten der Minimalen Überdeckung die redundant sind werden in einer Analyse Redundanter Funktionaler Abhängigkeiten repräsentiert (siehe `RedundandDependenciesAnalysis` in Abbildung 10). Attribut in Funktionalen Abhängigkeiten der Minimalen Überdeckung die überflüssig sind werden in einer Analyse Überflüssiger Attribute repräsentiert (siehe `ExtraneousAttributesAnalysis` in Abbildung 10). Fehlende bzw. Falsche Funktionale Abhängigkeiten der Minimalen Überdeckung werden in einer Analyse der Hülle Funktionaler Abhängigkeiten repräsentiert (siehe `ExtraneousAttributesAnalysis` in Abbildung 10).

Beschreibung des Analyse Vorgangs

Um festzustellen, ob sich die Funktionalen Abhängigkeiten der Lösung des Studenten in kanonischer Darstellung befinden wird der Analysator der Kanonischen Darstellung von Funktionalen Abhängigkeiten aktiviert (siehe Abschnitt 3.1.12). Die resultierenden Analysen der Kanonischen Darstellung von Funktionalen Abhängigkeiten werden der Analyse der Minimalen Überdeckung hinzugefügt. Wird erkannt, dass sich eine Funktionale Abhängigkeit der Lösung des Studenten nicht in kanonischer Darstellung befindet, wird in der Analyse der Minimalen Überdeckung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

Um festzustellen, ob sich in den Funktionalen Abhängigkeiten der Lösung des Studenten triviale Funktionale Abhängigkeiten befinden, wird der Analysator für Triviale Funktionale Abhängigkeiten aktiviert (siehe Abschnitt 3.1.11). Die resultierenden Analysen Trivialer Funktionaler Abhängigkeiten werden der Analyse der Minimalen Überdeckung hinzugefügt. Wird eine triviale Funktionale Abhängigkeit in der Lösung des Studenten erkannt, wird in der Analyse der Minimalen Überdeckung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

Um festzustellen, ob sich in den Funktionalen Abhängigkeiten der Lösung des Studenten redundante Funktionale Abhängigkeiten befinden, wird der Analysator für Redundante Funktionale Abhängigkeiten aktiviert (siehe Abschnitt 3.1.10). Die resultierenden Analysen Redundanter Funktionaler Abhängigkeiten werden der Analyse der Minimalen Überdeckung hinzugefügt. Wird eine redundante Funktionale Abhängigkeit erkannt, wird in der Analyse der Minimalen Überdeckung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

Um festzustellen, ob sich in den Funktionalen Abhängigkeiten der Lösung des Studenten solche befinden, die überflüssige Attribute aufweisen, wird der Analysator für Überflüssige Attribute in Funktionalen Abhängigkeiten aktiviert (siehe Abschnitt 3.1.9). Die resultierenden Analysen Überflüssiger Attribute in Funktionalen Abhängigkeiten werden der Analyse der Minimalen Überdeckung hinzugefügt. Wird eine Funktionale Abhängigkeit erkannt in der zumindest ein überflüssiges Attribut enthalten ist, wird in der Analyse der Minimalen Überdeckung festgehalten, dass die Lösung des Studenten nicht korrekt ist und der Analysevorgang wird abgebrochen.

Für das Identifizieren von fehlenden bzw. falschen Funktionalen Abhängigkeiten in der Lösung des Studenten wird die Hülle dieser Funktionalen Abhängigkeiten mit der Hülle der Funktionalen Abhängigkeiten der Basis Menge verglichen. Um diesen Vergleich durchzuführen, wird der Analysator der Hülle Funktionaler Abhängigkeiten aktiviert (siehe 3.1.13). Die resultierende Analyse der Hülle Funktionaler Abhängigkeiten wird in die Analyse der Minimalen Überdeckung aufgenommen. Wird eine fehlende bzw. falsche Funktionale Abhängigkeit erkannt, wird in der Analyse der Minimalen Überdeckung festgehalten, dass die Lösung des Studenten nicht korrekt ist.

3.1.9 Teilanalyse - Überflüssige Attribute

Bei diesem Analysevorgang werden überflüssige Attribute in Funktionalen Abhängigkeiten identifiziert.

Benötigte Information

- *Menge Funktionaler Abhängigkeiten*: Die Funktionalen Abhängigkeiten für die überprüft werden soll, ob und welche überflüssigen Attribute sie enthalten.

Festgestellte Abweichungen

- *Überflüssige Attribute*: Attribute die in einer Funktionalen Abhängigkeit überflüssig sind.

3.1.9.1 Realisierung

- *Analysator*: MinimalCoverAnalyzer
- *Aktivierende Methode*: analyzeExtraneousAttributes
- *Analyse*: ExtraneousAttributesAnalysis

Repräsentation der benötigten Information

Die Methode `analyzeExtraneousAttributes` erwartet die für den Analyse Vorgang benötigte Information im folgenden Input Parameter:

- `Collection dependencies`: Die Menge an Funktionalen Abhängigkeiten von denen jede auf überflüssige Attribute untersucht wird.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `ExtraneousAttributesAnalysis` repräsentiert. Diese Klasse stellt eine Map `extraneousAttributes` zur Verfügung, in der als Schlüssel Funktionale Abhängigkeiten vermerkt werden können und als Werte die in der jeweiligen Funktionalen Abhängigkeit enthaltenen überflüssigen Attribute.

Beschreibung des Analyse Vorgangs

Um die in einer Funktionalen Abhängigkeit enthaltenen überflüssigen Attribute zu identifizieren, werden die Funktionalen Abhängigkeiten iteriert. Für jede Funktionale Abhängigkeit werden die Attribute der linken Seite iteriert (überflüssige Attribute können nur in den linken Seiten von Funktionalen Abhängigkeiten enthalten sein). Für jedes Attribut der linken Seite einer Funktionaler Abhängigkeit werden folgende Schritte ausgeführt:

1. Es wird eine Kopie der Funktionalen Abhängigkeit angelegt. In dieser Kopie wird das momentan iterierte Attribut der linken Seite gelöscht.

2. Es wird eine Kopie der Funktionalen Abhängigkeiten der Lösung des Studenten angelegt, in der die zuvor erstellte Funktionale Abhängigkeit eingefügt und die ursprüngliche Funktionale Abhängigkeit gelöscht wird.

3. Es wird überprüft, ob die Hülle der Funktionalen Abhängigkeiten der Lösung des Studenten und die Hülle der zuvor erstellten und manipulierten Kopie dieser Funktionalen Abhängigkeiten gleich sind. Dieser Vergleich wird unter Zuhilfenahme des Algorithmus `Cover` durchgeführt. Sind diese beiden Hüllen gleich wird das momentan iterierte Attribut als überflüssig in der momentan iterierten Funktionalen Abhängigkeit in der Analyse Überflüssiger Attribute vermerkt. Weiters wird in der Analyse der Überflüssigen Attribute vermerkt, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird aber in jedem Fall fortgesetzt, um in einem Analysevorgang alle überflüssigen Attribute aller Funktionaler Abhängigkeiten zu identifizieren.

3.1.10 Teilanalyse - Redundante Funktionale Abhängigkeiten

Bei diesem Analysevorgang werden redundante Funktionale Abhängigkeiten in einer Menge Funktionaler Abhängigkeiten identifiziert.

Benötigte Information

- *Menge Funktionaler Abhängigkeiten*: Die Funktionalen Abhängigkeiten für die überprüft werden soll, ob sie redundante Funktionale Abhängigkeiten enthalten.

Festgestellte Abweichungen

- *Redundante Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die redundant sind.

3.1.10.1 Realisierung

- *Analysator*: MinimalCoverAnalyzer
- *Aktivierende Methode*: analyzeRedundandDependencies
- *Analyse*: RedundandDependenciesAnalysis

Repräsentation der benötigten Information

Die Methode `analyzeRedundandDependencies` erwartet die für den Analyse Vorgang benötigte Information in folgendem Input Parameter:

- `Collection dependencies`: Eine Menge an Funktionalen Abhängigkeiten in der nach redundanten Funktionalen Abhängigkeiten gesucht wird.

Repräsentation der festgestellten Abweichungen

Die aus diesem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `RedundandDependenciesAnalysis` repräsentiert. Diese Klasse stellt eine Liste `redundandDependencies` zur Verfügung, in der redundante Funktionale Abhängigkeiten vermerkt werden können.

Beschreibung des Analyse Vorgangs

Um die redundanten Funktionalen Abhängigkeiten zu identifizieren, werden die Funktionalen Abhängigkeiten der Lösung des Studenten iteriert. Für jede Funktionale Abhängigkeit werden folgende Schritte ausgeführt:

1. Es wird eine Kopie der Funktionalen Abhängigkeiten der Lösung des Studenten angelegt, in der die momentan iterierte Funktionale Abhängigkeit gelöscht wird.

2. Es wird überprüft, ob die Hülle der Funktionalen Abhängigkeiten der Lösung des Studenten und die Hülle der zuvor erstellten und manipulierten Kopie dieser Funktionalen Abhängigkeiten gleich sind. Dieser Vergleich wird unter Zuhilfenahme des Algorithmus `Cover` durchgeführt. Sind diese beiden Hüllen gleich wird die momentan iterierte Funktionale Abhängigkeit als redundante Funktionale Abhängigkeit in der Analyse Redundanter Funktionaler Abhängigkeiten vermerkt. Weiters wird in der Analyse Redundanter Funktionaler Abhängigkeiten vermerkt, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird aber in jedem Fall fortgesetzt, um in einem Analysevorgang alle redundanten Funktionalen Abhängigkeiten der Lösung des Studenten zu identifizieren.

3.1.11 Teilanalyse - Triviale Funktionale Abhängigkeiten

Bei diesem Analysevorgang werden triviale Funktionale Abhängigkeiten in einer Menge Funktionaler Abhängigkeiten identifiziert.

Benötigte Information

- *Menge Funktionaler Abhängigkeiten*: Die Funktionalen Abhängigkeiten für die überprüft werden soll, ob sie triviale Funktionale Abhängigkeiten enthalten.

Festgestellte Abweichungen

- *Triviale Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die trivial sind.

3.1.11.1 Realisierung

- *Analysator*: MinimalCoverAnalyzer
- *Aktivierende Methode*: analyzeTrivialDependencies
- *Analyse*: TrivialDependenciesAnalysis

Repräsentation der benötigten Information

Die Methode `analyzeTrivialDependencies` erwartet die für den Analyse Vorgang benötigte Information in folgendem Input Parameter:

- `Collection dependencies`: Die Menge an Funktionalen Abhängigkeiten in der nach trivialen Funktionalen Abhängigkeiten gesucht wird.

Repräsentation der festgestellten Abweichungen

Die aus diesem Analysevorgang resultierende Analyse wird in einem Objekt der Klasse `TrivialDependenciesAnalysis` repräsentiert. Diese Klasse stellt eine Liste `trivialDependencies` zur Verfügung, in der triviale Funktionale Abhängigkeiten vermerkt werden können.

Beschreibung des Analyse Vorgangs

Um die trivialen Funktionalen Abhängigkeiten zu identifizieren, werden die Funktionalen Abhängigkeiten der Lösung des Studenten iteriert. Für jede Funktionale Abhängigkeit wird überprüft, ob alle Attribute der rechten Seite auch in den Attributen der linken Seite der Funktionalen Abhängigkeit enthalten sind. Trifft dies zu, so wird die momentan iterierte Funktionale Abhängigkeit in der Analyse der Trivialen Funktionalen Abhängigkeiten vermerkt und festgehalten, dass die Lösung des Studenten nicht korrekt ist. Der Analyse Vorgang wird nicht abgebrochen, um in einem Analysevorgang alle trivialen Funktionalen Abhängigkeiten zu identifizieren.

3.1.12 Teilanalyse - Kanonische Darstellung

Bei diesem Analysevorgang werden Funktionale Abhängigkeiten in einer Menge Funktionaler Abhängigkeiten identifiziert, die sich nicht in kanonischer Darstellung befinden.

Benötigte Information

- *Menge Funktionaler Abhängigkeiten*: Die Funktionalen Abhängigkeiten für die überprüft werden soll, ob sie sich in kanonischer Darstellung befinden.

Festgestellte Abweichungen

- *Nicht kanonisch dargestellte Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten die sich nicht in kanonischer Darstellung befinden werden festgestellt.

3.1.12.1 Realisierung

- *Analysator*: MinimalCoverAnalyzer
- *Aktivierende Methode*: analyzeCanonicalRepresentation
- *Analyse*: CanonicalRepresentationAnalysis

Repräsentation der benötigten Information

Die Methode `analyzeCanonicalRepresentation` erwartet die für den Analysevorgang benötigte Information in folgendem Input Parameter:

- `Collection dependencies`: Die Menge an Funktionalen Abhängigkeiten für die überprüft wird, ob sie sich in kanonischer Darstellung befinden.

Repräsentation der festgestellten Abweichungen

Die aus diesem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `CanonicalRepresentationAnalysis` repräsentiert. Diese Klasse stellt eine Liste `notCanonicalDependencies` zur Verfügung gestellt, in der Funktionale Abhängigkeiten vermerkt werden können, die sich nicht in Kanonischer Darstellung befinden

Beschreibung des Analyse Vorgangs

Um die Funktionalen Abhängigkeiten zu identifizieren die sich nicht in kanonischer Darstellung befinden, werden die Funktionalen Abhängigkeiten der Lösung des Studenten iteriert. Für jede Funktionale Abhängigkeit wird überprüft, ob mehr als ein Attribut in der rechten Seite der momentan iterierten Funktionalen Abhängigkeit enthalten ist. Trifft dies zu, so wird die momentan iterierte Funktionale Abhängigkeit in der Analyse der Kanonischen Darstellung von Funktionalen Abhängigkeiten vermerkt und festgehalten, dass die Lösung des Studenten nicht korrekt ist. Der Analyse Vorgang wird in jedem Fall fortgestetzt, um alle Funktionalen Abhängigkeiten, die sich nicht in kanonischer Darstellung befinden in einem Vorgang zu identifizieren.

3.1.13 Teilanalyse - Hülle Funktionaler Abhängigkeiten

Bei diesem Analyse Vorgang werden die Hüllen zweier Mengen Funktionaler Abhängigkeiten verglichen.

Benötigte Information

- *Basis Menge an Funktionalen Abhängigkeiten*: Die Hülle dieser Funktionalen Abhängigkeiten wird zum Zweck der Überprüfung als Referenz genommen.
- *Zu prüfende Funktionale Abhängigkeiten*: Die Hülle dieser Funktionalen Abhängigkeiten wird mit der Hülle der Funktionalen Abhängigkeiten der Basis Menge verglichen.

Festgestellte Abweichungen

- *Fehlende Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten, die in den Funktionalen Abhängigkeiten der Basis Menge enthalten sind aber aus den zu prüfenden Funktionalen Abhängigkeiten nicht ableitbar sind.
- *Falsche Funktionale Abhängigkeiten*: Funktionale Abhängigkeiten, die in den zu prüfenden Funktionalen Abhängigkeiten enthalten sind aber aus den Funktionalen Abhängigkeiten der Basis Menge nicht ableitbar sind.

3.1.13.1 Realisierung

- *Analysator*: MinimalCoverAnalyzer
- *Aktivierende Methode*: analyzeDependenciesCover
- *Analyse*: DependenciesCoverAnalysis

Repräsentation der benötigten Information

Die Methode `analyzeDependenciesCover` erwartet die für den Analyse Vorgang benötigte Information in den beiden folgenden Input Parametern:

- `Collection correctDdependencies`: Diese Menge an Funktionalen Abhängigkeiten repräsentiert die Funktionalen Abhängigkeiten der Basis Menge.
- `Collection submittedDependencies`: Diese Menge an Funktionalen Abhängigkeiten repräsentiert die zu prüfenden Funktionalen Abhängigkeiten.

Repräsentation der festgestellten Abweichungen

Die aus dem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `DependenciesCoverAnalysis` repräsentiert. Diese Klasse stellt zwei Listen `additionalDependencies` und `missingDependencies` zur Verfügung in denen falsche bzw. fehlende Funktionale Abhängigkeiten vermerkt werden können.

Beschreibung des Analyse Vorgangs

Um die fehlenden Funktionalen Abhängigkeiten zu berechnen werden die Funktionalen Abhängigkeiten der Basis Menge iteriert. In jedem Iterationsschritt wird überprüft, ob die momentan iterierte Funktionale Abhängigkeit aus den zu prüfenden Funktionalen Abhängigkeiten ableitbar ist. Für diese Überprüfung wird der Algorithmus `Member` verwendet. Liefert der Algorithmus `Member` false, so wird die momentan iterierte Funktionale Abhängigkeit als fehlende Funktionale Abhängigkeit in der Analyse der Hülle Funktionaler Abhängigkeiten vermerkt und festgehalten, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird in jedem Fall fortgesetzt, um in einem Vorgang alle fehlenden Funktionalen Abhängigkeiten zu identifizieren.

Um die falschen Funktionalen Abhängigkeiten zu identifizieren werden die Funktionalen Abhängigkeiten der zu prüfenden Funktionalen Abhängigkeiten iteriert. In jedem Iterationsschritt wird überprüft, ob die momentan iterierte Funktionale Abhängigkeit aus den Funktionalen Abhängigkeiten der Basis Menge ableitbar ist. Für diese Überprüfung wird wiederum der Algorithmus `Member` verwendet. Liefert der Algorithmus `Member` false, so wird die momentan iterierte Funktionale Abhängigkeit als falsche Funktionale Abhängigkeit in der Analyse der Hülle

Funktionaler Abhängigkeiten vermerkt und festgehalten, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird in jedem Fall fortgesetzt, um in einem Vorgang alle falschen Funktionalen Abhängigkeiten zu identifizieren.

3.1.14 Teilanalyse - Bestimmen der Normalform einer Relation

Bei diesem Analysevorgang wird festgestellt ob die von einem Studenten berechnete Normalform einer Relation mit der korrekten Normalform dieser Relation übereinstimmt.

Benötigte Information

- *Funktionale Abhängigkeiten der Relation*: Die Funktionalen Abhängigkeiten der Relation, deren Normalform überprüft werden soll.
- *Schlüssel der Relation*: Die Schlüssel der Relation, deren Normalform überprüft werden soll.
- *Teilschlüssel der Relation*: Die Teilschlüssel der Relation, deren Normalform überprüft werden soll.
- *Angegebene Normalform der Relation*: Die vom Studenten angegebenen Normalform, in der sich die Relation befindet. Diese Angabe wird überprüft.

Festgestellte Abweichungen

- *Verletzungen der 2.Normalform*: Funktionale Abhängigkeiten, die eine Verletzung der 2. Normalform darstellen.
- *Verletzungen der 3 .Normalform*: Funktionale Abhängigkeiten, die eine Verletzung der 3. Normalform darstellen.
- *Verletzungen der BCNF*: Funktionale Abhängigkeiten, die eine Verletzung der BCNF darstellen.
- *Richtigkeit der angegebenen Normalform*: Es wird festgestellt, ob die vom Studenten angegebene Normalform mit der berechneten Normalform übereinstimmt.

Anmerkung: Verletzungen der 1. Normalform werden nicht analysiert, da diese nicht auf Basis der Funktionalen Abhängigkeiten festgestellt werden können und in den Übungsbeispielen auch davon ausgegangen wird, dass die 1. Normalform immer erfüllt ist.

3.1.14.1 Realisierung

- *Analysator*: NormalformAnalyzer
- *Aktivierende Methode*: analyze
- *Analyse*: NormalformAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwartet die für den Analyse Vorgang benötigte Information in einem Objekt der Klasse `NormalformAnalyzerConfig`. Abbildung 11 veranschaulicht den Aufbau dieser Klasse.

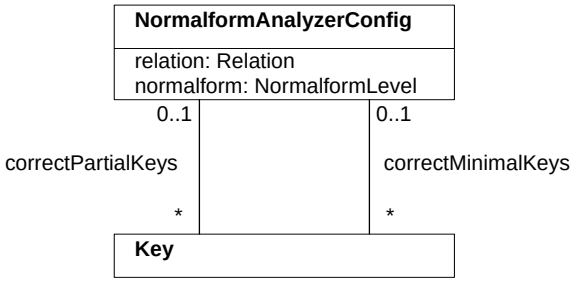

Abbildung 11: Aufbau der Klasse NormalformAnalyzerConfig.

Im Property `relation` werden die Funktionalen Abhängigkeiten der Relation repräsentiert, deren Normalform überprüft werden soll. Im Property `normalform` wird die vom Studenten angegebene Normalform dargestellt. In den Properties `correctPartialKeys` und `correctMinimalKeys` werden die Teilschlüssel bzw. Schlüssel der Relation deren Normalform überprüft werden soll repräsentiert.

Repräsentation der festgestellten Abweichungen

Die aus diesem Analyse Vorgang resultierende Analyse wird in einem Objekt der Klasse `NormalformAnalysis` repräsentiert. Diese Klasse stellt vier Listen (`firstNFViolations`, `secondNFViolation`, `thirdNFViolation`, `boyceCottNFViolation`) zur Verfügung in denen Funktionale Abhängigkeiten vermerkt werden können, die eine Verletzung der jeweiligen Normalform darstellen. Dabei wird nur die niedrigste verletzte Normalform vermerkt und nicht jede durch die jeweilige Funktionale Abhängigkeit verletzte Normalform.

Beschreibung des Analyse Vorgangs

Um festzustellen ob eine Funktionale Abhängigkeit eine Verletzung einer bestimmten Normalform darstellt, werden die Funktionalen Abhängigkeiten der Relation deren Normalform überprüft werden soll iteriert. Stellt eine Funktionale Abhängigkeit eine Verletzung einer Normalform dar, wird diese in der Analyse der Normalform einer Relation hinzugefügt. Die Überprüfung einer Funktionalen Abhängigkeit auf Verletzung einer Normalform wird folgendermaßen durchgeführt:

- Verletzung der 2. Normalform: Falls die rechte Seite der Funktionalen Abhängigkeit ein nicht primes Attribut enthält und die linke Seite ein Teilschlüssel ist, ist durch diese Funktionale Abhängigkeit eine Verletzung der 2. Normalform gegeben. Die betroffene Funktionale Abhängigkeit wird in der Analyse der Normalform einer Relation hinzugefügt und der Analysevorgang mit der nächsten Funktionalen Abhängigkeit fortgesetzt.

- Verletzung der 3. Normalform: Falls die rechte Seite der Funktionalen Abhängigkeit ein nicht primes Attribut enthält und die linke Seite kein Oberschlüssel ist, ist durch diese Funktionale Abhängigkeit eine Verletzung der 3. Normalform gegeben. Die betroffene Funktionale Abhängigkeit wird in der Analyse der Normalform einer Relation hinzugefügt und der Analysevorgang mit der nächsten Funktionalen Abhängigkeit fortgesetzt.

- Verletzung der BCNF: Falls die linke Seite der Funktionalen Abhängigkeit kein Schlüssel ist, ist durch diese Funktionale Abhängigkeit eine Verletzung der BCNF gegeben. Die betroffene Funktionale Abhängigkeit wird in der Analyse der Normalform einer Relation hinzugefügt und der Analysevorgang mit der nächsten Funktionalen Abhängigkeit fortgesetzt.

Anschließend wird die korrekte Normalform in der sich die Relation befindet bestimmt. Dies geschieht dadurch, dass für jede Normalform, beginnend mit der BCNF, überprüft wird, ob eine Verletzung gefunden wurde. Falls ja, wird die Normalform der Relation auf die nächst niedrigere Normalform gesetzt. Entspricht die berechnete Normalform der Relation nicht der Normalform der Lösung des Studenten wird in der Analyse der Normalform einer Relation festgehalten, dass die Lösung des Studenten nicht korrekt ist.

3.1.15 Gesamtanalyse - Attribut Hülle

Bei diesem Analysevorgang wird die Gesamtanalyse einer Abgabe eines Beispiel vom Typ Attribut Hülle erstellt.

Benötigte Information

- *Menge an Funktionalen Abhängigkeiten*: Die Funktionalen Abhängigkeiten, die der Berechnung der Attribut Hülle zugrunde liegen.
- *Basis Attribute*: Die Attribute, deren Hülle berechnet wird.
- *Vom Studenten berechnete Hülle*: Die Attribute, die vom Studenten als Hülle der Basis Attribute berechnet wurden.

Festgestellte Abweichungen

- *Fehlende Attribute*: Attribute, die in der korrekten Hülle vorhanden sind, nicht aber in der Hülle der Lösung des Studenten.
- *Falsche Attribute*: Attribute, die in der Hülle der Lösung des Studenten enthalten sind, nicht aber in der korrekten Hülle.

3.1.15.1 Realisierung

- *Analysator*: AttributeClosureAnalyzer
- *Aktivierende Methode*: analyze
- *Analyse*: AttributeClosureAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwartet die für den Analysevorgang benötigte Information in den folgenden Input Parametern:

- `Collection dependencies`: Die darin enthaltenen Funktionalen Abhängigkeiten repräsentieren jene Funktionalen Abhängigkeiten, die der Berechnung der Attribut Hülle zugrunde liegen.
- `Collection baseAttributes`: Die darin enthaltenen Attribute repräsentieren jene Attribute, deren Hülle berechnet wird.
- Collection submittedAttributes: Die darin enthaltenen Attribute repräsentieren jene Attribute, die vom Studenten als Hülle der Basis Attribute berechnet wurden.

Repräsentation der festgestellten Abweichungen

Die aus diesem Analysevorgang resultierende Analyse wird in einem Objekt der Klasse `AttributeClosureAnalysis` repräsentiert. Diese Klasse stellt die beiden Listen `additionalAttributes` und `missingAttributes` zur Verfügung in denen falsche bzw. fehlende Attribute vermerkt werden können.

Beschreibung des Analyse Vorgangs

Die Berechung der korrekten Hülle der Basis Attribute wird unter Verwendung des Algorithmus `AttributeClosure` durchgeführt.

Um fehlende Attribute in der Hülle der Lösung des Studenten zu eruieren, werden die Attribute der korrekten Hülle iteriert. Für jedes dieser Attribute wird überprüft, ob es auch in der Hülle der Lösung des Studenten vorhanden ist. Ist ein Attribut nicht enthalten, wird es als fehlendes Attribut in der Analyse der Attribut Hülle vermerkt. Weiters wird festgehalten, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird in jedem Fall fortgesetzt, um in nur einem Vorgang alle fehlenden Attribute zu identifizieren.

Um falsche Attribute in der Hülle der Lösung des Studenten zu eruieren, werden diese Attribute iteriert. Für jedes dieser Attribute wird überprüft, ob es auch in der korrekten Hülle vorhanden ist. Ist ein Attribut nicht darin enthalten, wird es als falsches Attribut in der Analyse der Attribut Hülle vermerkt. Weiters wird festgehalten, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird in jedem Fall fortgesetzt, um in nur einem Vorgang alle falschen Attribute zu identifizieren.

3.1.16 Gesamtanalyse - Bestimmen von Schlüsseln

Bei diesem Analysevorgang wird die Gesamtanalyse der Abgabe eines Beispiel vom Typ Bestimmen von Schlüsseln einer Relation erstellt.

Benötigte Information

- *Attribute der Relation*: Die Attribute der Relation deren Schlüssel bestimmt werden.
- *Funktionale Abhängigkeiten der Relation*: Die Funktionalen Abhängigkeiten der Relation deren Schlüssel bestimmt werden.
- *Vom Studenten gefundene Schlüssel*: Die vom Studenten berechneten Schlüssel der Relation.

Festgestellte Abweichungen

- *Fehlende Schlüssel*: Schlüssel die in der korrekten Lösung enthalten sind, nicht aber in der Lösung des Studenten.
- *Falsche Schlüssel*: Schlüssel die in der Lösung des Studenten, nicht aber in der korrekten Lösung, enthalten sind.

3.1.16.1 Realisierung

- *Analysator*: KeysAnalyzer
- *Aktivierende Methode*: analyze
- *Analyse*: KeysAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwartet die für den Analysevorgang benötigt Information in den beiden folgenden Input Parametern:

Repräsentation der festgestellten Abweichungen

- `Relation relation`: Die Attribute und Funktionalen Abhängigkeiten dieser Relation repräsentieren die Attribute und Funktionalen Abhängigkeiten der Relation, deren Schlüssel überprüft werden sollen.
- `KeysAnalyzerConfig config`: In diesem Parameter sind die korrekten Schlüssel enthalten.

Beschreibung des Analyse Vorgangs

Da das Berechnen der Schlüssel sehr rechenintensiv ist, werden die korrekten Schlüssel einer Relation nicht vom Analysator der Schlüssel einer Relation berechnet sondern bereits vor dem Aufruf der Methode `analyze`. Diese Vorgehensweise hat den Vorteil, dass sofern die Schlüssel der jeweiligen Relation bereits im Rahmen eines anderen Analysevorgangs berechnet wurden, diese nicht nochmals berechnet werden müssen. Das Berechnen der Schlüssel wird von der Klasse `KeysDeterminator` durchgeführt. Im Folgenden wird die Funktionsweise dieser Klasse kurz beschrieben.

1. *Berechnung der Oberschlüssel*: Bei der Berechnung der Oberschlüssel werden in einem ersten Schritt alle Attribute der Relation gesucht, die in keiner Funktionalen Abhängigkeit auf der rechten Seite vorkommen. Diese Attribute müssen Teil jedes Schlüssels und somit auch jedes Oberschlüssels sein. In einem zweiten Schritt werden aus den restlichen Attributen alle möglichen Attributkombinationen berechnet. Zu jeder dieser Attributkombinationen werden im letzten Schritt die Attribute, die in keiner Funktionalen Abhängigkeit auf der rechten Seite vorkommen, zu jeder Attributkombination hinzugefügt. Die resultierenden Attributkombinationen stellen alle Oberschlüssel der Relation dar.

2. *Berechnung der Schlüssel*: Von jedem Oberschlüssel wird ein Attribut nach dem anderen entfernt und überprüft, ob die Hülle dieser Attribute noch alle Attribute der Relation enthält. Sobald die Hülle einer Attributkombination nicht mehr alle Attribute der Relation enthält, ist sichergestellt, dass die vorherige Attributkombination ein Schlüssel ist.

3. *Berechnung der Teilschlüssel*: Für jeden Schlüssel werden alle möglichen Kombinationen der Attribute des Schlüssels berechnet. Jede dieser Kombinationen stellt einen Teilschlüssel dar.

Um die fehlenden Schlüssel in der Lösung des Studenten zu identifizieren, werden die korrekten Schlüssel iteriert. Ist einer dieser Schlüssel nicht in der Lösung des Studenten enthalten, wird dieser als fehlender Schlüssel in der Analyse der Schlüssel einer Relation vermerkt. Weiters wird festgehalten, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird in jedem Fall fortgesetzt um in nur einem Vorgang alle fehlenden Schlüssel zu identifizieren.

Um die falschen Schlüssel in der Lösung des Studenten zu identifizieren, werden diese Schlüssel iteriert. Ist einer dieser Schlüssel nicht in der korrekten Lösung enthalten, wird der jeweilige Schlüssel als falscher Schlüssel in der Analyse der Schlüssel einer Relation vermerkt. Weiters wird festgehalten, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird in jedem Fall fortgesetzt um in nur einem Durchgang alle fehlenden Schlüssel zu identifizieren.

3.1.17 Gesamtanalyse - Bestimmen von Normalform Verletzungen

Bei diesem Analysevorgang wird die Gesamtanalyse der Abgabe eines Beispiels vom Typ Finden von Normalform Verletzungen erstellt. Unter einer Normalform Verletzung wird dabei eine Funktionale Abhängigkeit verstanden, die eine Verletzung einer bestimmten Normalform in einer Relation darstellt.

Benötigte Information

- *Funktionale Abhängigkeiten der Relation*: Die Funktionalen Abhängigkeiten für die überprüft wird, ob sie eine Verletzung einer Normalform darstellen.
- *Schlüssel der Relation*: Die Schlüssel der Relation für die Normalform Verletzungen festgestellt werden sollen. Diese Schlüssel werden benötigt, um Verletzungen der zweiten Normalform feststellen zu können.
- *Teilschlüssel der Relation*: Die Schlüssel der Relation für die Normalform Verletzungen festgestellt werden sollen. Diese Schlüssel werden benötigt, um Verletzungen der dritten Normalform feststellen zu können.
- *Normalform Verletzungen*: Die vom Studenten angegebenen Funktionalen Abhängigkeiten, die eine bestimmte Normalform verletzen.
- *Angegebene Normalform der Relation*: Die vom Studenten angegebenen Normalform, in der sich die Relation befindet.

Festgestellte Abweichungen

- *Korrektheit der angegebenen Normalformverletzungen*: Es wird festgestellt, ob eine Funktionale Abhängigkeit der Lösung des Studenten auch wirklich eine Verletzung der angegebenen Normalform darstellt. Weiters wird festgestellt, welche Normalform durch die jeweilige Funktionale Abhängigkeit tatsächlich verletzt wird, sofern überhaupt eine Verletzung einer Normalform besteht.
- *Korrektheit der angegebenen Normalform*: Es wird festgestellt, ob die vom Studenten angegebene Normalform der Relation mit der berechneten Normalform übereinstimmt.

3.1.17.1 Realisierung

- *Analysator*: NormalformDeterminationAnalyzer
- *Aktivierende Methode*: analyze

- *Analyse*: NormalformDeterminationAnalysis

Repräsentation der benötigten Information

Die Methode `analyze` erwartet die für den Analysevorgang benötigte Information in den beiden folgenden Input Parametern:

- `NormalformDeterminationSubmission submission`: In diesem Parameter sind die vom Studenten gefundenen, eine bestimme Normalform verletzenden Funktionalen Abhängigkeiten enthalten. Zu jeder Funktionalen Abhängigkeit ist die niedrigste verletzte Normalform vermerkt. Weiters ist in diesem Parameter, die vom Studenten berechnete Normalform der Relation enthalten.
- `NormalformAnalyzerConfig config`: In diesem Parameter sind die Funktionalen Abhängigkeiten, Schlüssel und Teilschlüssel der Relation für die die Normalform Verletzungen überprüft werden sollen enthalten (siehe Abschnitt 3.1.14).

Repräsentation der festgestellten Abweichungen

Die aus diesem Analysevorgang resultierende Analyse wird in einem Objekt der Klasse `NormalformDeterminationAnalysis` repräsentiert. Diese Klasse stellt eine Liste `wrongLeveledDependencies` zur Verfügung, in der Tripel bestehend aus einer Funktionalen Abhängigkeit, der vom Studenten angegebenen verletzten Normalform und der tatsächlich verletzten Normalform vermerkt werden können.

Beschreibung des Analyse Vorgangs

In einem ersten Schritt wird der Analysator der Normalform einer Relation aktiviert. In der resultierenden Analyse der Normalform einer Relation befindet sich eine Gegenüberstellung von Funktionalen Abhängigkeiten und der niedrigsten Normalform die die jeweilige Funktionale Abhängigkeit verletzt.

Anhand dieser Analyse wird für jede Normalform Verletzung der Lösung des Studenten überprüft, ob die jeweilige Funktionale Abhängigkeit auch tatsächlich eine Verletzung der angegebenen Normalform darstellt. Trifft dies nicht zu, wird das in der Analyse der Normalform Verletzungen vermerkt. Weiters wird vermerkt, dass die Lösung des Studenten nicht korrekt ist. Der Analysevorgang wird in jedem Fall fortgesetzt, um alle falschen Normalform Verletzungen der Lösung des Studenten in einem Durchlauf zu identifizieren.

Abschließend wird überprüft, ob die angegebene Normalform der Lösung des Studenten auch mit der Normalform der Analyse der Normalform einer Relation übereinstimmt. Trifft dies nicht zu, wird in der Analyse der Normalform Verletzungen festgehalten, dass die Lösung des Studenten nicht korrekt ist.

3.2 Bewertung

Die Bewertung von Beispiel Abgaben ist im RDBD Modul in der derzeitigen Ausbaustufe nur rudimentär realisiert. Bei der Bewertung wird überprüft, ob in der zugrundeliegenden Analyse vermerkt ist, dass Abweichungen festgestellt wurden. Trifft dies zu, werden Null Punkte vom RDBD Modul vorgeschlagen. Trifft dies nicht zu, wird vom RDBD Modul einer von maximal

einem erreichbaren Punkt vorgeschlagen. Diese Vorgehensweise wird für alle unterstützen Beispieltypen gemeinsam und unabhängig von konkreten Beispielen verfolgt. Die Bewertungsfunktionalität ist in der Klasse `Grader` implementiert.

3.3 Berichterstellung

Im RDBD Modul wird die Erstellung von Berichten für die Evaluationsarten "check", "submit" und "diagnose" für die einzelnen Beispieltypen unterstützt. Berichte für die Evaluationsart "diagnose" können für drei Diagnosestufen erstellt werden. Tabelle 9 gibt einen Überblick über die grundsätzliche Intention der Meldungen für die einzelnen Evaluationsarten und Diagnosestufen.

Evaluationsart	*Meldung*	*Intention*
check	keine	Der Student erfährt lediglich ob die Beispiel Abgabe korrekt ist oder nicht. Darüber hinaus erhält er keine Rückmeldung.
submit	siehe diagnose low	siehe diagnose low
diagnose low	Die Art der Abweichung wird allgemein beschrieben.	Durch eine allgemeine Beschreibung der Abweichung kann der Student gezielter nach den Abweichungen suchen.
diagnose medium	Die Anzahl der Abweichungen der jeweiligen Art wird angegeben.	Der Student weiß wie viele Abweichungen von ihm zu beseitigen sind, ohne weitere Evaluationen durchführen zu müssen.
diagnose high	Die konkreten Abweichungen der jeweiligen Art werden aufgelistet.	Dem Studenten wird dadurch die Lösung mitgeteilt. Diese Vorgehensweise eignet sich nicht im Abgabe Modus aber sehr wohl im Übungs-Modus.

Tabelle 9: Übersicht über Evaluationsarten und Meldungen.

Der Erstellung von Berichten liegt im RDBD Modul die Überlegung zugrunde, dass die benötigte Funktionalität für das Erstellen eines Berichts, in Analogie zur benötigten Funktionalität für das Erstellen einer Analyse, nicht auf dem Typ eines Beispiels sondern auf den zu berichtenden Abweichungen der jeweiligen Analysen basiert. Sofern in einer Analyse eine bestimmte Abweichung in der Beispiel Abgabe von der berechneten Lösung vorliegt, wird dem Studenten eine bestimmte Rückmeldung gegeben unabhängig vom Typ des jeweiligen Beispiels.

Ein Beispiel zur Veranschaulichung: Sofern eine Ergebnisrelation einer Zerlegung in der Beispiel Abgabe nicht die gewünschte Normalform aufweist, ist dem Studenten eben genau diese Abweichung zu berichten und zwar unabhängig davon, ob der Student die Ergebnisrelation im Rahmen eines Beispiels vom Typ Decompose Algorithmus oder vom Typ Allgemeine Normalisierung erstellt hat.

Grundlegend wird diese Vorgehensweise dadurch ermöglicht, dass im RDBD Modul ein Modell für Berichte entwickelt wurde. Abbildung 12 veranschaulicht dieses Modell. Die kleinste Einheit stellt dabei eine Meldung (ErrorReport) dar. Eine Meldung dient dazu, eine bestimmte Abweichung aufzuzeigen (Property error), diese Abweichung zu beschreiben (Property description) und eventuell einen Hinweis zu geben, wo weiterführende Informationen gefunden werden können (Property hint). Da nicht jede Abweichung auch zwangsläufig einen Fehler darstellt, kann in Meldungen festgelegt werden, in welcher Art die Meldung angezeigt werden soll

(Property type). Meldungen können als Fehler, Warnung oder ledigliche Information angezeigt werden. Meldungen werden in einem Bericht zusammengefasst (Klasse Report). Ein Bericht stellt das gesamte an einen Studenten returnierte Feedback zu der Analyse einer Beispiel Abgabe dar. Ein Bericht hat immer einen Prolog (Property prolog). Im Prolog eines Berichts werden allgemeine Aussagen wie zum Beispiel "das Beispiel wurde korrekt gelöst" oder "das Beispiel wurde nicht korrekt gelöst" vermerkt. Sofern einzelne Meldungen einen besonders starken semantischen Zusammenhang aufweisen, können diese in einer Meldungsgruppe zusammengefasst werden (ErrorReportGroup). Eine Meldungsgruppe hat eine Bezeichnung (Property header) die den Zusammenhang der darin gruppierten Meldungen deutlich macht. Solche Meldungsgruppen können neben einzelnen Meldungen in einem Bericht enthalten sein. Meldungsgruppen können auch verschachtelt werden. Weiters stellen Berichte, Meldungen und Meldungsgruppen Konfigurations-Möglichkeiten zur Verfügung, die es erlauben, das Anzeigen einzelner Properties an und auszuschalten.

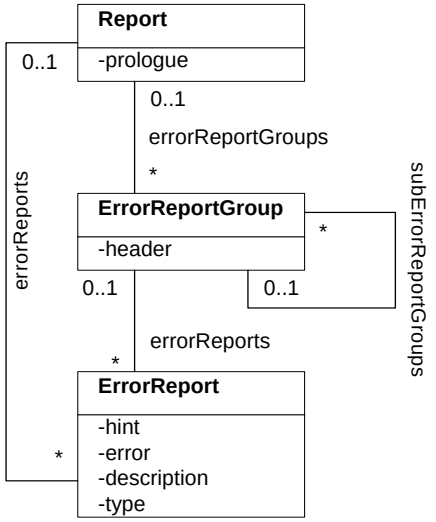

Abbildung 12: Modell eines Berichts.

Im RDBD Modul wird für jeden unterstützten Beispieltyp eine Komponente zur Verfügung gestellt, die die Erstellung eines entsprechenden Berichts ermöglicht. In der derzeitigen Ausbaustufe des RDBD Moduls werden Berichte ausschließlich in Englisch erstellt. Es ist derzeit auch noch kein Mechanismus realisiert, der es ermöglichen würde Berichte in anderen Sprachen zu erstellen. Diese Komponenten werden im Folgenden als Berichtersteller bezeichnet. Jeder Berichtersteller stellt eine Methode `Report report(Analysis analysis, Grading grading, ReporterConfig config)` zur Verfügung, durch deren Aufruf die Erzeugung des jeweiligen Berichts veranlasst wird. In der `ReporterConfig` wird festgehalten, für welche Diagnosestufe und für welche Evaluationsart ("check", "diagnose", "submit") der Bericht erstellt werden soll. Das vom eTutor Core vorgesehene Interface für das Erstellen eines Berichts wird folglich nicht von den einzelnen Berichterstellern implementiert sondern vom `RDBDEvaluator`. Der `RDBDEvaluator` ist dafür zuständig, die Inputs des eTutor Cores für das Erstellen eines Berichts so umzuwandeln, dass sie den Anforderungen der jeweiligen Berichtersteller entsprechen und diese dann zu aktivieren.

Dadurch, dass für Abweichungen der gleichen Art die gleichen Meldungen erstellt werden, muss nicht jeder Berichtersteller selbst alle Meldungen erstellen, sondern kann sich dafür anderer Berichtersteller bedienen. Darüber hinaus muss sich nicht jeder Berichtersteller selbst um die Darstellung des Berichts in der Benutzerschnittstelle der Beispiel Ausarbeitung kümmern. Statt dessen gibt es eine einzelne Komponente, die für die Darstellung eines Berichts zuständig ist. Die Darstellung eines Berichts in der Benutzerschnittstelle der Beispiel Ausarbeitung ist in der JSP Seite `printReport.jsp` implementiert.

In den folgenden Abschnitten werden die einzelnen Berichte und Meldungen des RDBD Moduls näher beschrieben. Der Fokus liegt dabei auf den Zusammenhängen zwischen den Berichten und dem Aufbau dieser Berichte. Beim Lesen der Beschreibungen der einzelnen Berichte und Meldungen sind die beiden folgenden Punkte zu berücksichtigen:

- *Der Prolog von Berichten*: In den folgenden Beschreibungen wird nicht explizit auf das Erstellen des Prologs eingegangen. Im Prolog eines Berichts ist immer vermerkt, ob die Lösung des Studenten der korrekten Lösung entspricht (oder eben nicht) und wie viele Punkte vorgeschlagen werden, sofern ein Bericht für die Evaluationsart "submit" erstellt wird.
- *Typen von Meldungen*: Sofern nicht explizit in den folgenden Beschreibungen der Meldungen angegeben wird, welchen Typ (Fehler, Warnung, Information) eine Meldung aufweist, werden Meldungen vom Typ Fehler erzeugt.
- *Erläuterung von Meldungen*: Soll ein Bericht im Rahmen der Evaluationsart "check" erstellt werden, wird lediglich der Prolog eines Berichts erstellt, aber keine Meldungen. Dem Studenten soll nur berichtet werden, ob seine Beispiel Abgabe richtig ist oder nicht. Soll ein Bericht im Rahmen der Evaluationsart "submit" erstellt werden, gleichen die Meldungen denen, die im Rahmen der Evaluationsart "diagnose" in der Diagnosestufe "low" erstellt werden. In den folgenden Beschreibungen wird nur mehr auf die Beschreibung der Meldungen in den einzelnen Diagnosestufen eingegangen und nicht mehr explizit erwähnt, wie sich die Meldungen in den Evaluationsarten "submit" und "check" gestalten.

3.3.1 Bericht - RBR Algorithmus

- Berichtersteller: RBRReporter
- Methode: report
- Analyse: RBRAnalysis

Dieser Bericht stellt das an einen Studenten returnierte Feedback für die Abgabe eines Beispiels vom Typ RBR Algorithmus dar. Abbildung 13 zeigt den Aufbau dieses Berichts

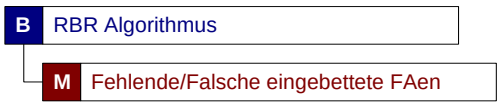

Abbildung 13: Bericht - RBR Algorithmus.

Wurden beim Analysieren der Beispiel Abgabe fehlende bzw. falsche eingebettete Funktionale Abhängigkeiten festgestellt, wird dem Bericht eine entsprechende Meldung hinzugefügt (siehe Abschnitt 3.3.21).

3.3.2 Bericht - Decompose Algorithmus

- *Berichtersteller*: DecomposeReporter
- *Methode*: report
- *Analyse*: DecomposeAnalysis

Dieser Bericht stellt das an einen Studenten returnierte Feedback für die Abgabe eines Beispiels vom Typ Decompose Algorithmus dar. Abbildung 14 zeigt den Aufbau dieses Berichts.

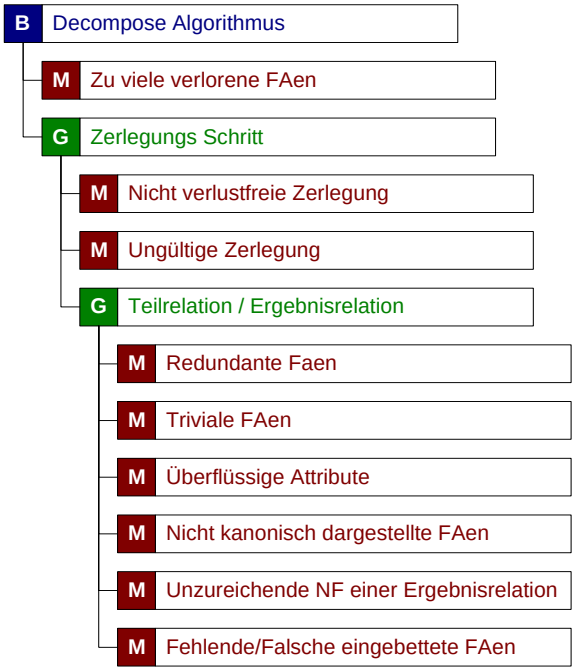

Abbildung 14: Bericht – Decompose Algorithmus.

Wurde beim Analysieren der Beispiel Abgabe festgestellt, dass bei der Zerlegung zu viele Funktionale Abhängigkeiten verloren wurden, wird dem Bericht eine entsprechende Meldung hinzugefügt (Abschnitt 3.3.19).

Wurden beim Analysieren der Beispiel Abgabe Abweichungen in einzelnen Zerlegungsschritten von der korrekten Lösung festgestellt, wird für jeden betroffenen Zerlegungsschritt eine Meldungsgruppe erstellt. Die Bezeichnung solcher Meldungsgruppen gibt an, für die Zerlegung welcher Relation darin Meldungen enthalten sind. Wurde festgestellt, dass es sich im jeweiligen Zerlegungsschritt um eine nicht verlustfreie bzw. ungültige Zerlegung handelt, werden entsprechende Meldungen hinzugefügt (siehe Abschnitt 3.3.20 und Abschnitt 3.3.18).

Wurden beim Analysieren der Beispiel Abgabe Abweichungen in einzelnen Teilrealtionen eines Zerlegungsschritts von der korrekten Lösung festgestellt, wird für jede dieser Teilrelationen innerhalb der Meldungsgruppe für den Zerlegungsschritt eine weitere Meldungsgruppe erstellt. Die Bezeichnung solcher Meldungsgruppen gibt an, für welche Teilrelation Meldungen enthalten sind.

Für jede Teilrelation unabhängig davon ob diese auch eine Ergebnisrelation ist oder nicht werden Meldungen für redundante, triviale, nicht kanonisch dargestellte Funktionale Abhängigkeiten bzw. Funktionale Abhängigkeiten, die überschüssige Attribute enthalten erstellt (siehe Abschnitt 3.3.13, Abschnitt 3.3.14, Abschnitt 3.3.10 und Abschnitt 3.3.12). Wurde beim Analysieren einer Teilrelation festgestellt, dass eingebettete Funktionale Abhängigkeiten fehlen bzw. falsch sind wird ebenfalls unabhängig davon, ob es sich um eine Ergebnisrelation handelt oder nicht eine entsprechende Meldung erstellt (siehe Abschnitt 3.3.21). Wurde beim Analysieren einer Teilrelation festgestellt, dass eine Teilrelation nicht die gewünschte Normalform auf, wird die entsprechende Meldung (siehe Abschnitt 3.3.17) als Fehler typisiert, sofern es sich bei der

Teilrelation um eine Ergebnisrelation handelt. Anderenfalls wird die entsprechende Meldung als lediglich Information typisiert. Der Grund dafür ist, dass eine unzureichende Normalform nur dann einen Fehler darstellt, wenn es sich um eine Ergebnisrelation handelt.

3.3.3 Bericht - Allgemeine Normalisierung

- *Berichtersteller*: NormalizationReporter
- *Methode*: report
- *Analyse*: NormalizationAnalysis

Dieser Bericht stellt das an einen Studenten returnierte Feedback für die Abgabe eines Beispiels vom Typ Allgemeine Normalisierung dar. Abbildung 15 zeigt den Aufbau dieses Berichts.

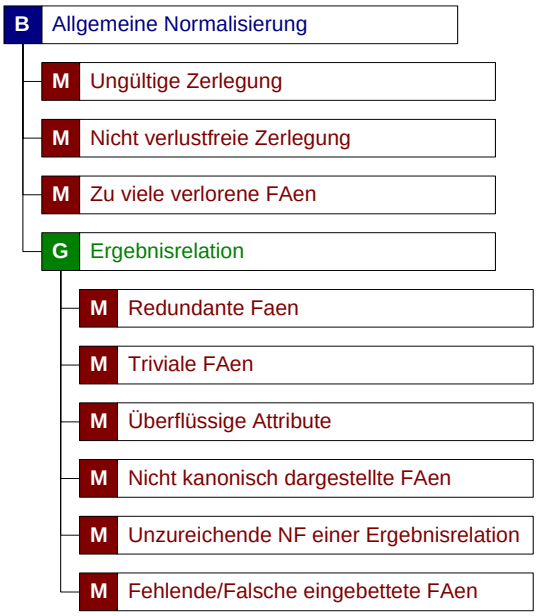

Abbildung 15: Bericht - Allgemeine Normalisierung.

Wurde beim Analysieren der Beispiel Abgabe festgestellt, dass bei der Zerlegung zu viele Funktionale Abhängigkeiten verloren wurden, wird dem Bericht eine entsprechende Meldung hinzugefügt (siehe Abschnitt 3.3.19). Wurde beim Analysieren der Beispiel Abgabe festgestellt, dass es sich um eine nicht verlustfreie bzw. ungültige Zerlegung handelt, werden dem Bericht entsprechende Meldungen hinzugefügt (siehe Abschnitt 3.3.20 und Abschnitt 3.3.18).

Wurden beim Analysieren der Beispiel Abgabe Abweichungen in den einzelnen Ergebnisrelationen von der korrekten Lösung festgestellt, wird für jede betroffenen Ergebnisrelation eine Meldungsgruppe erstellt. Die Bezeichnung solcher Meldungsgruppen gibt an, für welche Ergebnisrelation Meldungen enthalten sind.

Für jede Ergebnisrelation werden Meldungen für redundante, triviale, nicht kanonisch dargestellte Funktionale Abhängigkeiten bzw. Funktionale Abhängigkeiten, die überschüssige Attribute enthalten erstellt (siehe Abschnitt 3.3.13, Abschnitt 3.3.14, Abschnitt 3.3.10 und Abschnitt 3.3.12). Wurde beim Analysieren einer Ergebnisrelation festgestellt, dass eingebettete Funktionale

Abhängigkeiten fehlen bzw. falsch sind oder wurde festgestellt, dass die Ergebnisrelation nicht die gewünschte Normalform aufweist werden dem Bericht ebenfalls entsprechende Meldungen hinzugefügt (siehe Abschnitt 3.3.21 und Abschnitt 3.3.17).

3.3.4 Bericht - Minimale Überdeckung

- *Berichtersteller*: MinimalCoverReporter
- *Methode*: report
- *Analyse*: MinimalCoverAnalysis

Dieser Bericht stellt das an einen Studenten returnierte Feedback für die Abgabe eines Beispiels vom Typ Minimale Überdeckung dar. Abbildung 16 zeigt den Aufbau dieses Berichts.

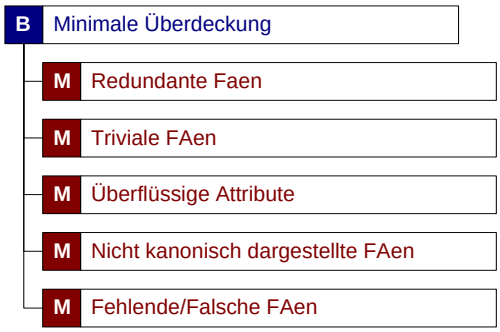

Abbildung 16: Bericht – Minimale Überdeckung.

Wurde beim Analysieren der Beispiel Abgabe festgestellt, dass in der Minimalen Überdeckung redundante, triviale, nicht kanonisch dargestellte Funktionale Abhängigkeiten bzw. Funktionale Abhängigkeiten mit überschüssige Attributen enthalten sind, werden entsprechende Meldungen dem Bericht hinzugefügt (siehe Abschnitt 3.3.13, Abschnitt 3.3.14, Abschnitt 3.3.10 und Abschnitt 3.3.12). Wurde beim Analysieren festgestellt, dass die Hülle der Funktionale Abhängigkeiten der Minimalen Überdeckung der Beispiel Abgabe nicht mit der Hülle der Funktionale Abhängigkeiten der korrekten Lösung übereinstimmt, wird eine Meldung der fehlenden bzw. falschen Funktionale Abhängigkeiten dem Bericht hinzugefügt (siehe Abschnitt 3.3.11).

3.3.5 Bericht - Attribut Hülle

- *Berichtersteller*: AttributeClosureReporter
- *Methode*: report
- *Analyse*: AttributeClosureAnalysis

Dieser Bericht stellt das an einen Studenten returnierte Feedback für die Abgabe eines Beispiels vom Typ Berechnen der Attribut Hülle dar. Abbildung 17 zeigt den Aufbau dieses Berichts.

Abbildung 17: Bericht – Attribut Hülle.

Wurde beim Analysieren festgestellt, dass in der Attribut Hülle der Beispiel Abgabe falsche Attribute enthalten sind bzw. Attribute fehlen wird dem Bericht eine entsprechende Meldung hinzugefügt (siehe Abschnitt 3.3.8).

3.3.6 Bericht - Schlüssel Bestimmen

- *Berichtersteller*: KeysReporter
- *Methode*: report
- *Analyse*: KeysAnalysis

Dieser Bericht stellt das an einen Studenten returnierte Feedback für die Abgabe eines Beispiels vom Typ Schlüssel Bestimmen dar. Abbildung 18 zeigt den Aufbau dieses Berichts.

Abbildung 18: Bericht – Schlüssel Bestimmen.

Wurde beim Analysieren der Beispiel Abgabe festgestellt, dass falsche Schlüssel angegeben wurden bzw. Schlüssel fehlen, wird dem Bericht eine entsprechende Meldung hinzugefügt (siehe Abschnitt 3.3.9).

3.3.7 Bericht – Finden von Normalform Verletzungen

- *Berichtersteller*: NormalformReporter
- *Methode*: report
- *Analyse*: NormalformDeterminationAnalysis

Dieser Bericht stellt das an einen Studenten returnierte Feedback für die Abgabe eines Beispiels vom Typ Finden von Normalform Verletzungen dar. Abbildung 19 zeigt den Aufbau dieses Berichts.

Abbildung 19: Bericht – Finden von Normalform Verletzungen.

Wurde beim Analysieren der Beispiel Abgabe festgestellt, dass die Normalform vom Studenten falsch erkannt wurde, wird dem Bericht eine entsprechende Meldung hinzugefügt (siehe Abschnitt 3.3.15).

Wurde beim Analysieren der Beispiel Abgabe festgestellt, dass Normalform Verletzungen vom Studenten falsche erkannt wurden, wird dem Bericht ebenfalls eine entsprechende Meldung hinzugefügt (siehe Abschnitt 3.3.16).

3.3.8 Meldung – Fehlende/Falsche Attribute

- *Reporter*: AttributeClosureReporter
- *Methode*: createAttributeClosureReport
- *Analyse*: AttributeClosureAnalysis
- *Meldung*: Incorrect attribute closure

Diese Meldung zeigt fehlende bzw. flasche Attribute in einer Attribut Hülle auf. Tabelle 10 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Fehlende Attribute	Angabe, dass zumindest ein Attribut fehlt.	Anzahl der fehlenden Attribute	Auflistung der fehlenden Attribute
Falsche Attribute	Angabe, dass zumindest ein Attribut falsch ist.	Anzahl der falschen Attribute	Auflistung der falschen Attribute

Tabelle 10: Beschreibung - Fehlende/falsche Attribute.

3.3.9 Meldung – Fehlende/Falsche Schlüssel

- *Reporter*: KeysReporter
- *Methode*: createKeysErrorReport
- *Analyse*: KeysAnalysis
- *Meldung*: Incorrect keys

Diese Meldung zeigt fehlende bzw. falsche Schlüssel einer Relation auf. Tabelle 11 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Fehlende Schlüssel	Angabe, dass zumindest ein Schlüssel fehlt.	Anzahl der fehlenden Schlüssel	Auflistung der fehlenden Schlüssel

Falsche Schlüssel	Angabe, dass zumindest ein Schlüssel falsch ist.	Anzahl der falschen Schlüssel	Auflistung der falschen Schlüssel

Tabelle 11: Beschreibung - Fehlende/falsche Schlüssel.

3.3.10 Meldung – Nicht kanonisch dargestellte Funktionale Abhängigkeiten

- *Reporter*: MinimalCoverReporter
- *Methode*: createCanonicalRepresentationErrorReport
- *Analyse*: CanonicalRepresentationAnalysis
- *Meldung*: Incorrect canonical representation

Diese Meldung zeigt nicht kanonisch dargestellte Funktionale Abhängigkeiten auf. Tabelle 12 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Funktionale Abhängigkeiten, die sich nicht in kanonischer Darstellung befinden	Angabe, dass sich zumindest eine Funktionale Abhängigkeit nicht in kanonischer Darstellung befindet	Anzahl an Funktionale Abhängigkeiten, die sich nicht in kanonischer Darstellung befinden	Auflistung der Funktionale Abhängigkeiten, die sich nicht in kanonischer Darstellung befinden

Tabelle 12: Beschreibung - Nicht kanonisch dargestellte Funktionale Abhängigkeiten.

3.3.11 Meldung – Fehlende/Falsche Funktionale Abhängigkeiten

- *Reporter*: MinimalCoverReporter
- *Methode*: createDependenciesCoverErrorReport
- *Analyse*: DependenciesCoverAnalysis
- *Meldung*: Incorrect number of functional dependencies

Diese Meldung zeigt fehlende bzw. falsche Funktionale Abhängigkeiten in der Hülle einer Menge Funktionaler Abhängigkeiten auf. Tabelle 13 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Fehlende Funktionale Abhängigkeiten	Angabe, dass zumindest eine Funktionale Abhängigkeit fehlt	Anzahl an fehlenden Funktionale Abhängigkeiten.	Auflistung der fehlenden Funktionale Abhängigkeiten

Falsche Funktionale Abhängigkeiten	Angabe, dass zumindest eine Funktionale Abhängigkeit falsch ist	Anzahl an falschen Funktionale Abhängigkeiten	Auflistung der falschen Funktionale Abhängigkeiten

Tabelle 13: Beschreibung - Fehlende/falsche Funktionale Abhängigkeiten.

3.3.12 Meldung – Überflüssige Attribute

- *Reporter*: MinimalCoverReporter
- *Methode*: createExtraneousAttributesErrorReport
- *Analyse*: ExtraneousAttributesAnalysis
- *Meldung*: Extraneous attribute

Diese Meldung zeigt Attribute von Funktionale Abhängigkeiten auf die überflüssig sind. Tabelle 14 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Überflüssige Attribute	Angabe, dass in zumindest einer Funktionale Abhängigkeit ein überflüssiges Attribut gefunden wurde.	Anzahl der überflüssigen Attribute	Auflistung der überflüssigen Attribute pro Funktionale Abhängigkeit

Tabelle 14: Beschreibung - Überflüssige Attribute.

3.3.13 Meldung – Redundante Funktionale Abhängigkeiten

- *Reporter*: MinimalCoverReporter
- *Methode*: createRedundandDependenciesErrorReport
- *Analyse*: RedundandDependenciesAnalysis
- *Meldung*: Redundand functional dependency

Diese Meldung zeigt redundante Funktionale Abhängigkeiten auf. Tabelle 15 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Redundante Funktionale Abhängigkeiten	Angabe, dass zumindest eine Funktionale Abhängigkeit	Anzahl der redundanten Funktionale Abhängigkeiten	Auflistung der redundanten Funktionale Abhängigkeiten

	redundant ist.		

Tabelle 15: Beschreibung - Redundante Funktionale Abhängigkeiten.

3.3.14 Meldung – Triviale Funktionale Abhängigkeiten

- *Reporter*: MinimalCoverReporter
- *Methode*: createTrivialDependenciesErrorReport
- *Analyse*: TrivialDependenciesAnalysis
- *Meldung*: Trivial functional dependency

Diese Meldung zeigt triviale Funktionale Abhängigkeiten auf. Tabelle 16 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Triviale Funktionale Abhängigkeiten	Angabe, dass zumindest eine Funktionale Abhängigkeit trivial ist.	Anzahl der trivialen Funktionale Abhängigkeiten	Auflistung der trivialen Funktionale Abhängigkeiten

Tabelle 16: Beschreibung - Triviale Funktionale Abhängigkeiten.

3.3.15 Meldung – Falsch erkannte Normalform

- *Reporter*: NormalformReporter
- *Methode*: createNormalformLevelDeterminationErrorReport
- *Analyse*: NormalformDeterminationAnalysis
- *Meldung*: Incorrect Normalform Level

Diese Meldung zeigt auf, dass die Normalform einer Relation falsch erkannt wurde. Tabelle 17 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Falsche Normalform einer Relation	Angabe, dass Normalform nicht korrekt erkannt wurde.	Angabe, dass Normalform nicht korrekt erkannt wurde.	Angabe, dass Normalform nicht korrekt erkannt wurde inkl. Angabe der korrekten Normalform

Tabelle 17: Beschreibung - Falsch erkannte Normalform.

3.3.16 Meldung – Falsch erkannte Normalform Verletzungen

- *Reporter*: NormalformReporter
- *Methode*: createNormalformLevelViolationErrorReport
- *Analyse*: NormalformDeterminationAnalysis
- *Meldung*: At least one Functional Dependency does not violate the specified Normalform

Diese Meldung zeigt falsch erkannte Normalform Verletzungen auf. Tabelle 18 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Funktionale Abhängigkeiten, die keine Verletzung der angegebenen Normalform darstellen	Angabe, dass zumindest bei einer Funktionale Abhängigkeit keine Verletzung der angegebenen Normalform besteht	Anzahl der Funktionale Abhängigkeiten, bei denen keine Verletzung der angegebenen Normalform besteht.	Auflistung der Funktionale Abhängigkeiten, bei denen keine Verletzung der angegebenen Normalform besteht inklusive der tatsächlich durch die jeweilige Funktionale Abhängigkeit verletzten Normalform, sofern überhaupt eine Verletzung einer Normalform besteht.

Tabelle 18: Beschreibung - Falsch erkannte Normalform Verletzungen.

3.3.17 Meldung – Unzureichende Normalform einer Ergebnisrelation

- *Reporter*: NormalformReporter
- *Methode*: createNormalformErrorReport
- *Analyse*: NormalformAnalysis
- *Meldung*: Insufficient Normalform Level (inklusive Angabe der erreichten und der geforderten Normalform)

Diese Meldung zeigt auf, dass die Normalform einer Ergebnisrelation nicht der geforderten Normalform entspricht. Tabelle 19 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Funktionale Abhängigkeiten, die eine Verletzung der 2.	Auflistung der Funktionale Abhängigkeiten, die eine Verletzung der 2.	Angabe pro Funktionale Abhängigkeit, dass in der rechten Seite der Funktionale	Auflistung pro Funktionale Abhängigkeit der nicht primen Attribute der

Normalform darstellen	Normalform darstellen	Abhängigkeit zumindest ein nicht primes Attribut enthalten ist. Angabe pro Funktionale Abhängigkeit, dass die linke Seite der Funktionale Abhängigkeit ein Teilschlüssel ist.	rechten Seite. Angabe pro Funktionale Abhängigkeit, dass die linke Seite der Funktionale Abhängigkeit ein Teilschlüssel ist.
Funktionale Abhängigkeiten, die eine Verletzung der 3. Normalform darstellen	Auflistung der Funktionale Abhängigkeiten, die eine Verletzung der 3. Normalform darstellen	Angabe pro Funktionale Abhängigkeit, dass in der rechten Seite der Funktionale Abhängigkeit zumindest ein nicht primes Attribut enthalten ist. Angabe pro Funktionale Abhängigkeit, dass die linke Seite der Funktionale Abhängigkeit kein Oberschlüssel ist.	Auflistung pro Funktionale Abhängigkeit der nicht primen Attribute der rechten Seite. Angabe pro Funktionale Abhängigkeit, dass die linke Seite der Funktionale Abhängigkeit kein Oberschlüssel ist.
Funktionale Abhängigkeiten, die eine Verletzung der BCNF darstellen	Auflistung der Funktionale Abhängigkeiten, die eine Verletzung der BCNF darstellen	Angabe pro Funktionale Abhängigkeit, dass die linke Seite der Funktionale Abhängigkeit kein Schlüssel ist.	Angabe pro Funktionale Abhängigkeit, dass die linke Seite der Funktionale Abhängigkeit kein Schlüssel ist.

Tabelle 19: Beschreibung - Unzureichende Normalform einer Ergebnisrelation.

3.3.18 Meldung – Ungültige Zerlegung

- *Reporter*: NormalizationReporter
- *Methode*: createDecompositionErrorReport
- *Analyse*: DecompositionAnalysis
- *Meldung*: Invalid Decomposition

Diese Meldung zeigt auf, dass die Zerlegung einer Relation ungültig ist. Tabelle 20 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Fehlende Attribute	Angabe, dass zumindest ein Attribut fehlt.	Anzahl der fehlenden Attribute	Auflistung der fehlenden Attribute

Tabelle 20: Beschreibung - Ungültige Zerlegung.

3.3.19 Meldung – Zu viele verlorene Funktionale Abhängigkeiten

- *Reporter*: NormalizationReporter
- *Methode*: createDependenciesPreservationErrorReport
- *Analyse*: DependenciesPreservationAnalysis
- *Meldung*: Decomposition is not dependencies preserving

Diese Meldung zeigt auf, dass bei einer Zerlegung zu viele Funktionale Abhängigkeiten verloren wurden. Tabelle 21 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
Verlorengegangene Funktionale Abhängigkeiten	Angabe, dass zumindest eine Funktionale Abhängigkeit verloren wurde	Anzahl der verloren gegangenen Funktionale Abhängigkeiten	Auflistung der Funktionale Abhängigkeiten die verloren wurden

Tabelle 21: Beschreibung – Zu viele verlorene Funktionale Abhängigkeiten.

3.3.20 Meldung – Nicht verlustfreie Zerlegung

- *Reporter*: NormalizationReporter
- *Methode*: createLossLessErrorReport
- *Analyse*: LossLessAnalysis
- *Meldung*: Decomposition is not loss less

Diese Meldung zeigt auf, dass eine Zerlegung nicht verlustfrei ist. Tabelle 22 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
nicht verlustfreie Zerlegung	Hinweis, dass der Verbund der Teilrelationen nicht gleich der Basisrelation ist	Hinweis, dass der Verbund der Teilrelationen nicht gleich der Basisrelation ist	Hinweis, dass der Verbund der Teilrelationen nicht gleich der Basisrelation ist

Tabelle 22: Beschreibung - Nicht verlustfreie Zerlegung.

3.3.21 Meldung – Fehlende/Falsche eingebettete Funktionale Abhängigkeiten

- *Reporter*: RBRReporter

- *Methode*: createRBRErrorReport
- *Analyse*: RBRAnalysis
- *Fehlermeldung*: Incorrect functional dependencies according to RBR-algorithm

Diese Meldung zeigt auf, dass fehlende bzw. falsche eingebettete Funktionale Abhängigkeiten festgestellt wurden. Tabelle 23 zeigt die Beschreibung dieser Meldung für die einzelnen Diagnosestufen.

Abweichung	**Beschreibung**		
	low	medium	high
fehlende eingebettete Funktionale Abhängigkeiten	Angabe, dass zumindest eine eingebettete Funktionale Abhängigkeit fehlt	Anzahl der fehlenden eingebetteten Funktionale Abhängigkeiten	Auflistung der fehlenden eingebetteten Funktionale Abhängigkeiten
falsche eingebettete Funktionale Abhängigkeiten	Angabe, dass zumindest eine eingebettete Funktionale Abhängigkeit falsch ist	Anzahl der falschen eingebetteten Funktionale Abhängigkeiten	Auflistung der falschen eingebetteten Funktionale Abhängigkeiten

Tabelle 23: Beschreibung - Fehlende/Falsche eingebettete Funktionale Abhängigkeiten.

4 Beispiel Administration

In diesem Kapitel werden die vom RDBD Modul zur Verfügung gestellten Funktionalitäten für die Beispiel Administration beschrieben. Abschnitt 4.1 erläutert das Anlegen einer Beispiel Angabe. Im Abschnitt 4.2 wird das automatisierte Erstellen des Textes einer Beispiel Angabe erläutert. Dieser Abschnitt ist auch an Benutzer des RDBD Moduls adressiert. Im Abschnitt 4.4 bzw. Abschnitt 4.5 sind das Laden einer neuen bzw. einer bestehenden Beispiel Angabe beschrieben. Die Abschnitte 4.3 und 4.6 behandeln das Löschen und Ändern einer Beispiel Angabe.

Wie bereits im Abschnitt 2.1.1 erläutert, ist die Funktionalität für die Administration von Beispielen in der Klasse `RDBDExerciseManager` und die zugehörige Komponente der Benutzerschnittstelle im Servlet `RDBDExerciseManagerServlet` implementiert. Die Beispieltyp spezifischen Ausprägungen der Klasse `RDBDExerciseManager` bzw. des Servlets `RDBDExerciseManagerServlet` dienen lediglich zur Konfiguration des eTutor Cores. Dieser benötigt für jeden einzelnen Beispieltyp eine eigene Komponente für die Funktionalität bzw. Benutzerschnittstelle der Beispiel Administration.

4.1 Anlegen einer Beispiel Angabe

Das Anlegen einer Beispiel Angabe wird durch einen Aufruf der Methode `createExercise` der Klasse `RDBDExerciseManager` veranlasst. Das der Methode `create` übergebene Objekt das die Beispiel Angabe repräsentiert wird serialisiert und zusammen mit der ebenfalls der Methode `create` übergebenen ID im Beispiel Repository in einem BLOB gespeichert. Tabelle 24 zeigt nach Beispieltypen gegliedert den Aufbau der Klassen, die zur Repräsentation von Beispiel Angaben verwendet werden.

Beispiel Angabe	*Beschreibung*
RBR Algorithmus	
RBRSpecification	
Collection baseAttributes	Die Attribute der Teilrelation
Relation baseRelation	Die darin enthaltenen Funktionalen Abhängigkeiten stellen die Funktionalen Abhängigkeiten der Ausgangsrelation dar. Die darin enthaltenen Attribute stellen die Attribute der Ausgangsrelation dar.
Allgemeine Normalisierung	
NormalizationSpecification	
int maxLostDependencies	Die maximale erlaubte Anzahl an verlorenen Funktionalen Abhängigkeiten
NormalformLevel targetLevel	Die zu erreichende Normalform der Ergebnisrelationen
Relation baseRelation	Die darin enthaltenen Attribute stellen die Attribute der Ausgangsrelation dar. Die darin enthaltenen Funktionalen Abhängigkeiten stellen die Funktionalen Abhängigkeiten der Ausgangsrelation dar.
Attribut Hülle	
AttributeClosureSpecification	
Collection baseAttributes	Die Attribute deren Hülle zu berechnen ist.
Relation baseRelation	Die darin enthaltenen Attribute stellen die Attribute der

	Ausgangsrelation dar. Die darin enthaltenen Funktionalen Abhängigkeiten stellen die Funktionalen Abhängigkeiten der Ausgangsrelation dar.
Decompose Algorithmus	
DecomposeSpecification	
int maxLostDependencies	Die maximale erlaubte Anzahl an verlorenen Funktionalen Abhängigkeiten
NormalformLevel targetLevel	Die zu erreichende Normalform der Ergebnisrelationen
Relation baseRelation	Die darin enthaltenen Attribute stellen die Attribute der Ausgangsrelation dar. Die darin enthaltenen Funktionalen Abhängigkeiten stellen die Funktionalen Abhängigkeiten der Ausgangsrelation dar.
Minimale Überdeckung	
Relation	
Collection attributes	Die Attribute der Ausgangsrelation
Collection dependencies	Die Funktionalen Abhängigkeiten der Ausgangsrelation
Schlüssel Bestimmen	
Relation	
Collection attributes	Die Attribute der Ausgangsrelation
Collection dependencies	Die Funktionalen Abhängigkeiten der Ausgangsrelation
Normalform Verletzungen	
Relation	
Collection attributes	Die Attribute der Ausgangsrelation
Collection dependencies	Die Funktionalen Abhängigkeiten der Ausgangsrelation

Tabelle 24: Klassen zur Repräsentation von Beispiel Angaben.

Beim Anlegen einer Beispiel Angabe besteht grundsätzlich das Problem, dass unter Umständen gleiche Beispiel Angaben erfasst werden könnten wodurch unerwünschte Duplikate entstehen würden. Im RDBD Modul wurde eine Möglichkeit gefunden, dass die Überprüfung auf Duplikate einer Beispiel Angabe, nicht mehr vom Benutzer durchgeführt werden muss sondern automatisiert durchgeführt werden kann. Die Basis dafür bildet ein Mechanismus zur Überprüfung der semantischen Gleichheit zweier Beispiel Angaben. Jede Klasse, die zur Repräsentation einer Beispiel Angabe verwendet wird, implementiert eine Methode `boolean semanticallyEquals(Object o)`. Diese Methode liefert true, sofern das übergebene Objekt als der Beispiel Angabe semantisch gleich erkannt wird, sonst false. Soll eine Beispiel Angabe im RDBD Modul angelegt werden, so werden alle im Beispiel Repository gespeicherten Beispiel Angaben des jeweiligen Beispieltyps ausgelesen. Für jede dieser Beispiel Angaben wird überprüft, ob die Methode `semanticallyEquals` parametrisiert mit der neu anzulegenden Beispiel Angabe true liefert. Trifft dies für eine der gespeicherten Beispiel Angaben zu, so wird die neu anzulegende Beispiel Angabe nicht im Beispiel Repository gespeichert und eine entsprechende Fehlermeldung in der Benutzerschnittstelle angezeigt.

Beispiel Angabe	*Bedingungen für Semantische Gleichheit*
RBRSpecification	
Collection baseAttributes	In beiden Beispiel Angaben müssen die gleichen Attribute in dieser Property enthalten sein. Die Reihenfolge der Attribute spielt dabei keine Rolle.
Relation baseRelation	Siehe Bedingungen für semantische Gleichheit der Klasse Relation.

NormalizationSpecification	
int maxLostDependencies	Die beiden Beispiel Angaben müssen den selben Wert in diesem Property aufweisen.
NormalformLevel targetLevel	Die beiden Beispiel Angaben müssen den selben Wert in diesem Property aufweisen.
Relation baseRelation	Siehe Bedingungen für semantische Gleichheit der Klasse Relation.
AttributeClosureSpecification	
Collection baseAttributes	In beiden Beispiel Angaben müssen die gleichen Attribute in dieser Property enthalten sein. Die Reihenfolge der Attribute spielt dabei keine Rolle.
Relation baseRelation	Siehe Bedingungen für semantische Gleichheit der Klasse Relation.
DecomposeSpecification	
int maxLostDependencies	Die beiden Beispiel Angaben müssen den selben Wert in diesem Property aufweisen.
NormalformLevel targetLevel	Die beiden Beispiel Angaben müssen den selben Wert in diesem Property aufweisen
Relation baseRelation	Siehe Bedingungen für semantische Gleichheit der Klasse Relation.
Relation	
Collection attributes	In beiden Beispiel Angaben müssen die gleichen Attribute in diesem Property enthalten sein. Die Reihenfolge der Attribute spielt dabei keine Rolle.
Collection dependencies	In beiden Beispiel Angaben müssen die gleichen Funktionale Abhängigkeiten in diesem Property enthalten sein. Eine Funktionale Abhängigkeit wird als gleich einer anderen Funktionale Abhängigkeit erachtet, sofern sie die jeweils gleichen Attribute in der linken und der rechten Seite aufweisen. Die Reihenfolge der Attribute spielt dabei keine Rolle.
Collection keys	In beiden Beispiel Angaben müssen die gleichen Schlüssel in diesem Property enthalten sein. Ein Schlüssel wird als gleich einem anderen Schlüssel erachtet, sofern sie die gleichen Attribute aufweisen. Die Reihenfolge der Attribute spielt dabei keine Rolle. *Hinweis*: Der Name einer Relation wird bei der Prüfung auf semantische Gleichheit nicht berücksichtigt.

Tabelle 25: Semantische Gleichheit von Beispiel Angaben.

Die Methode `semanticallyEquals` liefert false, sofern die zu überprüfende Beispiel Angabe null oder nicht vom gleichen Beispieltyp ist. Trifft dies nicht zu, so werden die Properties der einzelnen Klassen auf semantische Gleichheit überprüft. Die Vorgehensweise bei diesen Überprüfungen können Sie Tabelle 25 entnehmen.

Für das Erfassen einer Beispiel Angabe ist es grundsätzlich bei jedem Beispieltyp notwendig, eine Menge an Attributen oder Funktionalen Abhängigkeiten zu erfassen. Um den Vorgang des Erfassens solcher Informationen in der Benutzerschnittstelle für versierte Benutzer zu beschleunigen, wird die Möglichkeit geboten, Attribute und Funktionale Abhängigkeiten textuell zu beschreiben. Genaueres dazu finden Sie im Allgemeinen Benutzerhandbuch des eTutor System. Diese textuellen Angaben werden vom sogenannten `SpecificationEditor` geparst und in JAVA Objekte transformiert. Aus diesem Grund arbeiten die Komponenten der Beispiel Administration

nicht direkt mit den für die Repräsentation der jeweiligen Beispiel Angabe vorgesehenen Objekten sondern mit Objekten der Klasse `SpecificationEditor`. In diesen Objekten sind die eigentlichen Objekte zur Repräsentation der Beispiel Angabe gekapselt. Folglich wird der Methode `createExercise` nicht direkt eine Beispiel Angabe übergeben, sondern eben ein Objekt der Klasse `SpecificationEditor`, das die Beispiel Angabe kapselt. Analog gilt dies auch für die Methoden `modifyExercise`, `deleteExercise`, `fetchExercise`, `updateExercise` und `generateHTML`.

4.2 Erzeugen des Angabetextes

Das RDBD Modul ermöglicht die automatisierte Erzeugung des Angabetextes für alle Beispiele aller unterstützen Beispieltypen in Englisch und Deutsch.

Das Erzeugen des Textes einer Beispiel Angabe wird durch einen Aufruf der Methode `generateHTML` der Klasse `RDBDExerciseManager` veranlasst. Dieser Methode wird die Beispiel Angabe gekapselt in einem Objekt der Klasse `SpecificationEditor` (siehe Abschnitt 4.1) übergeben, auf deren Basis der Angabetext erzeugt wird.

RBR Algorithmus

Berechnen Sie die funktionalen Abhängigkeiten $\mathbf{F}_S$ für das Subschema **S** der Relation **R** mit den Funktionalen Abhängigkeiten **F**.

R (« Attribute der Ausgangsrelation »)
F = { « Funktionale Abhängigkeiten der Ausgangsrelation » }
S (« Attribute der Teilrelation »)

Allgemeine Normalisierung

Finden Sie eine **verlustfreie Zerlegung** der Relation **R0** mit den Funktionalen Abhängigkeiten **F** in « Gewünschte Normalform ». Geben Sie für jede Teilrelation die Schlüssel und die von **F** ableitbaren Funktionalen Abhängigkeiten an. Sie dürfen bei der Zerlegung maximal « Maximale Anzahl an verlorenen Funktionale Abhängigkeiten » Funktionale Abhängigkeiten verlieren!

R (« Attribute der Ausgangsrelation »)
F = { « Funktionale Abhängigkeiten der Ausgangsrelation » }

Decompose Algorithmus

Wenden Sie den Decompose Algorithmus an, um eine **verlustfreie Zerlegung** der Relation **R1** mit den Funktionalen Abhängigkeiten **F** in « Gewünschte Normalform » zu finden. Geben Sie für jede Teilrelation die Schlüssel und die von **F** ableitbaren Funktionalen Abhängigkeiten an. Sie dürfen bei der Zerlegung maximal « Maximale Anzahl an verlorenen Funktionale Abhängigkeiten » Funktionale Abhängigkeiten verlieren!

R (« Attribute der Ausgangsrelation »)
F = { « Funktionale Abhängigkeiten der Ausgangsrelation » }

Minimale Überdeckung

Geben Sie für die Menge **F** an Funktionalen Abhängigkeiten eine minimale Überdeckung an.

F = { « Funktionale Abhängigkeiten » }

Hinweis: Streichen Sie alle redundanten Funktionalen Abhängigkeiten und alle redundanten Attribute in den linken Seiten der Funktionalen Abhängigkeiten.
Attribut Hülle
Berechnen Sie die Hülle der Attribut Kombination **A** bezüglich der Menge an Funktionalen Abhängigkeiten **F** der Relation **R**. **R** (« Attribute der Ausgangsrelation ») **F** = { « Funktionale Abhängigkeiten der Ausgangsrelation » } **A** (« Basis Attribute »)
Schlüssel Bestimmen
Berechnen Sie alle Schlüssel der Relation **R** auf Basis der Funktionalen Abhängigkeiten **F**. **R** (« Attribute der Ausgangsrelation ») **F** = { « Funktionale Abhängigkeiten der Ausgangsrelation » }
Finden von Normalform Verletzungen
Geben Sie an, in welcher Normalform sich die Relation **R** mit den Funktionalen Abhängigkeiten **F** befindet. Geben Sie weiters für jede Funktionale Abhängigkeit $\mathbf{F_i}$ an, welche Normalform durch $\mathbf{F_i}$ verletzt wird. **R** (« Attribute der Ausgangsrelation ») **F** = { « Funktionale Abhängigkeiten der Ausgangsrelation » }

Tabelle 26: Automatisiert erzeugte Angabetexte.

Tabelle 26 zeigt für jeden Beispieltyp welcher Angabetextes erzeugt wird. Dabei werden nur die Angabetexte in Deutsch gezeigt. Die Angabetexte in Englisch sind gleich strukturiert. Die in Spitzen Klammern eingeschlossenen Teile der Angabetexte stellen von der konkreten Beispielangabe abhängige Teile dar.

4.3 Löschen einer Beispiel Angabe

Die Methode `deleteExercise` der Klasse `RDBDExerciseManager` realisiert die zum Löschen einer Beispiel Angabe notwendige Funktionalität. Die Beispiel Angabe mit der ID, die der Methode `deleteExercise` übergeben wird, wird im Beispiel Repository gelöscht.

4.4 Laden einer neuen Beispiel Angabe

Die Methode `fetchExerciseInfo` der Klasse `RDBDExerciseManager` realisiert die zum Laden einer neuen Beispiel Angabe notwendige Funktionalität. Diese Methode wird vom eTutor Core aufgerufen um beim Erstellen einer neuen Beispiel Angabe ein Objekt zu ermitteln, das die Beispieltyp spezifischen Informationen einer neu initialisierten Beispiel Angabe repräsentiert. Dieses Objekt ist wiederum vom Typ `SpecificationEditor` und kapselt die eigentliche Beispiel Angabe (siehe Abschnitt 4.1).

4.5 Laden einer bestehenden Beispiel Angabe

Die Methode `fetchExercise` der Klasse `RDBDExerciseManager` realisiert die zum Laden einer bestehenden Beispiel Angabe notwendige Funktionalität. Diese Methode wird vom eTutor Core aufgerufen um beim Ändern eines existierenden Beispiels ein Objekt zu ermitteln, das die Beispieltyp spezifischen Informationen der jeweiligen Beispiel Angabe repräsentiert. Die zu der der Methode `fetchExercise` übergeben ID passende Beispiel Angabe wird aus dem Beispiel Repository geladen und wiederum in einem Objekt der Klasse `SpecificationEditor` (siehe Abschnitt 4.1) gekapselt. Dieses Objekt wird dann zurückgegeben.

4.6 Ändern einer Beispiel Angabe

Die Methode `updateExercise` der Klasse `RDBDExerciseManager` realisiert die zum Ändern einer Beispiel Angabe notwendige Funktionalität. Dieser Methode werden die ID der zu ändernden Beispiel Angabe und ein Objekt, das die neue Beispiel Angabe repräsentiert übergeben. Dieses Objekt ist wiederum vom Typ `SpecificationEditor` und kapselt die eigentliche Beispiel Angabe (siehe Abschnitt 4.1). Die der ID zugehörige Beispiel Angabe wird aus dem Beispiel Repository gelöscht und die neue Beispiel Angabe wird zu der selben ID im Beispiel Repository eingefügt.

5 Hinweise zur Weiterentwicklung des Moduls

5.1 Verbesserung der Bewertungsfunktionalität

Die Bewertungsfunktionalität im RDBD Modul ist derzeit nur rudimentär realisiert. Aufgrund der feinen Analysen im RDBD Modul könnte eine generische Komponente für die Bewertung beliebiger Analysen des RDBD Moduls entwickelt werden.

Die Grundidee hinter dieser generischen Bewertungskomponente ist, dass es eine Konfigurationsmöglichkeit gibt, in der zu einer Analyse eine bestimmte Punktezahl (absolut oder relativ) vermerkt wird. Die generische Bewertungskomponente bekommt als Input eine Analyse und eine solche Konfiguration und retourniert die entsprechenden Punkte retournieren. Die konfigurierten Punkte werden vergeben, wenn in der Analyse keine Abweichungen festgestellt worden sind (Flag `submissionSuitsSolution`). Um zu erreichen, dass Punkte nicht nur für Arten von Abweichungen, sondern für konkrete Abweichungen gegeben werden können oder nicht, müsste in jeder Analyse zusätzlich vermerkt sein, wie viele Abweichungen der jeweiligen Art enthalten sind (zum Beispiel in einem Flag `deviationsCount`). Da die Analysen im RDBD Modul verschachtelt sind, müsste auch die generische Bewertungskomponente in einer Analyse enthaltene (Sub-)Analysen erkennen und diese wiederum nach einer bestimmten Konfiguration bewerten.

Beispiel Konfiguration "Analyse Trivialer Funktionaler Abhängigkeiten": Beim Analysieren Trivialer Funktionaler Abhängigkeiten entsteht eine Analyse der Klasse TrivialDependenciesAnalysis. Angenommen es gäbe eine Konfiguration, in der vermerkt ist, dass zur Klasse TrivialDependendenciesAnalysis für jede festgestellte triviale Funktionale Abhängigkeit (Flag deviationsCount) 0,5 Punkte abgezogen werden sollen, aber maximal 2 Punkte. Mit dieser Konfiguration könnte die generische Bewertungskomponente zusammen mit einer konkreten Analyse Trivialer Funktionaler Abhängigkeiten eine Bewertung vornehmen.

Die notwendigen Konfigurationsmöglichkeiten müssen noch detailliert erarbeitet werden. Allerdings verspricht diese Vorgehensweise, dass ohne Source Code Adaptionen beliebig komplexe Bewertungsstrategien verfolgt werden können. Es könnte auch überlegt werden, inwiefern eine solche Bewertungskomponente nicht auch für andere Module verwendbar wäre.

5.2 Einbindung in das eTutor System über RMI

Da das RDBD Modul derzeit noch keine Einbindung in das eTutor System über RMI unterstützt, muss es am Web-Container des eTutor Cores laufen. Um eventuelle Ressourcen Engpässe zu vermeiden, sollte eine RMI Unterstützung im RDBD Modul realisiert werden. Die RMI Unterstützung kann dabei analog zu der anderer Module implementiert werden (vgl. Modul JDBC).

5.3 Adaption des Modells von Berichten

Das Modell von Berichten ist im RDBD Modul über längere Zeigt hinweg gewachsen. Aus diesem Grund, passen einige der Klassen- und Properties Namen nicht mehr mit dem zusammen, was sie eigentlich repräsentieren sollen. Zum Beispiel Repräsentiert die Klasse ErrorReport eine Meldung, was offensichtlich nicht erwartungskonform ist. Das Modell von Berichten sollte grundsätzlich überarbeitet werden, so dass es der im Abschnitt 3.3 dargestellten Semantik entspricht.

Abbildungsverzeichnis