Universität Rostock

Traditio et Innovatio

Masterarbeit

Inverse Anfragen in ProSA

Vorgelegt von:

Dennis Spolwind

Eingereicht am:

15. März 2022

Betreuer:

M.Sc. Tanja Auge

apl.Prof. Dr.-Ing. habil. Meike Klettke

UNIVERSITÄT ROSTOCK | FAKULTÄT FÜR INFORMATIK UND ELEKTROTECHNIK

Institut für Datenbank- und Informationssysteme

Albert-Einstein-Straße 22, 18059 Rostock| Tel.: +49 381 498 7591

Abstract

Im Bereich des *Forschungsdatenmangament* wird eine Analyse, Archivierung und Verarbeitung von großen Mengen an Forschungsdaten durchgeführt. Für die Unterstützung dieser Auswertung, Archivierung und Analyse von Forschungsdaten wurde das Tool *ProSA* entwickelt. Dabei verfolgt das Tool *ProSA* das Ziel eine Rückverfolgung der Anfrageergebnisse zu realisieren. Um eine Rückverfolgbarkeit eines Anfrageergebnis zu gewährleisten, ist eine Berechnung einer minimalen Teildatenbank notwendig. Die minimale Teildatenbank beschreibt die Archivierung der verwendeten Daten, die für die Generierung des Anfrageergebnis benötigt werden. Für die Berechnung einer minimalen Teildatenbank wird eine invertierte Anfrage benötigt, um eine Rückabbildung vom Anfrageergebnis auf die verwendeten Originaldaten zu realisieren.
Das Ziel der Masterarbeit: *Inverse Anfragen in ProSA* ist eine Invertierungsmethode, für die Generierung einer invertierten Anfrage, zu konstruieren. Diese Konstruktion wird basiert auf bestehende Invertierungsmethoden aufgebaut, die in vorherigen Arbeiten an der Universität Rostock entwickelt wurden und für die Invertierung von Anfragen entsprechend erweitert. Anschließend wird die Implementierung der entwickelten Invertierungsmethode für die Anwendung mit ProSA durchgeführt. Als weiteres Ziel dieser Masterarbeit dient die Definition der Darstellung der Anfragen als Menge von Abhängigkeiten und die Definition der Invertierung der Abhängigkeiten. Diesbezüglich werden unterschiedlichste SQL-Anfragen und dessen Darstellung als Abhängigkeiten betrachtet und auf Vollständigkeit untersucht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Der Begriff des *Forschungsdatenmanagement* umfasst die Analyse, Archivierung und Verarbeitung von großen Mengen an Forschungsdaten, die beispielsweise im Rahmen von Projekten, Experimenten und Beobachtungen anfallen [BHSS17]. Um diese Auswertung, Analyse und Archivierung von Daten zu unterstützen bzw. durchzuführen wurde das Tool *ProSA* (Provenance Management durch Schema-Abbildungen und Annotationen) [AH19] entwickelt. Das Tool ProSA umfasst eine Kombination aus Data Provenance, Schema- und Datenevolution, auf Basis von (relationalen) Datenbanken. Im Forschungsbereich des Provenance Management bzw. Data Provenance wird eine mögliche Rückverfolgbarkeit vom Anfrageergebnis zu den verwendeten Daten aus der Originalrelation untersucht. Ebenfalls werden die Reproduzierbarkeit und Nachvollziehbarkeit von Ergebnissen geprüft, welches ein wichtiges Ziel bzw. Herausforderung im Bereich des Forschungsdatenmanagement ist. Dementsprechend beschäftigt sich Data Provenance mit der Beantwortung folgender Fragestellung:

- Woher kommen die verwendeten Daten die zum Ergebnis führen? (**Where**)
- Welche Daten waren bei der Ergebnisbildung beteiligt? (**Why**)
- Wie wurde das Ergebnis berechnet? (**How**)
- Warum fehlt im Ergebnis ein bestimmter erwarteter Wert? (**Why not**)

Ein weiterer wichtiger Aspekt bei ProSA ist die Berechnung einer minimalen Teildatenbank [AH19]. Diese minimale Teildatenbank dient zur Archivierung der verwendeten Daten in der Orginaldatenbank, um bei Veränderung der Forschungsdaten bzw. der Datenbank die Reproduzierbarkeit eines Ergebnisses, zu einem späteren Zeitpunkt, weiterhin zu gewährleisten. Aufgrund dessen, dass ohne diese Berechnung bei jeder Veränderung der Datenbank der komplette Datenbestand für die Reproduktion des Ergebnisses gespeichert werden muss, ist die Berechnung der minimalen Teildatenbank ein entscheidendes Problem im Bereich Forschungsdatenmanagement. Um diese Berechnung durchzuführen kommt in ProSA Data Provenance in Kombination mit dem CHASE zum Einsatz. Dabei dient der CHASE im Allgemeinen für die Einarbeitung von Abhängigkeiten/Parameter $\star$ in ein Objekt $\bigcirc$, um als Ergebnis ein Objekt $\circledast$ zu erhalten, das die Abhängigkeiten erfüllt:

$$chase_{\star}(\bigcirc) = \circledast$$

Im Rahmen von ProSA stellen die Parameter Anfragen dar, die auf eine Datenbankinstanz als Objekt angewendet werden, um ein Anfrageergebnis (Teil der Datenbank der die Anfragen erfüllt) zu erhalten.

1.1 Motivation und Zielsetzung

Im Rahmen dieser Arbeit werden Inversen Anfragen für ProSA untersucht. Diese Inversen dienen der Bildung einer minimalen Teildatenbank, um damit die How- und Why-Fragestellungen zu beantworten, ohne die gesamte Datenbank bei jeder Veränderung speichern zu müssen. Ein Beispiel für die Anwendung einer Inverse ist in Abbildung 1.1 dargestellt. Diesbezüglich wird eine Anfrage Q, mithilfe des CHASE, auf einer Datenbankinstanz ($I(S_1)$) angewendet und ein Anfrageergebnis (blaue Box) erzeugt. Darauffolgend wird eine Inverse Anfrage Q_{inv} von Q definiert und auf das Ergebnis ($K(S_2)$) angewendet, ebenfalls mit dem CHASE-Algorithmus. Diese Anwendung der Inverse erzeugt eine minimale Teildatenbank (rote Box), die archiviert werden muss, um die Rückverfolgbarkeit des Anfrageergebnisses zu gewährleisten. Um diese Rückverfolgung in ProSA umzusetzen, wird in dieser Arbeit typische Anfrageoperationen und dessen Invertierung untersucht, besonders in Hinblick auf verschachtelte Anfragen. An dieser Stelle sei anzuführen, dass die Anfragen erst in sogenannte S-t-tgds konvertiert werden müssen, bevor der CHASE-Algortithmus diese auf die Datenbankinstanz anwenden kann. Aus diesem Grund wird, in dieser Arbeit, die Korrektheit und Vollständigkeit der Darstellung der Anfragen in S-t-tgds überprüft. Darauffolgend wird eine Definition der Inversen, der zuvor betrachteten Anfragen, als S-t-tgds vorgenommen und ebenfalls auf Korrektheit und Vollständigkeit überprüft. Schlussendlich wird ein Invertierungsmodul in ProSA implementiert, dass eine automatische Invertierung der Anfragen durchführt. Dementsprechend werden mehrere Ansätze bzw. Algorithmen für die automatische Invertierung betrachtet und gegebenfalls erweitert.

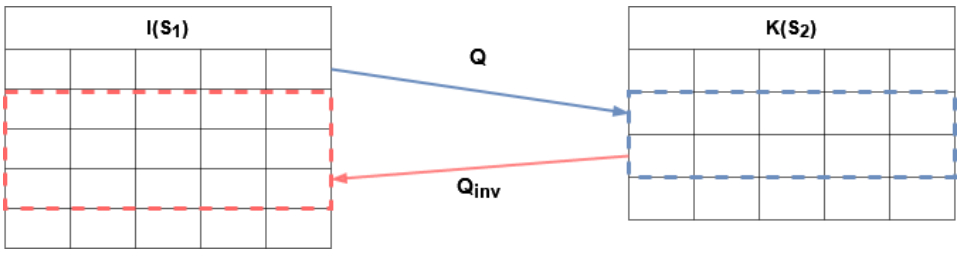

Abbildung 1.1: Beispiel für Inverse Anfragen (ähnlich wie in [AH19])

1.2 Weiterführendes Beispiel

Im weiterem Verlauf dieser Arbeit wird ein fiktiver Datensatz verwendet, um Beispiele für Anfragen, Definitionen und Methoden zu generieren. Dieser fiktiver Datensatz repräsentiert eine vereinfachte Darstellung eines Datensatzes einer Universität. In diesem Datensatz werden die Studierenden, die Veranstaltungen, die Dozenten und die Prüfungsergebnisse einer Universität aufgeführt. Infolgedessen werden fünf Tabellen definiert, die die Studierenden (Tabelle 1.1), die Veranstaltungen/Module (Tabelle 1.2), die Dozenten und Professoren (Tabelle 1.3), die Teilnehmer der Module (Tabelle 1.4) und die Prüfungsergebnisse der Studierenden (Tabelle 1.5) beinhalten. An dieser Stelle sei anzuführen, dass die fünf Tabellen aus dem Projekt ProSA [Aug22c] entnommen wurden. Für die Vereinfachung des in ProSA vorliegen fiktiven Datensatz, wurde in diesem Beispiel die Tabellen für die Teilnehmer (Tabelle 1.4) und für die Prüfungsergebnisse (Tabelle 1.5) auf das Modul *Mathematics for Computer Scientists I* beschränkt.
Der Datensatz bzw. die Tabellen beschreiben das Szenario, dass Studierende einer Universität an Veranstaltungen (Module) teilnehmen können. Hierbei werden die Veranstaltungen durch einen oder mehrere Dozenten/Professoren gehalten. Anschließend können die Studierenden eine Prüfung im jeweiligen Modul ablegen und eine Benotung für das Modul erhalten.
Im Folgenden werde die einzelnen Relationsschematas und Domänen $\mathbb{D}$ der Tabellen aufgeführt, ähnlich wie in [Aug17]. Hierbei entspricht ein Relationsschemata einer Menge von Attributen (Spalten) und die Domäne definiert den Wertebereich der einzelnen Attributen (Abschnitt 2.1.1).

- Students = $\{\text{ID}_{\text{Students}}$, MatricNo., LastName, FirstName, StudyCourse$\}$ mit
  - $\mathbb{D}(\text{ID}_{\text{Students}}) = \{S_i : i \in \mathbb{N}^+\}$
  - $\mathbb{D}(\text{MatricNo}) = \mathbb{N}^+$
  - $\mathbb{D}$(LastName) = {Fieber, Sonnenschein, Müller, Johansen, Miller, Mustermann, Johannes}
  - $\mathbb{D}$(FirstName) = {Fabian, Sarah, Max, Mira, Johannes, Mia, Paul}
  - $\mathbb{D}$(SudyCourse) = {Computer Science for Lectureship, Mathematics, Electrical Engineering, Computer Science, Electrical Engineering, ITTI}
- Modules = $\{\text{ID}_{\text{Modules}}$, ModulNo., Title, Major$\}$
  - $\mathbb{D}(\text{ID}_{\text{Modules}}) = \{M_i : i \in \mathbb{N}^+\}$
  - $\mathbb{D}(\text{ModulNo}) = \mathbb{N}^+$
  - $\mathbb{D}$(Title) = {Databases III, Mathematics for Computer Scientists I, Pattern Recognition and Context Analysis, Individual Knowledge Management, Data Warehouses and Business Intelligence, Kognitive Systems, Theory of relational Databases, Systems Biology, New Developments in Computer Science}

  - $\mathbb{D}$(Major) = {Information Systems, Smart Computing, Information Systems, Business Informatics, Models and Algorithms, $\perp$} mit $\perp$ als Nullwert

- Tutors = {$\text{ID}_{\text{Tutors}}$, Tutor, ModulNo}
  - $\mathbb{D}(\text{ID}_{\text{Tutors}}) = \{T_i : i \in \mathbb{N}^+\}$
  - $\mathbb{D}$(Tutor) = {Professor A, Tutor A, Professor B, Tutor B, Professor C, Professor D, Professor E}
  - $\mathbb{D}(\text{ModulNo}) = \mathbb{N}^+$
- Participants = {$\text{ID}_{\text{Participants}}$, ModulNo, MatricNo}
  - $\mathbb{D}(\text{ID}_{\text{Participants}}) = \{P_i : i \in \mathbb{N}^+\}$
  - $\mathbb{D}(\text{ModulNo}) = \mathbb{N}^+$
  - $\mathbb{D}(\text{MatricNo}) = \mathbb{N}^+$
- Grades = {$\text{ID}_{\text{Grades}}$, ModulNo, MatricNo, Semester, Grade}
  - $\mathbb{D}(\text{ID}_{\text{Grades}}) = \{G_i : i \in \mathbb{N}^+\}$
  - $\mathbb{D}(\text{ModulNo}) = \mathbb{N}^+$
  - $\mathbb{D}(\text{MatricNo}) = \mathbb{N}^+$
  - $\mathbb{D}$(Semester) = {WS 15/16}
  - $\mathbb{D}$(Grade) = {1.0, 1.3, 1.7, 2.0, 2.3, 2.6, 3.0, 3.3, 3.7, 4.0, 5.0}

In den Tabellen 1.1-1.5 bilden die unterstrichenen Attribute die Schlüsselattribute für die jeweiligen Tabellen (Relationen). Dabei ist ein Schlüsselattribut ein eindeutiges Attributes, welches für die Identifizierung der einzelnen Tupeln (Zeilen) in den Tabellen dient. Beispielsweise ist jeder Studierende in der Relation *Students* durch seine eindeutige Matrikelnummer (*MatricNo*) identifizierbar.
Des weiterem enthält jede Tabelle eine eindeutige und fortlaufende nummerierte Identifikationsnummer (ID), welche im Attribut ID dargestellt ist. Diese Identifikationsnummer repräsentiert die eindeutige Tupelidentifikation (Tupel-ID), die für die Anwendung von Provenance-Informationen benötigt wird. Dabei ist diese Tupel-ID kein Schlüsselattribut und dient ausschließlich für die Anwendung von Provenance. Beispielsweise dient die ID für das Aufstellen von Zeugen für die Why-Provenance. Hierbei entsprechen die Zeugen die Tupel bzw. die Tupel-IDs die bei einer Anwendung einer Anfrage, bei der Bildung des Ergebnis beteiligt waren.

$\mathbf{ID_{Students}}$	**MatricNo**	**LastName**	**FirstName**	**StudyCourse**
S_1	1	Fieber	Fabian	Computer Science for Lectureship
S_2	2	Sonnenschein	Sarah	Mathematics
S_3	3	Müller	Max	Electrical Engineering
S_3	4	Müller	Mira	Computer Science
S_5	5	Johansen	Johannes	Computer Science
S_6	6	Miller	Mia	Computer Science
S_7	7	Mustermann	Max	Electrical Engineering
S_8	8	Johannes	Paul	ITTI

Tabelle 1.1: Students-Relation

$\mathbf{ID_{Modules}}$	**ModulNo**	**Title**	**Major**
M_1	001	Databases III	Information Systems
M_2	002	Mathematics for Computer Scientists I	$\perp$
M_3	003	Pattern Recognition and Context Analysis	Smart Computing
M_4	004	Individual Knowledge Management	Information Systems
M_5	005	Data Warehouses and Business Intelligence	Business Informatics
M_6	006	Kognitive Systems	Smart Computing
M_7	007	Theory of relational Databases	Information Systems
M_8	008	Systems Biology	Models and Algorithms
M_9	009	New Developments in Computer Science	$\perp$

Tabelle 1.2: Modules-Relation

$\mathbf{ID_{Tutors}}$	**Tutor**	**ModulNo**
$T_{1.1}$	Professor A	001
$T_{1.2}$	Tutor A	001
T_2	Professor B	002
T_3	Professor C	003
T_4	Professor D	004
T_5	Tutor B	005
T_6	Professor D	006
T_7	Professor A	007
T_8	Professor E	008
T_9	Professor A	009

Tabelle 1.3: Tutors-Relation

$\mathbf{ID_{Participants}}$	ModulNo	MatricNo
P_5	002	1
P_6	002	2
P_7	002	3
P_8	002	4
P_9	002	5
P_{10}	002	6
P_{11}	002	7

Tabelle 1.4: Participants-Relation für Modul 002

$\mathbf{ID_{Grades}}$	ModulNo	MatricNo	Semester	Grade
G_5	002	1	WS 15/16	3.7
G_6	002	2	WS 15/16	1.3
G_7	002	3	WS 15/16	2.3
G_8	002	4	WS 15/16	1.0
G_9	002	5	WS 15/16	1.3
G_{10}	002	6	WS 15/16	2.0
G_{11}	002	7	WS 15/16	3.3

Tabelle 1.5: Grades-Relation für Modul 002

1.3 Aufbau der Arbeit

Im Folgendem wird ein Überblick über dem Aufbau dieser Arbeit gegeben. Hierbei wird der Inhalt der folgenden fünf Kapitel kurz beschrieben:

Im Kapitel 2 werden die allgemeinen Grundlagen für das Verständnis dieser Arbeit beschrieben. Diesbezüglich werden die Grundlagen von Datenbanken beschrieben, wobei das Relationale Modell, Anfragesprachen und Abhängigkeiten betrachtet werden. Insbesondere die Beschreibungen der SQL-Anfragen und der Abhängigkeiten (S-t-tgd, Tgds und Egds) besitzen eine hohe Relevanz für das Verständnis dieser Arbeit. Ebenfalls wird in diesem Kapitel die unterschiedlichen Arten von Inversen betrachtet, wobei eine Unterscheidung von der Fagin-Inverse, der Quasi-Inverse und der CHASE-Inversen durchgeführt wird.

Im darauffolgenden Kapitel 3 wird der aktuelle Stand der Technik vorgestellt. Infolgedessen wird ein Überblick über das Tool ChaTEAU gegeben, welches für die Ausführung des CHASE-Algorithmus dient. Dementsprechend wird hierbei der CHASE-Algorithmus beschrieben, der in ProSA für die Ausführung der Anfragen dient. Ebenfalls wird in diesem Kapitel ein Überblick über ProSA gegeben, wobei die Invertierung und der Ablauf in ProSA beschrieben wird. Des weiterem werden im Kapitel 3 zwei Ansätze für die Invertierung von Anfragen bzw. von Abhängigkeiten aufgeführt.

Im Kapitel 4 wird das eigentliche Konzept dieser Arbeit vorgestellt. Diesbezüglich erfolgt eine Betrachtung der SQL-Anfragen und dessen Darstellung als Menge von Abhängigkeiten. Dabei wird die Konvertierung und Invertierung der SQL-Anfragen aufgelistet und die Vollständigkeit dieser Auflistung geprüft. Des weiterem wird ein eigener Algorithmus für die Invertierung von Anfragen bzw. von Abhängigkeiten entwickelt. Diesbezüglich erfolgt ein Vergleich der beiden Ansätze für die Invertierung, die im Kapitel 3 vorgestellt worden sind. Anschließend wird einer dieser Ansätze als Vorlage für den eigenen Invertierungalgorithmus verwendet.

Das fünfte Kapitel umfasst die Implementierung und Umsetzung des Invertierungsalgorithmus, der in Kapitel 4 beschrieben wurde. Infolgedessen wird die Umsetzung von jedem einzelnen Schritt des Algorithmus für die Invertierung betrachtet.

Schließlich wird im letzten Kapitel (Kapitel 6) eine kurze Zusammenfassung dieser Arbeit gegeben. Hierbei werden Ergebnisse, mögliche Erweiterungen und offene Problemstellungen aufgeführt.

2 Grundlagen

In ProSA wird der Anwendungsfall betrachtet, dass Anfragen auf eine Datenbankinstanz angewendet werden, um ein Anfrageergebnis zu erzeugen. Anschließend wird eine invertierte Anfrage auf das Anfrageergebnis angewendet, um die minimale Teildatenbank zu berechnen. Folglich ist das Ziel von ProSA die Berechnung der minimalen Teildatenbank, die für die Rekonstruktion des Anfrageergebnisse dient. Diesbezüglich wird für die Betrachtung von ProSA bzw. für diesem Anwendungsfall die Grundlagen für (relationale) Datenbanken benötigt. Diese Grundlagen werden in Abschnitt 2.1 aufgeführt. Hierbei wird das relationale Modell und Anfragen auf eine Datenbank beschrieben. In ProSA werden die Anfragen in Form von Abhängigkeiten (S-t-tgd, Tgd und Egds) auf die Datenbank bzw. einer Datenbankinstanz angewendet. Dementsprechend werden diese Abhängigkeiten im Abschnitt 2.1.3 eingeführt und beschrieben. Des weiterem besteht der Schwerpunkt dieser Arbeit auf die Invertierung der Anfragen. Diesbezüglich werden im Abschnitt 2.2 die Inversen eingeführt und unterschiedlichen Arten von Inversen bzw. invertierten Anfragen betrachtet.

2.1 Datenbanken

In diesem Abschnitt werden die Grundlagen von relationalen Datenbanken beschrieben. Eine Datenbank dient für die Verwaltung und Speicherung von Daten. Eine allgemeine Definition einer Datenbank wird im Folgendem aufgeführt:

Definition 2.1 (Datenbank, [Sch96])**.** Eine Datenbank ist eine Ansammlung von Daten, die untereinander in Beziehung miteinander stehen und von einem Datenbankverwaltungssystem (*Database Management System*, DBMS) verwaltet werden.

2.1.1 Relationales Modell

Das relationale Datenbankmodell wurde 1970 von E.F.Codd [Cod70] eingeführt und ist bis heute das am meisten verwendete Modell für Datenbanken. Die Grundlagen dieses Modells bilden Tabellen, auch als Relationen bezeichnet. Dementsprechend besteht eine relationale Datenbank aus mehreren Tabellen/Relationen, die alle einen eindeutigen Namen (Relationsnamen) besitzen. Eine Relation wird von einem Relationsschemata abgeleitet, dass aus einer Menge von unterschiedlichen Attributen (Spaltennamen) besteht. Hierbei wird jedem Attribut einen konkreten Wertebereich (auch Domäne genannt) zugeordnet. Diese Wertebereiche bestehen aus atomaren Werten, beispielsweise Integer,

Strings und boolesche Werte. Im Folgenden wird eine genaue Definition von einem Relationsschemata gegeben.

Definition 2.2 (Relationsschemata, [HSS18])**.** Ein Relationsschemata R ist eine Menge von Attributen $\{A_1, ..., A_n\}$, für ein $n \in \mathbb{N}^+$. Für jedes A_i existiert eine nichtleere endliche Menge D_i, die die Domäne von A_i definiert. Die Domäne von A_i wird im Weiterem als $dom(A_i)$ bezeichnet.

Eine Relation wird als Menge von Abbildungen auf einem Relationsschemata, folgendermaßen definiert:

Definition 2.3 (Relation, [HSS18])**.** Eine Relation $r(R)$ über dem Relationsschemata R, ist eine endliche Menge von Abbildungen:

$$t : R \rightarrow \bigcup_{i=1}^{n} D_i.$$

Die Abbildungen t werden als Tupel bezeichnet, wobei gilt $t(A_i) \in dom(A_i)$ (für alle $1 \leq i \leq n$, mit $n \in \mathbb{N}^+$)

Demzufolge besteht eine Relation aus Attributen (Spalten bzw. Spaltennamen), Tupeln (Zeilen der Tabelle) und einen Relationsname. Für die Veranschaulichung stellt Abbildung 2.1, den Aufbau einer Relation noch einmal bildlich dar.

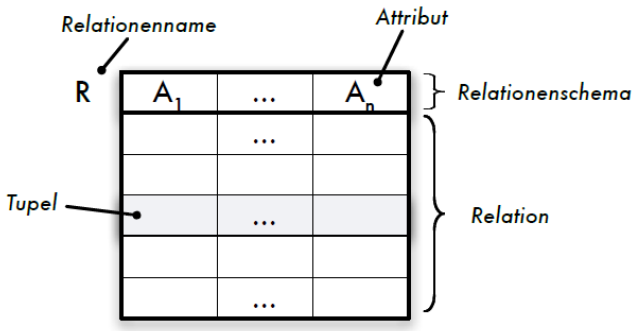

Abbildung 2.1: Aufbau einer Relation [HSS18]

Beispiel. Ein Beispiel einer Relation stellt die Relation *Students* in Tabelle 1.1 dar, die Informationen über Studierende an einer Universität speichert. Folglich basiert die Relation *Students* auf folgenden Relationsschemata:

$$Students = \{ID_{Students}, MatricNo, LastName, FirstName, StudyCourse\}$$

Die Tupel t der Students-Relation repräsentieren jeweils einen Studierenden an der Universität, beispielsweise:

$$t_1(ID) = S_1,$$
$$t_1(MatricNo) = 1,$$
$$t_1(LastName) = \text{Fieber},$$
$$t_1(FirstName) = \text{Fabian},$$
$$t_1(StudyCourse) = \text{Computer Science for Lecturship}.$$

Abschließend wird eine Datenbank basierend auf einem Datenbankschema, folgendermaßen definiert:

Definition 2.4 (Datenbankschema)**.** Ein Datenbankschema $S = \{R_1, ..., R_n\}$ ist eine Menge von Relationsschemata $\{R_1, ..., R_n\}$, für ein $n \in \mathbb{N}^+$.

Definition 2.5 (Datenbank)**.** Eine Datenbank $d = \{r_1, ..., r_n\}$ ist eine Menge von Relationen $\{r_1, ..., r_n\}$ über ein Datenbankschema $S = \{R_1, ..., R_n\}$ $(n \in \mathbb{N}^+)$, wobei $r_i(R_i)$ für alle $1 \leq i \leq n$ gilt.

2.1.2 Anfragesprachen

Eine Anfragesprache definiert im Bereich der Datenbanken Operationen, die für die Veränderung oder dem Suchen von Daten benötigt werden. Folglich ist eine Anfrage Q als Funktion definiert, die eine Datenbank d als Eingabe erhält und eine Relation r als Ausgabe liefert [GMS12]. Die Anfrage Q basiert dabei auf den Operationen der verwendeten Anfragesprache und dient somit zur Veränderung oder dem Suchen von Daten. In diesem Abschnitt werden die Anfragesprachen Relationale Algebra und SQL (*Structured Query Language*) für relationale Datenbanken betrachtet und eine Definition von dessen Operationen angegeben.

2.1.2.1 Relationale Algebra

Die Relationale Algebra, ebenfalls durch E.F.Codd [Cod70] eingeführt, bildet die Grundlage für die Anfragesprachen im Relationalen Modell. Im Folgendem werden die Operationen Projektion, Verbund, Selektion und Umbenennung der Relationalen Algebra beschrieben. Ebenfalls wird die Umsetzung klassischen Mengenoperationen Vereiningung, Differenz und Durchschnitt angegeben. Für die Beschreibung der Operationen werden die Definitionen, aus dem Buch: *Datenbanken - Konzepte und Sprachen* [HSS18] entnommen.

Projektion. Die Projektion π gibt an, welche Teilmenge der Attribute (Spalten) in der Ergebnisrelation erhalten bleiben sollen. Auf Grund dessen werden durch die Projektion Attribute bzw. Spalten ausgeblendet. Dementsprechend ist die Projektion folgendermaßen definiert:

Definition 2.6 (Projektion)**.** Bei einer Projektion π für eine Relation $r(R)$, auf einer Attributmenge $X \subseteq R$ in R, gilt:

$$\pi_X(r) = \{t(X) | t \in r\}.$$

Für die Veranschaulichung einer Projektion auf eine Relation, stellt die Tabelle 2.1 die Ergebnisrelation der Anfrage $\pi_{MatricNo,LastName}(Students)$ auf die Relation *Students* (Tabelle 1.1) dar. Demzufolge tauchen in der Ergebnisrelation ausschließlich die Attribute *MatricNo* und *LastName* auf, die restlichen Attribute der Relation *Students* werden ausgeblendet.

MatricNo	LastName
1	Fieber
2	Sonnenschein
3	Müller
4	Müller
5	Johansen
6	Miller
7	Mustermann
8	Johannes

Tabelle 2.1: Projektion-Beispiel

Selektion. Bei der Selektion σ werden Bedingungen definiert, die einzelne Attribute/Attributwerte erfüllen müssen, um in die Ergebnisrelation aufgenommen zu werden. Dabei wird bei der Selektion zwischen der Selektion mit Konstanten und der Selektion mit Attributen unterschieden.

Definition 2.7 (Selektion mit Konstante)**.** Bei einer Selektion σ mit Konstanten wird ein Vergleich $A\theta c$, eines Attributes $A \in R$ mit einer Konstante c, durchgeführt. Dabei ist $\theta \in \{=, \neq, \leq, <, \geq, >\}$ die Vergleichsoperation.

$$\sigma_{A\theta c}(r) = \{t | t \in r \wedge t(A)\theta c\}.$$

Definition 2.8 (Selektion mit Attribut)**.** Bei einer Selektion σ mit Attributen wird ein Vergleich $A_i\theta A_j$, zwischen zwei Attributen $A_i, A_j \in R$, durchgeführt (mit $\theta \in \{=, \neq, \leq, <, \geq, >\}$).

$$\sigma_{A_i\theta A_j}(r) = \{t | t \in r \wedge t(A_i)\theta t(A_j\}.$$

Die Tabelle 2.2 repräsentiert eine Ergebnisrelation, die aus der Anwendung einer Projektion und einer Selektion auf die Grades-Relation (Tabelle 1.5) resultiert. Hierbei wurde die Anfrage $\pi_{MatricNo,Grade}(\sigma_{Grade \leq 2}(Grades))$ durchgeführt. Dementsprechend enthält die Ergebnisrelation nur Tupel mit einer Note besser oder gleich 2.0 und die Attribute *MatricNo* und *Grade*.

MatricNo	Grade
2	1.3
4	1.0
5	1.3
6	2.0

Tabelle 2.2: Selektion-Beispiel

Verbund. Der (natürliche) Verbund $\bowtie$ dient zur Verknüpfung von zwei Relationen, zu einer Ergebnisrelation. Hierbei wird der Verbund über ein gemeinsames Attribut, in beiden Relationen, mit gleichen Attributwerten durchgeführt.

Definition 2.9 (natürlicher Verbund)**.** Beim natürliche Verbund $r_1 \bowtie r_2$, von zwei Relationen r_1 und r_2, gilt:

$$r_1 \bowtie r_2 = \{t | t(R_1 \cup R_2) \wedge \exists t_1 \in r_1 : t_1 = t(R_1) \wedge \exists t_2 \in r_2 : t_2 = t(R_2)\}.$$

Die Ergebnisrelation für den natürlichen Verbund der Relationen *Students* und *Grades*, ist in Tabelle 2.3 dargestellt. Dabei wird die Anfrage $\pi_{MatricNo,Lastname,Grade}(Students \bowtie Grades)$ durchgeführt, die einen Verbund über dem gemeinsamen Attribut *MatricNo* realisiert.

MatricNo	LastName	Grade
1	Fieber	3.7
2	Sonnenschein	1.3
3	Müller	2.3
4	Müller	1.0
5	Johansen	1.3
6	Miller	2.0
7	Mustermann	3.3

Tabelle 2.3: Verbund-Beispiel

Mengenoperationen. Die Relationale Algebra liefert die Möglichkeit die klassischen Mengenoperationen Vereinigung, Differenz und Durchschnitt auf die Relationen anzuwenden. Allerdings gilt die Bedingung, dass die Relationen das gleiche Relationsschemata besitzen. Die Mengenoperationen sind folgendermaßen definiert:

- Vereinigung: $r_1 \cup r_2 = \{t | t \in r_1 \vee t \in r_2\}$
- Differenz: $r_1 - r_2 = \{t | t \in r_1 \wedge t \notin r_2\}$
- Durchschnitt: $r_1 \cap r_2 = \{t | t \in r_1 \wedge t \in r_2\}$

Umbenennung. Die Umbenennung β dient für die Veränderung der Namen der Attribute in einer Relation. Dementsprechend kann die Umbenennung bei dem Verbund eingesetzt werden, um gemeinsame Attribute in den Relationen zu erzeugen. Ebenfalls kann eine Umbenennung der Attribute bei den Mengenoperationen erfolgen, um die Bedingung der gleichen Relationsschemata zu erfüllen. Die Umbenennung wird folgendermaßen beschrieben:

Definition 2.10 (Umbenennung)**.** Die Umbenennung $\beta_{B \leftarrow A}$ von Attribut A in Attribut B, in einer Relation r, wird folglich definiert:

$$\beta_{B \leftarrow A}(r) = \{t' | \exists t \in r : t'(R - A) = t(R - A) \wedge t'(B) = t(A)\}.$$

2.1.2.2 SQL

Die Anfragesprache SQL (*Structured Query Language*) basiert auf der Relationalen Algebra und dient der Manipulation und Anfrage von Daten, in einer relationalen Datenbank. Die Grundlage von SQL-Anfragen bilden SELECT-FROM-WHERE Ausdrücke, die die Operationen Projektion und Selektion von der Relationalen Algebra wiedergeben. Folglich stellt die SELECT-Klausel die Projektion dar, die auf der FROM-Klausel deklarierten Relation angewendet wird. Dagegen wird durch die WHERE-Klausel die Selektionsbedingung definiert, die ebenfalls auf die Relation der FROM-Klausel angewendet wird. Ein Beispiel einer SQL-Anfrage ist die Anfrage 2.1. Diese Anfrage lässt sich durch folgenden Ausdruck in der Relationalen Algebra definieren:

$$\pi_{MatricNo,LastName}(\sigma_{StudyCourse='ComputerScience'}(Students)).$$

Für die Veranschaulichung repräsentiert die Tabelle 2.4, die Ergebnisrelation der Anfrage 2.1. Diesbezüglich gibt die Anfrage 2.1 den Nachnamen und die Matrikelnummer aller Studenten (aus Tabelle 1.1) wieder, die Informatik (*Computer Sciene*) studieren.

```
SELECT MatricNo,LastName
FROM Students
WHERE StudyCourse = 'Computer Science'
```

Anfrage 2.1: SQL-Beispiel

MatricNo	LastName
4	Müller
5	Johansen
6	Miller

Tabelle 2.4: Ergebnisrelation von Anfrage 2.1

Des weiterem werden Verbunde in SQL durch die JOIN-Klausel umgesetzt. Ein Beispiel eines Verbundes von zwei Relationen veranschaulicht die Anfrage 2.2. Hierbei werden folgende Operationen, der Relationalen Algebra, durchgeführt:

$$\pi_{MatricNo,LastName,Grade}(\sigma_{Grade<3.0}(Students \bowtie Grades)).$$

Demzufolge wird ein natürlicher Verbund der Relationen *Students* und *Grades*, über dem gemeinsamen Attribut *MatricNo*, durchgeführt. Die Anfrage gibt als Ergebnis eine Relation wieder, die alle Studierende erhält (mit den Attributen *MatricNo*, *LastName*, *Grade*) die eine bessere Note als 3.0 erzielt haben.

```
SELECT MatricNo, LastName, Grade
FROM Students NATURAL JOIN Grades
WHERE Grade < 3.0
```

Anfrage 2.2: SQL-Verbund-Beispiel

Neben den grundlegenden Operationen (Projektion, Selektion und Verbund), bietet SQL die Möglichkeit Aggregatfunktionen zu verwenden. Diese Aggregatfunktionen dienen beispielsweise für die Berechnung des maximalen Wertes einer Spalte und werden innerhalb der SELECT-Klausel angegeben. Diesbezüglich werden folgende Funktionen von SQL definiert:

- MAX(b)/MIN(b): Bestimmt den maximalen bzw. minimalen Wert, der Spalte b.
- SUM(b): Berechnet die Summe aller Werte, in Spalte b.
- COUNT(b): Bestimmt die Anzahl der Tupel.
- AVG(b): Berechnet den Durchschnitt aller Werte, in Spalte b.

Für die Veranschaulichung stellt Anfrage 2.3, eine Anfrage mit der Aggregationsfunktion MIN dar. Infolgedessen wird durch die Anfrage 2.3, die beste Note (minimaler Wert von Spalte/Attribut *Grade*) im Modul 002 berechnet.

```
SELECT MIN(Grade)
FROM Grades
WHERE ModulNo = '002'
```

Anfrage 2.3: Aggregation-Beispiel

2.1.3 Abhängigkeiten

In diesem Abschnitt werden Abhängigkeiten von einer relationalen Datenbank beschrieben. Infolgedessen werde die Integritätsbedingungen Schlüssel, Funktionale Abhängigkeiten und Verbundabhängigkeiten betrachtet. Bei Integritätsbedingungen handelt es sich um Regeln, die die Datenbank bzw. einzelne Relationen dauerhaft erfüllen müssen, um beispielsweise die Konsistenz der Daten zu gewährleisten. Ein Beispiel einer Integritätsbedingung wäre die Regel, dass ausschließlich eindeutige Werte (kein Wert doppelt) in einer Spalte einer Relation auftreten darf (Schlüssel). Ebenfalls wird in diesem Abschnitt die von ProSA verwendeten Abhängigkeiten Tgd, S-t-tgd und Egds betrachtet.

Schlüssel. Ein Schlüssel (Schlüsselabhängigkeit) wird über eine identifizierende Attributmenge definiert, infolgedessen ist ein Schlüssel eine minimale identifizierende Attributmenge (bzgl. $\subseteq$). Jedes Attribut eines Schlüssels wird als Primärattribut bezeichnet und ein Primärschlüssel ist ein ausgezeichneter Schlüssel. Diesbezüglich ist eine identifizierende Attributmenge folglich definiert:

Definition 2.11 (Identifizierende Attributmenge, [HSS18])**.** Eine identifizierende Attributmenge K für eine Relation $r(R)$, ist eine Teilmenge $K \in \{B_1, ..., B_k\} \subseteq R$ über dem Relationsschemata R, sodass gilt:

$$\forall t_1, t_2 \in r[t_1 \neq t_2 \Rightarrow \exists B \in K : t_1(B) \neq t_2].$$

Funktionale Abhängigkeiten. Eine funktionale Abhängigkeit (FD) $X \rightarrow Y$ bedeutet, dass die Attributmenge Y durch die Attributmenge X (in einer Relation) bestimmt wird. Dementsprechend besitzen gleiche Werte in der Attributmenge X, in unterschiedlichen Tupeln, auch die gleichen Werte in der Attributmenge Y.

Definition 2.12 (Funktionale Abhängigkeit, [HSS18])**.** Eine Relation r(R) erfüllt die funktionale Abhängigkeit $X \rightarrow Y$, mit den Attributmengen $X, Y \subseteq R$, wenn gilt:

$$|\pi_Y(\sigma_{X=x}(r))| \leq 1, \forall x \in X.$$

Verbundabhängigkeiten. Eine Verbundabhängigkeit definiert die Bedingung, dass bei einer Zerlegung einer Relation, die Relation durch einen Verbund (bzw. mehrere Verbunde) wiederhergestellt werden kann.

Definition 2.13 (Verbundabhängigkeit, [HSS18])**.** Eine Verbundabhängigkeit $JD \bowtie [R_1, ..., R_n]$ für ein Datenbankschema $S = R_1, ..., R_n$, wird von einer Relation $r(R)$ erfüllt wenn, gilt:

$$r = \pi_{R_1}(r) \bowtie ... \bowtie \pi_{R_n}(r).$$

Im Folgendem werden die Abhängigkeiten S-t-tgds, Tgds und Egds definiert. An dieser Stelle wird der Begriff des atomaren Ausdrucks bzw. Atome verwendet und anhand dessen die Abhängigkeiten definiert. Definitionsgemäß ist ein Atom ein Ausdruck der Form $R(x_1, ..., x_n)$, über einer Relation/Relationsnamen R und $x_1, ..., x_n$ Variablen [FKPT11b].

Definition 2.14 (Tgd, [AH18b])**.** Eine Tgd (*Tuple-generating dependencie*) ist ein Ausdruck der folgenden Form, wobei $\phi(x)$ und $\psi(x, y)$ Konjunktionen von atomaren Ausdrücken, aus x und y, sind:

$$\forall x : (\phi(x) \rightarrow \exists y : \psi(x, y)).$$

Definition 2.15 (S-t-tgd, [AH18b])**.** Eine S-t-tgd (*Source-to-target tuple-generating dependency*) ist ein Ausdruck der folgender Form, wobei $\phi(x)$ eine Konjunktion von atomaren Ausdrücken über einen Quellschema S ist und $\psi(x, y)$ eine Konjunktion von atomaren Ausdrücken über ein Zielschema T darstellt:

$$\forall x : (\phi(x) \rightarrow \exists y : \psi(x, y)).$$

Definition 2.16 (Egd, [AH18b])**.** Eine Egd (*Equality-generating dependency*) ist für zwei Variablen x_1,$x_2 \in x$, ein Ausdruck der Form:

$$\forall x : (\phi(x) \rightarrow (x_1 = x_2)).$$

Bei den S-t-tgds, Tgds und Egds Abhängigkeiten wird der vordere Teil der Implikation $\phi(x)$ als Rumpf (*Body*) und der hintere Teil $\psi(x, y)$ bzw. $(x_1 = x_2)$ als Kopf (*Head*) deklariert. Hierbei stehen die existenzquantifizierte Variablen ausschließlich im Kopf der Abhängigkeit. Im Rahmen dieser Arbeit wird auf die Angabe des Allquantors verzichtet, um die Angabe der Abhängigkeiten zu vereinfachen. Entsprechend werden im Folgenden alle Variablen die nicht existenzquantifiziert sind, als allquantifiziert angenommen.

In diesem Abschnitt wurden die unterschiedlichen Abhängigkeiten definiert. Im folgenden Abschnitt wird die Invertierung dieser Abhängigkeiten (S-t-tgds, Tgds und Egds) betrachtet. Hierbei werden unterschiedliche Arten von der Invertierung der Abhängigkeiten definiert und beschrieben.

2.2 Inversen

In diesem Abschnitt wird ein Überblick über das Prinzip der inversen Schemaabbildungen gegeben. Im Grundprinzip dient eine Inverse von einer Schemaabbildung, um eine Ausführung bzw. dessen Effekt einer Schemaabbildung möglichst wieder rückgängig zu machen [Fag07]. Diese Prinzip kommt beispielsweise in den Bereichen des Datenaustausch, der Datenintegration und der Schemaevolution zum Einsatz. Folglich wird in

diesem Abschnitt die Definition einer Inverse und einer Quasi-Inverse, nach R.Fagin [Fag07, FKPT08], gegeben. Diesbezüglich wird folgende Definition einer Schemaabbildung verwendet:

Definition 2.17 (Schemaabbildung,[FKPT11b])**.** Eine Schemaabbildung ist ein Tripel der Form $\mathcal{M} = (S, T, \Sigma)$, mit einem Quellschema S, einem Zielschema T und einer Menge von Abhängigkeiten Σ, die die Beziehungen zwischen S und T beschreiben. Des weiterem ist die Schemaabbildung $\mathcal{M}$, durch folgende binäre Relation definiert:

$$Inst(\mathcal{M}) = \{(I, J)|(I, J) \models \Sigma\}.$$

Infolgedessen entspricht I einer Instanz über S und J ist eine Instanz über T. Außerdem erfüllt das Paar $(I, J) \in \mathcal{M}$ die Abhängigkeiten in Σ ($(I, J) \models \Sigma$) und dementsprechend die Bedingungen in $\mathcal{M}$. Demnach wird die Instanz J als Ergebnis von der Instanz I bezeichnet, somit ist das Ergebnis $Sol_{\mathcal{M}}(I)$ (von Instanz I)[Pér13] definiert als:

$$Sol_{\mathcal{M}}(I) = \{J|(I, J) \in \mathcal{M}\}.$$

2.2.1 Inverse nach Fagin

Das Prinzip der Inversen von Schemaabbildungen wurde von R. Fagin [Fag07], im Jahr 2007, eingeführt. Demnach wurde eine Inverse $\mathcal{M}'$, von einer Schemaabbildung $\mathcal{M}$, über einer Komposition $\mathcal{M} \circ \mathcal{M}' = Id$ definiert. Die Identität Id entspricht, dabei die Abbildung eines Schemas auf sich selbst.

Definition 2.18 (Komposition, [FKPT05])**.** Gegeben seien die Schemaabbildungen $\mathcal{M}_{12} = (S_1, S_2, \Sigma_{12})$ und $\mathcal{M}_{23} = (S_2, S_3, \Sigma_{23})$, wobei I_n eine Instanz über S_n sei (mit $n \in \{1, 2, 3\}$). Diesbezüglich ist eine Komposition $M_{13} = M_{12} \circ M_{23}$ folglich definiert:

$$M_{12} \circ M_{23} = \{(I_1, I_3)|\exists I_2 : (I_1, I_2) \in \mathcal{M}_{12} \wedge (I_2, I_3) \in \mathcal{M}_{23}\}.$$

Definition 2.19 (Inverse nach Fagin, [Fag07])**.** Seien die Schemaabbildungen $\mathcal{M}_{12} = (S_1, S_2, \Sigma_{12})$ und $\mathcal{M}_{21} = (S_2, \hat{S}_1, \Sigma_{21})$ gegeben, wobei $\hat{S}_1$ als eine Kopie von S_1 interpretiert wird. Ebenfalls ist die Komposition $\sigma = \Sigma_{12} \circ \Sigma_{21}$ von $\mathcal{M}_{12}$ und $\mathcal{M}_{21}$ gegeben und sei $\mathcal{M}_{11} = (S_1, \hat{S}_1, \sigma)$. Hierbei entspricht I die Instanz über S_1 und $\hat{I}$ ist die Instanz über $\hat{S}_1$. Folglich ist $\mathcal{M}_{12}$ eine Inverse von $\mathcal{M}_{21}$ für I, wenn $\mathcal{M}_{11}$ und die identifizierende Schemaabbildung $\mathcal{M}_{id} = (S_1, \hat{S}_1, \Sigma_{id})$ äquivalent bezüglich I sind. Dementsprechend ist $\mathcal{M}_{21}$ eine Inverse von $\mathcal{M}_{12}$ für I, wenn für jedes J (Ergebnisinstanz von I unter der identifizierende Schemaabbildung $\mathcal{M}_{id}$) gilt:

$$(I, J) \models \sigma, \text{genau dann, wenn } \hat{I} \subseteq J.$$

An dieser Stelle sei anzuführen, dass es sich bei den Abhängigkeiten Σ in einer Schemaabildung $\mathcal{M}$, um S-t-tgds bzw. Tgds Abhängigkeiten (Abschnitt 2.1.3) handelt. Folglich entsprechen die Abhängigkeiten Σ_{id}, der identifizierende Schemaabbildung

$\mathcal{M}_{id} = (S_1, \hat{S}_1, \Sigma_{id})$, Tgds der Form:

$$\Sigma_{id} = \{R(x_1, ..., x_n) \rightarrow \hat{R}(x_1, ..., x_n) | \forall R \in S\}.$$

Bei diesen Tgds ist R eine Relationssymbol vom Schema S, mit $x_1, ..., x_n$ Variablen und $\hat{R}$ ein Relationssymbol von Schema $\hat{S}$. Demnach wird durch die Abhängigkeit Σ_{id} eine Kopie/Abbildung von S auf $\hat{S}$ realisiert. Dementsprechend wird eine Identität $\overline{Id}_S$, also eine Abbildung von S nach S, folglich definiert:

Definition 2.20 (Identität, [Pér13])**.** Sei eine Schemaabbildung $\mathcal{M}$ von S nach S gegeben, mit I, J-Instanzen über S, dann gilt:

$$\overline{Id}_S = \{(I, J) | I \subseteq J\}.$$

Diesbezüglich entspricht die Identität $\overline{Id}_S$ eine Definition der Identität, die auf der Verwendung von S-t-tdgs angepasst ist. Mithilfe dieser Definition der Identität lässt sich eine einfachere Definition von Inversen (nach Fagin) treffen.

Definition 2.21 (Inverse nach Fagin,[Pér13])**.** Sei $\mathcal{M}_{12} = (S_1, S_2, \Sigma_{12})$ eine Schemaabbildung, von S_1 nach S_2, und $\mathcal{M}_{21} = (S_2, S_1, \Sigma_{21})$, von S_2 nach S_1. Dann ist $\mathcal{M}_{21}$ eine Inverse von $\mathcal{M}_{12}$, wenn gilt:

$$\mathcal{M}_{12} \circ \mathcal{M}_{21} = \overline{Id}_{S_1}.$$

Beispiel. Für die Veranschaulichung des Prinzips einer Fagin-Inverse wird im Folgendem ein Beispiel, nach [Pér13], durchgeführt. Im Beispiel wird die Schemaabbildung $\mathcal{M}_{12} = (S_1, S_2, \Sigma_{12})$ betrachtet, wobei $S_1 = \{A(a_1, a_2)\}$ und $S_2 = \{B(b_1, b_2, b_3)\}$ gilt. Außerdem sei folgende S-t-tgd, als Abhängigkeit in Σ_{12}, gegeben:

$$A(x, y) \rightarrow B(x, x, y).$$

Daraufhin wird eine invertierte Schemaabbildung $\mathcal{M}_{21} = (S_2, S_1, \Sigma_{21})$ gebildet, indem die Pfeilrichtung der S-t-tgd umgekehrt wird. Entsprechend wird folgende invertierte S-t-tgd, in Σ_{21}, definiert:

$$B(x, x, y) \rightarrow A(x, y).$$

Hierbei seien die Instanzen I und J über dem Schemas S_{12} gegeben, mit $(I, J) \in \mathcal{M}_{12} \circ \mathcal{M}_{21}$. Ebenfalls existiert eine Instanz K vom Schema S_{21} mit der Eigenschaft $(I, K) \in \mathcal{M}_{12}$ und $(K, J) \in \mathcal{M}_{21}$. Beispielsweise sei die Instanz $I = \{A(1, 2)\}$ und die Instanz $K = \{B(1, 1, 2)\}$, dann ist die Instanz I Teilmenge von J bzw. $A(1, 2) \in J$. Folglich ist jede Instanz $I = \{A(a_1, a_2)\}$ eine Teilmenge von J, wobei $K = \{B(a_1, a_1, a_2)\}$ entspricht. Demnach sei eine Instanz L von S_21 gegeben, die $B(a_1, a_1, a_2)$ enthält genau dann, wenn die Instanz I das Relationssymbol $A(a_1, a_2)$ beinhaltet. Darauffolgend wird $(I, J) \in$

$\mathcal{M}_{12} \circ \mathcal{M}_{21}$, durch $(I, L) \in \mathcal{M}_{12}$ und $(L, J) \in \mathcal{M}_{21}$, impliziert. Dementsprechend wurde bewiesen, dass $\mathcal{M}_{12} \circ \mathcal{M}_{21} = \overline{Id}_{S_1}$, durch $(I, J) \in \mathcal{M}_{12} \circ \mathcal{M}_{21}$, genau dann wenn $I \subseteq J$ gilt, impliziert wird [Pér13]. Die Schemaabbildung $\mathcal{M}_{21}$ ist somit eine Fagin-Inverse, von der Schemaabbildung $\mathcal{M}_{12}$.
Eine Fagin-Inverse ist eine Invertierung einer Schemaabbildung ohne eine Verlust von Informationen zu generieren. Für viele Schemaabbildung existiert keine Fagin-Inverse und somit keine verlustfreie Invertierung. Beispielsweise sei folgende S-t-tgd gegeben:

$$St(ln, fn) \rightarrow R(ln).$$

Hierbei wird die Relation *Students*, mit dem Attributen *LastName* und *FirstName*, auf die Relation R, mit dem Attribut *LastName*, abgebildet. Bei der Invertierung dieser S-t-tgd würde die Information des Vornamens (*FirstName*) verloren gehen. Diesbezüglich wäre die invertierte S-t-tgd folglich definiert:

$$R(ln) \rightarrow \exists fn : St(ln, fn).$$

Folglich entspricht diese Invertierung keiner Fagin-Inverse. Um die Existenz einer Fagin-Inverse formal zu beweisen, muss die dazugehörige S-t-tgd die Bedingung der sogenannten *unique-solutions property* erfüllen. Diese Bedingung ist folgendermaßen definiert:

Definition 2.22 (Unique-solution property, [Pér13])**.** Sei eine Schemaabbildung $\mathcal{M}$, von S_{12} nach S_{21}, gegeben. Dann erfüllt $\mathcal{M}$ die *unique-solution property*, wenn durch jedes Paar von Instanzen I_1 und I_2 von S_{12}, dass $Sol_{\mathcal{M}}(I_1) = Sol_{\mathcal{M}}(I_2)$ erfüllt, $I_1 = I_2$ impliziert wird.

2.2.2 Quasi-Inverse

Die Quasi-Inverse ist eine Abschwächung der Fagin-Inverse, wobei ein Informationsverlust auftreten kann. Dementsprechend basiert das Prinzip der Quasi-Inverse [FKPT08], eingeführt von Fagin im Jahr 2007, auf den Definitionen einer Inverse nach Fagin. Allerdings wird bei der Quasi-Inverse eine Äquivalentsrelation $\backsim_{\mathcal{M}}$ eingeführt. Bei der $I_1 \backsim_{\mathcal{M}} I_2$ erfüllt wird, wenn die Instanzen I_1 und I_2, über der Schemaabbildung $\mathcal{M}$, die gleiche Größe von Ergebnissen besitzen, also $Sol_{\mathcal{M}}(I_1) = Sol_{\mathcal{M}}(I_2)$. Demzufolge ist für eine Abbildung D, von S nach S, und eine Schemaabbildung $\mathcal{M}$ von S nach T, eine Äquivalentsrelation $D_{\backsim_{\mathcal{M}}}$ folglich definiert [Pér13]:

$$D_{\backsim_{\mathcal{M}}} = \{(I_1, I_2) | \exists I_1', I_2' : I_1 \backsim_{\mathcal{M}} I_1', I_2 \backsim_{\mathcal{M}} I_2' \wedge (I_1', I_2') \in D\}.$$

Mithilfe dieser Äquivalentsrelation lässt sich eine Quasi-Inverse definieren.

Definition 2.23 (Quasi-Inverse,[Pér13])**.** Sei die Schemaabbildung $\mathcal{M}_{21} = (S_2, S_1, \Sigma_{21})$ eine Quasi-Inverse von $\mathcal{M}_{12} = (S_1, S_2, \Sigma_{12})$, dann gilt:

$$(\mathcal{M}_{12} \circ \mathcal{M}_{21})_{[\backsim_M]} = Id_{S_1[\backsim_M]}).$$

Beispiel. Als Beispiel einer Quasi-Inverse wird das obige Beispiel wiederverwendet, mit der S-t-tgd:

$$St(ln, fn) \rightarrow R(ln).$$

Die Invertierung dieser S-t-tgd ist keine Fagin-Inverse, aufgrund des Informationsverlustes. Allerdings erfüllt die invertierte S-t-tgd die Bedingungen einer Quasi-Inverse.

2.2.3 CHASE-Inverse

Die CHASE-Inverse beschreibt eine Inverse für die Reproduzierbarkeit der verwendeten Originaldaten, mithilfe des CHASE-Algorithmus [FKPT11b]. Infolgedessen wird für die Schemaabbildung $\mathcal{M} = (S, S', \Sigma)$ eine invertierte Schemaabbildung $\mathcal{M}^* = (S', S, \Sigma')$ gebildet. Hierbei seien S und S' Schematas und Σ und Σ' seien Mengen von Abhängigkeiten. Exemplarisch stellt die Schemaabbildung $\mathcal{M}$ eine Schemaevolution, vom Schema S zum Schema S', dar. Diesbezüglich sorgt die Inverse $\mathcal{M}^*$ für die Reproduzierbarkeit der verwendeten Originaldaten, die zur Bildung des Schemas S' geführt haben, indem von Schema S' zum Schema S zurück abgebildet wird. Dabei wird der CHASE-Algorithmus genutzt, um die Quellinstanz I von S in die Zielinstanz I' von S' zu überführen. Dementsprechend ist die Zielinstanz I' durch $I' = chase_{\mathcal{M}}(I)$ definiert. Ebenfalls wird die Instanz I'', die durch die invertierte Schemaabbildung $\mathcal{M}^*$ entsteht, durch eine erneute Anwendung des CHASE generiert. Folglich ist die Instanz I'', durch $I'' = chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I))$ bzw. $I'' = chase_{\mathcal{M}^*}(I')$ definiert. An dieser Stelle sei anzuführen, dass bei der Anwendung einer CHASE-Inverse ein Informationsverlust möglich ist, indem die Inverse nicht alle verwendeten Originaldaten rekonstruieren kann. In diesem Fall werden durch den CHASE Nullwerte eingeführt, als Platzhalter für die nicht vorhandenen Informationen.
Im Folgendem werden unterschiedliche Arten von CHASE-Inversen betrachtet und anhand von Beispielen, ähnlich wie in [Aug17], dargestellt. Einerseits werden die von Fagin [FKPT11b] eingeführten CHASE-Inversen Arten, exakte CHASE-Invere, klassische CHASE-Invere und relaxte CHASE-Inverse, betrachtet. Anderseits werden zwei weitere CHASE-Inversen Arten (von [AH18c]), die ergebnisäquvialente CHASE-Inverse und tupelerhaltende relaxte CHASE-Inverse, beschrieben. Dabei wird bei der Definition der Inversen, der Bergriff der GLAV-Schemaabbildung verwendet. Eine GLAV (*global-and-local-as-view*) entspricht einer Kombination der Bedingungen von LAV (*local-as-view*) und GAV (*global-as-view*). Definitionsgemäß ist eine LAV eine S-t-tgd mit nur einem Atom im Rumpf und eine GAV ist eine S-t-tgd mit nur einem Atom im Kopf ohne existenzquantifizierte Variablen. Dagegen ist eine GLAV eine S-t-tgd, die eine Konjunktion von Atomen im Rumpf und im Kopf erhalten kann. Folglich entspricht eine GLAV die Definition einer S-t-tgd (Definition 2.15). Dementsprechend ist eine GLAV-Schemaabbildung eine Schemaabbildung mit S-t-tgds als Abhängigkeiten. An dieser Stelle sei anzuführen, dass im Rahmen dieser Arbeit der Begriff der GLAV bzw. der GLAV-Schema-

abbildung nur für die konkrete Definition von CHASE-Inversen, nach [FKPT11b], verwendet wird und im Allgemeinem als Synonym für eine S-t-tgd gilt.

Exakte CHASE-Inverse. Die exakte CHASE-Inverse ist eine Invertierung ohne Informationsverlust, wobei die Quellinstanz I wiederhergestellt wird und $I = I''$ gilt.

Definition 2.24 (Exakte CHASE-Inverse, [FKPT11b])**.** Sei eine GLAV-Schemaabbildung $\mathcal{M}$, vom Schema S zum Schema S' gegeben. Dann sei eine GLAV-Schemaabbildung $\mathcal{M}^*$, von S zu S', eine exakte CHASE-Inverse, wenn für jede Instanz I über S gilt:

$$I = chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I)).$$

Beispiel. Als Beispiel sei eine GLAV-Schemaabbildung $\mathcal{M}^* = (S', S, \Sigma')$ gegeben, wobei folgende Abhängigkeit in Σ' enthalten ist:

$$St(ma, ln, fn) \wedge G(ma, g) \rightarrow St'(ma, ln, fn).$$

In diesem Beispiel wurden die Relationen *Students* (kurz: St) und *Grades* (kurz: G) vereinfacht, um einen besseren Überblick zu gewährleisten. Hierbei sei die Schemaabbildung $\mathcal{M}^*$ eine exakte CHASE-Inverse, für die GLAV-Schemaabbildung $\mathcal{M} = (S, S', \Sigma)$ mit folgende Abhängigkeit in Σ:

$$St'(ma, ln, fn) \rightarrow \exists g : St(ma, ln, fn) \wedge G(ma, g).$$

Die Zielinstanz I wird durch die folgenden CHASE-Schritte wiederhergestellt:

$$\begin{aligned} chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I)) &= chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(St'(ma, ln, fn))) \\ &= chase_{\mathcal{M}^*}(St(ma, ln, fn), G(ma, g)) \\ &= St'(ma, ln, fn) \\ &= I. \end{aligned}$$

Klassische CHASE-Inverse. Im Gegensatz zur exakten CHASE-Inverse benötigt die klassische CHASE-Inverse keine Gleichheit zwischen der Quellinstanz I und der Instanz nach der Invertierung I''. Folglich muss die Quellinstanz nicht vollständig wiederhergestellt werden. Bei der klassischen CHASE-Inverse muss eine Äquivalenz, zwischen den beiden Instanzen I und I'', vorliegen. Eine Äquivalenz $I \leftrightarrow I''$ zwischen zwei Instanzen ist dann erfüllt, wenn zwei Homomorphismen $I \rightarrow I''$ und $I'' \rightarrow I$ existieren die in beide Richtungen abbilden. Aufgrund dessen, dass ein Homomorphismus die Nullwerte durch konkrete Werten ersetzen kann und Homomorphismen in beide Richtungen existieren, ist die klassischen CHASE-Inverse ebenfalls eine Invertierung ohne Informationsverlust.

Definition 2.25 (Klassische CHASE-Inverse, [FKPT11b])**.** Sei eine GLAV-Schemaabbildung $\mathcal{M}$, vom Schema S zum Schema S', gegeben. Dann sei eine GLAV-Schemaabbildung $\mathcal{M}^*$, von S zu S', eine klassische CHASE-Inverse, wenn für jede Instanz I über S gilt:

$$I \leftrightarrow chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I)).$$

Beispiel. Als Beispiel sei eine Schemaabbildung $\mathcal{M} = (S, S', \Sigma)$ gegeben, mit den folgender Abhängigkeit in Σ:

$$St(ln, fn) \rightarrow \exists z : R(ln, z) \wedge R(z, fn).$$

Für eine bessere Übersicht würde die Relation *Students* vereinfacht, somit enthält die Relation nur die Attribute *LastName* und *FirstName*. Ebenfalls wird ausschließlich die Quellinstanz $I = St(Johannes, Paul), St(Johansen, Johannes)$ betrachtet. Demnach ist eine invertierte Schemaabbildung $\mathcal{M}^* = (S', S, \Sigma')$ definiert, mit folgender invertierten Abhängigkeit (S-t-tgd) in Σ':

$$R(ln, z) \wedge R(z, fn) \rightarrow St(ln, fn).$$

Mit der invertierte Schemaabbildung $\mathcal{M}^*$ wird eine klassische CHASE-Inverse definiert, da gilt:

$$\begin{aligned} I'' &= chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I)) \\ &= chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(\{St(Johannes, Paul), St(Johansen, Johannes)\})) \\ &= chase_{\mathcal{M}^*}(\{R(Johannes, \eta_1), R(\eta_1, Paul), R(Johansen, \eta_2), R(\eta_2, Johannes)\} \\ &= I \cup St(\eta_1, \eta_2). \end{aligned}$$

Dementsprechend ist die Instanz $I'' = I \cup St(\eta_1, \eta_2)$ äquivalent zur Quellinstanz I, da beide Instanzen durch Homomorphismen aufeinander abgebildet werden können.

Relaxte CHASE-Inverse. In der relaxten CHASE-Inverse werden die Bedingungen einer Inverse weiter abgeschwächt, somit muss keine Äquvialenz zwischen den Instanzen I und I'' vorliegen. Sondern für die Bildung einer relaxten CHASE-Inverse muss eine Ergebnisäquivalenz, zwischen den Instanzen I und I'' vorliegen. Ergebnisäquivalenz bedeutet, dass die Ergebnisse bei einer erneuten Anwendung des CHASE-Algorithmus auf beide Instanzen äquivalent sind. Eine weitere Bedingung der relaxten CHASE-Inverse ist, dass ein Homomorphismus von der Instanz I'' auf die Quellinstanz I existieren muss.

Definition 2.26 (Ergebnisäquivalenz, [FKPT11b])**.** Sei eine GLAV-Schemaabbildung $\mathcal{M}$, vom Schema S zum Schema S', gegeben. Seien die Instanzen I und I'' über dem Schema S gegeben. Dann sind I und I'' ergebnisäquivalent, wenn gilt:

$$chase_{\mathcal{M}}(I) \leftrightarrow chase_{\mathcal{M}}(I'').$$

In Kurzschreibweise: $I \leftrightarrow_{\mathcal{M}} I''$.

Definition 2.27 (Relaxte CHASE-Inverse, [FKPT11b])**.** Sei eine GLAV-Schemaabbildung $\mathcal{M}$, vom Schema S zum Schema S', gegeben. Dann sei eine GLAV-Schemaabbildung $\mathcal{M}^*$, von S zu S', eine relaxte CHASE-Inverse, wenn für jede Instanz I über S und für die Instanz $I'' = chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I))$ gilt:

$$I'' \leftrightarrow_{\mathcal{M}} I,$$
$$I'' \rightarrow I.$$

Beispiel. Als Beispiel sei die GLAV-Schemaabbildung $\mathcal{M} = (S, S', \Sigma)$ gegeben, wobei die folgende Abhängigkeit angewendet wird:

$$St(ln, fn, sc) \rightarrow \exists ma : St'(ma, ln) \wedge C(ma, sc).$$

Diese Abhängigkeit realisiert eine Aufteilung der Relation *Students*, in zwei Relationen *Students* und *Courses* (kurz: C), wobei ein Attribut *MatricNo* (kurz: ma) hinzugefügt wird. Im Beispiel wird die Relation *Students* wieder vereinfacht, demnach enthält die Relation die Attribute *LastName*, *FirstName* und *StudyCourse*. Ebenfalls wird nur die Instanz $I = \{St(Mueller, Mira, ComputerScience)\}$ betrachtet. Eine invertierte Schemaabbildung $\mathcal{M}^*$ wird über folgende invertierte Abhängigkeit definiert:

$$St'(ma, ln) \wedge C(ma, sc) \rightarrow \exists fn : St(ln, fn, sc).$$

Folglich gilt:

$$\begin{aligned} I'' &= chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I)) \\ &= chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(\{St(Mueller, Mira, ComputerScience)\})) \\ &= chase_{\mathcal{M}^*}(\{St'(\eta_1, Mueller), G(\eta_1, ComputerScience)\} \\ &= \{St(Mueller, \eta_1, ComputerScience)\}. \end{aligned}$$

Bei dieser Invertierung existiert keine Äquivalenz zwischen den Instanzen I und I'', da kein Homomorphismus $I \rightarrow I''$ realisiert werden kann. Allerdings existiert ein Homomorphismus $I'' \rightarrow I$ und die Bedingung der Ergebnisäquivalenz ist erfüllt, da die erneute Anwendung des CHASE (unter $\mathcal{M}$) auf die Instanzen I und I'' das gleiche Ergebnis liefert. Dementsprechend ist die Schemaabbildung $\mathcal{M}^*$ eine relaxte CHASE-Inverse von $\mathcal{M}$. An dieser Stelle sei anzumerken, dass diese Invertierung ein Informationsverlust zur Folge hat, da die Vornamen der Studierende bei der Invertierung verloren gehen.

Ergebnisäquivalente CHASE-Inverse. Die ergebnisäquivalente CHASE-Inverse ist eine Abschwächung der relaxten CHASE-Inverse. Dementsprechend muss bei der ergebnisäquivalente CHASE-Inverse nur die Bedingung der Ergebnisäquivalenz erfüllt sein. Folglich muss kein Homomorphismus von I nach I'' existieren.

Definition 2.28 (Ergebnisäquivalente CHASE-Inverse, [AH18c])**.** Sei eine GLAV-Schemaabbildung $\mathcal{M}$, vom Schema S zum Schema S', gegeben. Dann sei eine GLAV-Schemaabbildung $\mathcal{M}^*$, von S zu S', eine Ergebnisäquivalente CHASE-Inverse, wenn für jede Instanz I über S und für die Instanz $I'' = chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I))$ gilt:

$$I'' \leftrightarrow_{\mathcal{M}} I.$$

Tupelerhaltende relaxte CHASE-Inverse. In der tupelerhaltende relaxte CHASE-Inverse soll die Anzahl der Tupeln von der originalen Datenbank erhalten bleiben. Diesbezüglich wird die relaxte CHASE-Inverse um die Bedingung erweitert, dass die Instanzen I und I'' die gleiche Anzahl von Tupeln besitzen soll.

Definition 2.29 (Tupelerhaltende relaxte CHASE-Inverse, [AH18c])**.** Sei eine GLAV-Schemaabbildung $\mathcal{M}$, vom Schema S zum Schema S', gegeben. Dann sei eine GLAV-Schemaabbildung $\mathcal{M}^*$, von S zu S', eine tupelerhaltende relaxte CHASE-Inverse, wenn für jede Instanz I über S und für die Instanz $I'' = chase_{\mathcal{M}^*}(chase_{\mathcal{M}}(I))$ gilt:

$$\begin{aligned} I'' &\leftrightarrow_{\mathcal{M}} I, \\ I'' &\rightarrow I, \\ |I| &= |I''|. \end{aligned}$$

Die unterschiedlichen Inversentypen lassen sich folgendermaßen, anhand ihrer Bedingungen, hierarchisch einordnen [AH18c]:

$$\text{ergebnisäquivalent} \preceq \text{relaxte} \preceq \text{tupelerhaltende relaxte} \preceq \text{exakte}.$$

Hierbei entspricht die Operation $\preceq$ eine Reduktion von einer Instanz auf einer anderen Instanz. Demnach existiert eine Reduktion $I'' \preceq I$, von der Instanz I auf die Instanz I'', wenn die Umsetzung eines Homomorphismus von I'' auf I möglich ist. Folglich ist die ergebnisäquivalente CHASE-Inverse die Inverse mit der ßchwächsten"Bedingung, auf die die anderen Typen reduziert werden können. In Tabelle 2.5 werden die Bedingungen und Zusammenhänge der Inversentypen noch einmal zusammengefasst.

In vorherigen Abschnitten wurden die allgemeinen Grundlagen für diese Arbeit beschrieben. Hierbei wurden relationale Datenbanken und unterschiedliche Arten von Inversen betrachtet. Im folgenden Kapitel werden diese Grundlagen verwendet, um den aktuellen Stand der Technik zu beschreiben. Dabei umfasst der Stand der Technik die verwendeten Werkzeuge und die Invertierungsmethoden für den weiteren Verlauf dieser Arbeit.

CHASE-Inverse	Notwendige Bedingung	Hinreichende Bedingung
Exakte	-	$I'' = I$
Klassische	Exakte CHASE-Inverse	$I'' \leftrightarrow I$
Tupelerhaltende	Exakte CHASE-Inverse	$I'' \preceq I$, $\lvert I''\rvert = \lvert I\rvert$
Relaxte	Tupelerhaltende CHASE-Inverse	$I'' \preceq I$
Ergebnisäquivalente	Relaxte CHASE-Inverse	$I'' \leftrightarrow_{\mathcal{M}} I$

Tabelle 2.5: Zusammenfassung Bedingungen der CHASE-Inversen Typen [AH18c]

3 Stand der Technik

In diesem Kapitel wird ein Überblick über die Werkzeuge ProSA und ChaTEAU gegeben. Hierbei wird im Abschnitt 3.1 das Werkzeug ChaTEAU vorgestellt, welches für die Umsetzung des CHASE-Algorithmus dient. Diesbezüglich wird der allgemeine CHASE-Algorithmus (Abschnitt 3.1.1) beschrieben, der in ProSA für die Anwendungen der Anfrage und der invertierten Anfrage auf die Datenbankinstanz bzw. auf das Anfrageergebnis verwendet wird. Anschließend werden im Abschnitt 3.2 die einzelnen Bestandteile von ProSA betrachtet und die Invertierung in ProSA beschrieben. Des weiterem werden in diesem Kapitel zwei Methoden für die Invertierung von Anfragen bzw. Abhängigkeiten vorgestellt (Abschnitt 3.3.3 und 3.4).

3.1 ChaTEAU

Das Tool ChaTEAU (*Chase for Transforming, Evolving, and Adapting databases and queries, Universal approach*) [AH19] ist ein universelles Werkzeug für die Anwendung des CHASE-Algroithmus. Diese Werkzeug wurde an der Universität Rostock entwickelt, wobei eine erste Umsetzung in [Jur18] gegeben ist. Diese erste Umsetzung wurde dann im Laufe von weiteren studentischen Arbeiten erweitert, beispielsweise in [Zim20, Gö20]. Eigentlich existieren viele Werkzeuge die eine Anwendung des CHASE realisieren, allerdings beschränken diese Werkzeuge sich auf eine oder zwei Varianten (Tabelle 3.1) des CHASE. Ein Überblick und Vergleich dieser Werkzeuge ist in [BKM$^+$17] gegeben. Die Grundidee von ChaTEAU ist die Konstruktion eines Werkzeugs, dass alle Varianten des CHASE beinhaltet. In den folgenden Abschnitten wird die Funktionsweise des CHASE-Algorithmus (Abschnitt 3.1.1) beschrieben und ein Überblick über ChaTEAU (Abschnitt 3.1.1) gegeben.

3.1.1 CHASE-Allgemein

Der CHASE-Algorithmus ist im Allgemeinem ein universeller Algorithmus für die Einarbeitung eines Parameters $\star$ in ein Objekt $\bigcirc$, um als Ergebnis ein Objekt zu erhalten, das den Parameter impliziert. Folglich wird durch den CHASE ein Ergebnis $\circledast$ erzeugt, wobei gilt:

$$chase_{\star}(\bigcirc) = \circledast.$$

Eine typische Anwendung für den CHASE ist die Einarbeitung von Abhängigkeiten als Parameter, in Form von FDs und JDs, in einem Datenbankschema als Objekt. Als

Ergebnis wird ein Datenbankschema erzeugt, welches die Abhängigkeiten/Integritätsbedingungen erfüllt. Diese Anwendung entspricht der ersten Grundidee vom CHASE-Algorithmus, die im Jahr 1979 entwickelt wurde [MMS79, ABU79]. Heutzutage existieren viele verschiedene Anwendungen und Varianten für den CHASE, die unterschiedlichen Varianten sind in Tabelle 3.1 veranschaulicht. Demnach ist die erste Grundidee des CHASE im Fall 0. der Tabelle 3.1 beschrieben. Ein weiteres Beispiel ist die Anwendung von Abhängigkeiten, in Form von S-t-tgds, Tgds und Egds, auf eine Datenbank, um ein Anfrageergebnis zu generieren (Fall VI). Dieser Fall beschreibt die Anwendung des CHASE-Algorithmus im Tool ProSA. Hierbei werden zwei CHASE-Phasen durchgeführt, wobei die erste Phase die Abhängigkeiten auf die Datenbank ausführt, um ein Anfrageergebnis zu erzeugen. Anschließend erfolgt die zweite Phase, indem die invertierten Abhängigkeiten bzw. die invertierte Anfrage auf das Anfrageergebnis angewendet werden, um eine minimale Teildatenbank zu berechnen.

	Parameter $\star$	Objekt $\bigcirc$	Ergebnis $\circledast$	Ziel
0.	Abhängigkeiten	Datenbankschema	Datenbankschema mit Integritätsbedingung	Optimieren des Datenbankentwurfs
I.	Abhängigkeiten	Anfrage	Anfrage	Semantische Optimierung
II.	Sicht (View)	Anfrage	Anfrage die Sicht verwendet	AQuV
II'.	Operation	Anfrage	Anfrage die Operation verwendet	AQuO
III.	S-t-tgd, Egd	Quelldatenbank	Zieldatenbank	Datenaustausch, Datenintegration
IV.	Tgd, Egd	Datenbank	modifiziert Datenbank	Cleaning
V.	Tgd, Egd	unvollständige Datenbank	Anfrageergebnis	Sichere Antworten
VI.	S-t-tgd, Tgd, Egd	Datenbank	Anfrageergebnis	Invertierte Anfrage ausführen

Tabelle 3.1: Varianten vom CHASE [AH19]

Im Algorithmus 1 wird die Umsetzung des Standard-CHASE skizziert, der eine Menge von Abhängigkeiten Σ in Form von S-t-tgds, Tgds und Egds in eine Datenbankinstanz I einarbeitet. Diesbezüglich liefert der Algorithmus eine Instanz I' als Ausgabe, welche alle Abhängigkeiten $\sigma \in \Sigma$ erfüllt. Als erstem Schritt wird nach einem aktiven Trigger h gesucht. Ein aktiver Trigger entspricht einem Homomorphismus, dessen linke Seite erfüllt ist und dessen rechte Seite noch zu erfüllen ist. Im Algorithmus wird jede Abhängigkeit σ, in Form von S-t-tgd, Tgd und Egd, als Homomorphismus interpretiert, dessen linke Seite von der Instanz I erfüllt wird. Ein aktiver Trigger ist vorhanden, wenn die Anwendung der rechten Seite der Abhängigkeit zu einer Veränderung der Instanz I führt. Die Anwendung der rechten Seite führt bei Tgds zu Erzeugung von neuen Tupeln und bei Egds zu der Erzsetzung von Nullwerten. In den nächsten Schritten wird ein Unterscheidung zwischen Tgd und Egd vorgenommen, und entsprechend neue Tupel hinzugefügt oder Nullwerte ersetzt. Schließlich terminiert der Algorithmus, wenn alle Abhängigkeiten $\sigma \in \Sigma$ bzw. alle aktive Trigger angewendet worden sind.

Algorithmus 1 : Allgemeiner CHASE-Algorithmus [AH19]

Input : Menge von Abhängigkeiten Σ, Datenbankinstanz I
Output : Veränderte Datenbankinstanz I'
foreach *Trigger h von einer Abhängigkeit $\sigma \in \Sigma$* **do**
  if *σ ist eine Tgd* **then**
    Hinzufügen der neuen Tupel zu der Instanz I
  else if *σ ist eine Egd* **then**
    if *Werte die verglichen werden sind unterschiedliche Konstanten* **then**
      CHASE scheitert
    else
      Ersetze die Nullwerte durch andere Nullwerte oder durch Konstanten

3.1.2 ChaTEAU-Allgemein

Die Funktionsweise von ChaTEAU entspricht der Umsetzung des Algorithmus 1, um eine allgemeine Anwendung des CHASE zu realisieren [AH19]. Entsprechend wird für alle Varianten des CHASE die gleiche Methode angewendet, wobei sich die Parameter $\star$ und das Objekt $\bigcirc$ des CHASE bei den Anwendungen unterscheiden können.
Diesbezüglich kann der Parameter $\star$ als Menge von Abhängigkeiten (S-t-tgds, Tgds und EGDs) oder in Form von Sichten vorliegen. Die Sichten als Parameter (Fall II in Tabelle 3.1) dienen der Umsetzung AQuV-Problems (*Answering Queries using Views*) [DH13], indem eine Anfrage nur mithilfe von vordefinierten Sichten (*Views*) definiert bzw. beantwortet soll. In ChaTEAU werden die Sichten als Abhängigkeiten interpretiert, um den CHASE anwenden zu können. Als Objekt $\bigcirc$ bestehen die Möglichkeiten eine Anfrage oder eine Datenbankinstanz zu verwenden. Hierbei wird bei einer Anfrage eine Ersetzung von Variablen durch andere Variablen oder Konstanten durchgeführt. Dagegen werden in einer Datenbankinstanz Nullwerte durch andere Nullwerte oder durch Konstanten ersetzt. Die Ausgabe $\circledast$ von ChaTEAU bzw. des CHASE ist abhängig von den gewählten Parameter $\star$ und des Objektes $\bigcirc$. Entsprechend liefert die Einarbeitung des Parameters in das Objekt, eine veränderte Datenbankinstanz, ein Anfrageergebnis oder ein verändertes Datenbankschema zurück.
Für die Implementierung von ChaTEAU werden die folgenden Hauptbestandteile benötigt: Terme, Homomorphismen, Abhängigkeiten, Atome und Instanzen [AH19]. Hierbei stellen die Abhängigkeiten (*Integrity Constraints*) S-t-tgds, Tgds und Egds dar, die aus einem Kopf und einem Rumpf bestehen. Diesbezüglich bestehen die Köpfe und Rümpfe der Abhängigkeiten aus einer Menge von Atomen. Diese Atome repräsentieren die Relationen im Kopf und im Rumpf der Abhängigkeiten, wobei diese wiederum aus einer Menge von Termen bestehen. In ChaTEAU können Terme Nullwerte, Variablen oder Konstanten entsprechen. Folglich bilden die Terme die Variablen bzw. Attributwerte in den S-t-tgds, Tgds und Egds. Des Weiterem bildet ein Homomorphismus eine Abbildung zwischen zwei Termen. Beispielsweise wird durch eine Homomorphismus eine Abbildung

von zwei Nullwerten $\eta_1 \rightarrow \eta_2$ realisiert, wobei der Nullwert η_1 durch den Nullwert η_2 ersetzt wird. Im CHASE-Algorithmus dienen die Homomorphismen als aktiver Trigger (Algorithmus 1), die die Bedingung für die Ausführung der Abhängigkeiten definieren.

In den vorherigen Abschnitten wurde das Werkzeug ChaTEAU, für die Umsetzung des CHASE-Algorithmus betrachtet. Dieses Werkzeug wird in ProSA verwendet, um das Anfrageergebnis und die minimale Teildatenbank zu generieren. Im folgenden Abschnitt wird der genaue Ablauf von ProSA beschrieben, der diese zwei Anwendungen von ChaTEAU beinhaltet. Ebenfalls wird die Verwendung der Inversen innerhalb von ProSA betrachtet.

3.2 ProSA

Das Tool ProSA dient zur Unterstützung der Analyse, Auswertung und Archivierung von großen Datenmengen, beispielsweise im Bereich des Forschungsdatenmanagement. Dabei verfolgt ProSA das Ziel eine Rekonstruktion der verwendeten Originaldaten, aus einem Ergebnis, durchzuführen. Diese Rekonstruktion dient dazu die verwendeten Originaldaten zu archivieren, um eine Nachvollziehbarkeit und Rückverfolgbarkeit des Ergebnisses zu gewährleisten. Besonders bei Veränderungen der Datenbank ist eine Archivierung der Originaldaten entscheidend, um die Nachvollziehbarkeit eines Ergebnisses zu einem späteren Zeitpunkt weiterhin zu gewährleisten. Allerdings ist eine vollständige Rekonstruktion aller Originaldaten nicht sinnvoll, da sonst bei jeder Veränderung der komplette Datensatz gespeichert werden muss. Jene Speicherung des kompletten Datensatzes, bei jeder Veränderung, würde zu Speicherung von massiven replizierten und redundanten Datensätze führen [AH18a]. Deswegen verfolgt ProSA das konkrete Ziel eine minimale Teildatenbank zu berechnen, um möglichst keine redundanten und replizierte Datensätze zu archivieren und die Menge der gespeicherten Originaldaten möglichst gering zu halten. Folglich soll die minimale Teildatenbank nur die verwendeten Originaldaten beinhaltet, die benötigt werden um das Ergebnis zu rekonstruieren.

3.2.1 Inversen in ProSA

Für die Berechnung einer minimalen Teildatenbank kombiniert ProSA den CHASE-Algorithmus mit Data Provenance. Hierbei dient der CHASE für die Einarbeitung einer Anfrage in einer Datenbank (Fall VI in Tabelle 3.1), wobei die Anfrage in Form einer Menge von Abhängigkeiten (S-t-tdgs, Tgds und Egds) vorliegen. Anschließend wird eine invertierte Anfrage benötigt, um eine Rückverfolgung vom Anfrageergebnis zu den verwendeten Originaldaten vorzunehmen. Folglich wird in ProSA die automatische Invertierung einer Anfrage (Inverse) benötigt, um die verwendeten Originaldaten und somit die minimale Teildatenbank zu berechnen. An dieser Stelle sei anzuführen, dass die Anwendung einer Inversen auf das Anfrageergebnis möglicherweise nicht zur vollständi-

gen Rekonstruktion der verwendeten Originaldaten führt. Infolgedessen ist die Inverse nicht vollständig und führt zu einem Informationsverlust. Diese Informationsverlust kann durch den Einsatz von Data Provenance reduziert werden, indem zusätzliche Informationen gespeichert werden. Die zusätzlichen Informationen liegen in Form von Zeugenbasen (Why-Provenance)[CCT09] oder Provance-Polynome (How-Provenance) [CCT09] vor. Ebenfalls können in ProSA sogennannte Side-Tables gespeichert werden, die zusätzliche Informationen für die Bildung der minimalen Teildatenbank enthalten. Allerdings ist eine vollständige Berechnung einer Inverse nicht immer notwendig oder gewollt, beispielsweise unter Datenschutzanforderungen werden bestimmte Informationen anonymisiert [AH19]. Dementsprechend ist ein Teilziel von ProSA eine Inverse (CHASE-Inverse) einer Anfrage automatisch zu berechnen, mit Berücksichtigung von Data Provenance.

In Abbildung 3.1 ist die Berechnung der minimale Teildatenbank, mithilfe des CHASE und der Inverse, unter einer Schemaevolution veranschaulicht. Hierbei wird eine Anfrage Q mithilfe des CHASE-Algorithmus auf die Datenbankinstanz $I(S_1)$ angewendet, um eine Anfrageergebnis (grüne Box) in der Instanz $K(S_2)$ zu erzeugen. Darauffolgend wird eine invertierte Anfrage Q_{prov} berechnet, die für die Bestimmung der minimale Teildatenbank (rote Box) benötigt wird. Des weiterem ist in Abbildung 3.1 der Fall einer Veränderung der Originaldaten bzw. des Quellschemas S_1 dargestellt. Bei dieser Veränderung wird eine Schemaevolution $\mathcal{E} : S_1 \rightarrow S_3$ durchgeführt, die das Quellschema S_1 in ein neues Schema S_3 überführt. Ebenfalls wird eine neue (invertierte) Anfrage Q'_{prov} definiert, die eine Rückabbildung von der Instanz $K(S_2)$ auf die neue Instanz $J(S_3)$ realisiert. Dementsprechend wird mit der (invertierte) Anfrage Q'_{prov} die minimale Teildatenbank (blaue Box), der Anwendung der Anfrage Q auf die Instanz $J(S_3)$, berechnet. Hierbei kann die Anfrage Q'_{prov} direkt mit der Komposition der originalen Anfrage Q mit der invertierten Schemaabildung $\mathcal{E}'$ berechnet werden. Diesbezüglich wird die neue minimale Teildatenbank (blaue Box) durch folgende Komposition bestimmt [AH19]:

$$Q'_{prov}(K^*(S_2)) = (Q_{prov} \circ \mathcal{E})(K^*(S_2)).$$

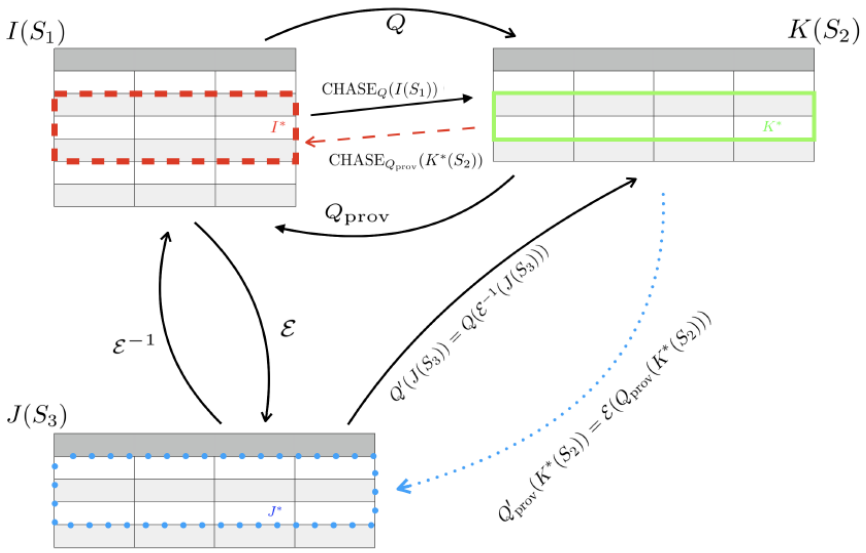

Abbildung 3.1: Inversen in ProSA [AH19]

3.2.2 ProSA-Ablauf

Der generelle Ablauf von ProSA für die Bestimmung der minimalen Teildatenbank, ist in Abbildung 3.2 veranschaulicht. Im Allgemeinen ist die Grundidee von ProSA, eine SQL-Anfrage in eine Menge von Abhängigkeiten, in Form von S-t-tgds und Tgds, zu konvertieren und anschließend diese Abhängigkeiten zu invertieren. Darauffolgend werden XML-Dokumente generiert, die die Abhängigkeiten (S-t-tgds, Tgds und Egds) und die verwendeten Datenbankinstanzen beinhalten. Folglich wird ein XML-Dokument für die eigentliche Anfrage generiert und ein XML-Dokument für die invertierte Anfrage bzw. Abhängigkeiten erzeugt. Diese XML-Dokumente dienen als Eingabe für das Tool ChaTEAU (Abschnitt 3.1). Hierbei dient ChaTEAU für die Anwendung des CHASE-Algorithmus auf die Menge von Abhängigkeiten. Dementsprechend wird zuerst der CHASE-Algorithmus angewendet, um das Anfrageergebnis der originalen Anfrage zu erzeugen. Darauffolgend wird der CHASE auf das Anfrageergebnis mithilfe der invertierten Abhängigkeiten (Inversen) angewendet, um eine minimale Teildatenbank zu berechnen. Ebenfalls können bei der Berechnung der minimalen Teildatenbank Provenance-Informationen (where, why, how) [CCT09] verwendet werden, um eine genauere Teildatenbank zu bestimmen. Beispielsweise können Zeugenbasen (why-Provenance) verwendet werden, um die genaue Tupelanzahl der verwendeten Datenbankinstanz zu rekonstruieren. Im Folgendem werden die einzelnen Module von ProSA (Abbildung 3.2) beschrieben, nach [Aug21a]:

GUI. Die GUI dient als Eingabe der SQL-Anfrage und der Provenance-Informationen. Ebenfalls wird durch die GUI eine Verbindung zu einer Postgres-Datenbank realisiert.

Parser. Der Parser dient zur Konvertierung der SQL-Anfrage in eine Menge von Abhängigkeiten, in Form von S-t-tgds, Tgds und Egds. Eine konkrete Beschreibung des Parsers ist im Abschnitt 3.2.3 angegeben.

Provenancer. Der Provenancer dient für die Verwendung von Data Provenance. Einerseits dient der Provenancer zur Vorbereitung der Datenbankinstanzen, indem er eindeutige Tupel-IDs zu den Instanzen in dem XML-Dokument hinzufügt. Diese Vorbereitung wird vor dem erstem CHASE-Schritt angewendet. Ebenfalls werden die benötigten Side-Tables definiert, die im CHASE-Schritt mit Informationen bzw. Tupeln gefüllt werden. Andererseits dient der Provenance für die Extraktion der minimalen Teildatenbank aus dem XML-Dokument, nach dem CHASE-Schritt (7).

ChaTEAU. Der CHASE-Algorithmus bildet ein zentrales Element in ProSA. Einerseits wird durch den CHASE das Anfrageergebnis berechnet, anderseits dient er zur Bestimmung der minimalen Teildatenbank. Dementsprechend erfolgen in ProSA zwei CHASE-Schritte, die durch das Tool ChaTEAU (Kapitel 3.1) realisiert werden.

Anonymisierer. Der Anoymisierer dient für die Einhaltung von Datenschutzanforderungen. Infolgedessen werden bestimmte Informationen anonymisiert, um die Anforderungen einzuhalten.

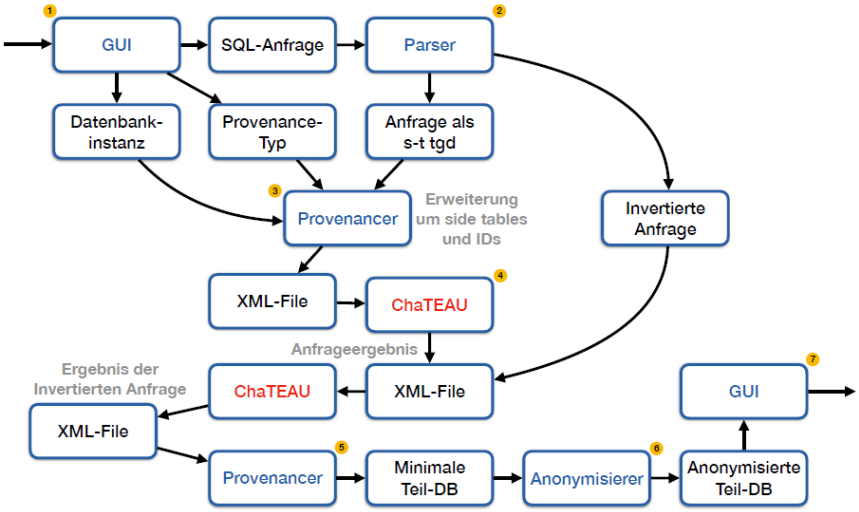

Abbildung 3.2: Ablauf von ProSA [Aug21a]

3.2.3 ProSA-Parser

Der ProSA-Parser [Aug21b] dient zur Konvertierung einer SQL-Anfrage in eine Menge von Abhängigkeiten (S-t-tgd, Tgd und Egd). Die Konvertierung basiert auf die Grundidee, eine SQL-Anfrage in einfache Basisanfragen zu zerlegen und anschließend einzeln zu konvertieren. Für die Zerlegung der Anfrage wird ein Syntaxbaum generiert, der die Variablen der Quellrelationen als Knoten verwendet und die Verbundattribute mit einer Kante verbindet. Anschließend kann eine Abhängigkeit/S-t-tgd aus dem Syntaxbaum abgeleitet werden. Infolgedessen bilden die Knoten des Syntaxbaums die Quellrelationen im Rumpf der S-t-tgd, mit dem entsprechenden Variablen. Außerdem bilden die Kanten die Variablen die gleichgesetzt werden, indem beide Variablen den gleichen Namen und den gleichen Index erhalten. Abschließend wird der Kopf der S-t-tgd aus der SELECT-Klausel abgeleitet, indem ein neues Relationssymbol (z.B. *Res*) eingeführt wird. Das neue Relationssymbol erhält die Variablen/Attribute, die in der SELECT-Klausel verwendet worden sind. Die Konvertierung der SQL-Anfrage ist nach Ablauf dieser Schritte abgeschlossen. Für die Veranschaulichung wird im Algorithmus 2, der Ablauf der Konvertierung dargestellt.

Beispiel. Im folgenden wird ein Beispiel für die Konvertierung der Anfrage 3.1 in eine S-t-tgd, anhand des Algorithmus 2, durchgeführt.

```
SELECT Lastname, FirstName
FROM Students NATURAL JOIN Grades
```

Anfrage 3.1: SQL-Beispiel (ähnlich wie in [Aug21b])

Als erster Schritt wird ein Syntaxbaum für die Anfrage aufgestellt. Hierbei dienen die Attribute der Quellrelationen *Students* und *Grades* als Knoten. Außerdem wird ein Verbund der beiden Quellrelationen, über dem Attribut *MatricNo* realisiert. Dementsprechend wird eine Kante, zwischen den Verbundattribute *St.MatricNo* und *G.MatricNo*, gebildet. In Abbildung 3.3 ist der Syntaxbaum für die Anfrage 3.1 veranschaulicht.

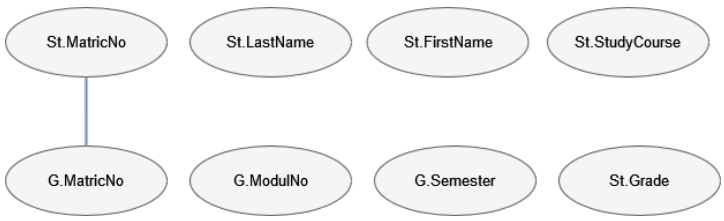

Abbildung 3.3: Syntaxbaum von Anfrage3.1 (ähnlich wie in [Aug21b])

Im nächsten Schritt wird der Rumpf der S-t-tgd erstellt, indem die Konten des Syn-

taxbaums betrachtet werden. Hierbei wird folgender Rumpf generiert:

$$St(ma_1, ln, fn, sc) \wedge G(ma_1, mo, se, g)$$

Im Syntaxbaum bilden die Attribute *St.MatricNo* und *G.MatricNo* eine Zusammenhangskomponente. Dementsprechend werden diese Attribute im Rumpf, durch einen gleichen Index, gleichgesetzt. Im letzten Schritt wird der Kopf aus der SELECT-Klausel abgeleitet. Folglich wird ein neues Relationssymbol *Res* eingeführt, welches die Attribute *LastName* und *FirstName* (der SELECT-Klausel) beinhaltet. Schlussendlich ergibt die Konvertierung, folgende S-t-tgd als Ergebnis:

$$St(ma_1, ln, fn, sc) \wedge G(ma_1, mo, se, g) \rightarrow Res(ln, fn).$$

In diesem Abschnitt wurde das Prinzip für die Konvertierung einer SQL-Anfrage in eine Abhängigkeit (S-t-tgd) vorgestellt. Im folgenden Abschnitt wird die konkrete Umsetzung dieses Prinzip beschrieben. Hierbei wurde im Rahmen eines KSWS-Projekt die konkrete Umsetzung des ProSA-Parsers entwickelt. Ebenfalls wird im folgenden Abschnitt eine Methode für die Invertierung einer SQL-Anfrage betrachtet. Diese Methode für die Invertierung verwendet die Funktionalitäten des ProSA-Parser, um für eine gegebene SQL-Anfrage eine invertierte S-t-tgd zu generieren. Ebenfalls wurde diese Invertierungsmethode im Rahmen eines KSWS-Projektes entwickelt.

Algorithmus 2 : Konvertierung einer SQL-Anfrage in S-t-tgds [Aug21b]

Input : SQL-Anfrage Q
Output : Anfrage Q' als S-t-tgd

1. Aufbau des Syntaxbaums
   a) Knoten: Variablen der Quellrelationen
   b) Kanten: Verbundattribute mit Kante verbunden
2. Zusammenhangskomponenten im Syntaxbaum bestimmen
3. Rumpf:
   a) Quellrelationen aufstellen (Knoten)
   b) Attribute innerhalb einer Zusammenhangskomponente des Snytaxbaumes gleichsetzen (Kanten)
4. Kopf: Variablen der SELECT-Klausel übernehmen, dabei pro Zusammenhangskomponente nur eine Variable ins Ergebnisschema übernehmen.

3.3 KSWS-Projekt

In diesem Abschnitt werden die Projekte *sql2sttgd* [KRSZ21] und *Data Provenance* [WWO+21] vorgestellt, die an der Universität Rostock im Rahmen der Veranstaltung KSWS (*Komplexe Software-Systeme*) entwickelt wurde. Das KSWS-Projekt sql2sttsgd beschäftigte sich mit der Konvertierung von SQL-Anfragen in eine XML-Datei, die die Abhängigkeiten repräsentiert und die Eingabe für ChaTEAU liefert, im Rahmen von ProSA. Dementsprechend war das Ziel für die technische Umsetzung des ProSA-Parsers (Abschnitt 3.2.3) Methoden zu entwickeln und diese in der Programmiersprache Java zu implementieren. Dagegen wurde in [WWO+21] eine Methode für die Invertierung der SQL-Anfragen entwickelt, auf Basis des ProSA-Parsers (SQL-Parsers).

3.3.1 Aufbau vom Parser

Der Aufbau des Projekts sql2sttgd wird in Abbildung 3.4 skizziert. Hierbei wird als Eingabe eine SQL-Anfrage genommen und als Ausgabe wird einen XML-Datei geliefert, die die Menge der Abhängigkeiten bzw. die S-t-tgds beinhaltet. Bezüglich Abbildung 3.4 ist die Umsetzung der Konvertierung von einer SQL-Anfrage in eine S-t-tgd (bzw. XML-Datei), in mehreren Modulen aufgeteilt. Diese einzelnen Module werden im Folgenden betrachtet, nach [KRSZ21]:

SQL-Parser. Die Aufgabe des *SQL-Parsers* ist es den Syntaxbaum aus der gegebenen SQL-Anfrage aufzustellen, um damit die weitere Konvertierung zu vereinfachen. Hierzu nimmt der Parser die SQL-Anfrage als String entgegen und prüft diese auf Syntaxfehler.

DB-Reader. Für die Bereitstellung der Datenbankinstanzen, die in der S-t-tgd verwendet werden, dient das Modul *DB-Reader*. Dementsprechend nimmt das Modul als Eingabe die Tabellennamen vom SQL-Parser, um damit die Tupel aus einer Datenbank (Postgres-Datenbank) auszulesen. Folglich werden aus den verwendeten Tupeln, die Instanzen für die S-t-tgd erzeugt.

Transformation. Im Modul *Transformation* wird die eigentliche Konvertierung der SQL-Anfrage, anhand des generierten Syntaxbaum im *SQL-Parser*, durchgeführt.

Compositor. Das Modul *Compositor* erhält die S-t-tgds vom Modul *Transformation* und die Instanzen vom *DB-Reader* und liefert als Ausgabe die zusammengeführte Informationen, im Form einer XML-Datei. Diese Datei dient anschließend als Eingabe für ChaTEAU.

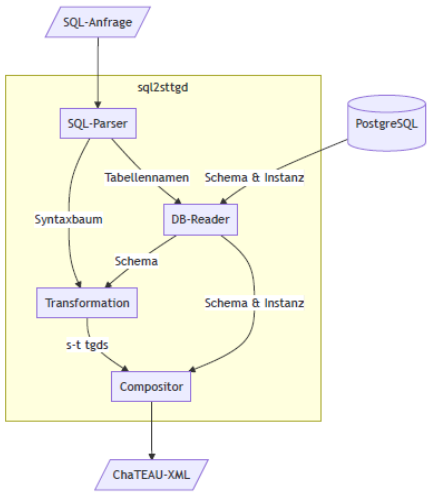

Abbildung 3.4: Aufbau Projekt sql2sttgd [KRSZ21]

3.3.2 Parser

An dieser Stelle wird die technische Umsetzung von der Konvertierung der SQL-Anfragen in eine Menge von Abhängigkeiten (S-t-tdgs) beschrieben. Daraufhin werden die Module *SQL-Parser* und *Transformation* betrachtet, da diese die primären Module für die eigentliche Konvertierung darstellen. Infolgedessen wird durch den *SQL-Parser* der Syntaxbaum einer SQL-Anfrage aufgestellt. Dieser Syntaxbaum wird durch die Bibliothek *JSQLParser* [JSQ22] generiert. Der JSQLParser nimmt als Eingabe eine SQL-Anfrage und liefert als Ausgabe die SQL-Anfrage als Hierarchie von Java-Klassen zurück. Folglich wird die erzeugte Java-Klassenhierache als Syntaxbaum interpretiert und eine Verschachtelung der SQL-Anfragen erzeugt. Diesbezüglich wird die Verschachtelung der SQL-Anfragen benötigt, um die Anfragen zu vereinfachen, indem die Anfragen auf ihre atomaren Teilanfragen herunter gebrochen werden. Anschließend werden diese atomaren Teilanfragen im Modul *Transformation* in S-t-tgds konvertiert. Für die Konvertierung der einzelnen Teilanfragen wird zuerst der Rumpf der S-t-tgd gebildet, indem für jede Relation in der FROM-Klausel ein Atom erstellt wird. Hierbei werden, mithilfe des Moduls *DB-Reader*, die Relationen aus der Datenbank ausgelesen und aus diesen Informationen die Atome für den Rumpf gebildet. An dieser Stelle sei anzuführen, dass die Variablen bzw. Attribute in den S-t-tgds erstmal unterschiedliche Indizes besitzen. Dementsprechend werden im nächsten Schritt die Indizes der Variablen gleichgesetzt, sollte eine Verbund-Operation in der SQL-Anfrage existieren. Folglich erhalten alle Verbundattribute/Variablen den gleichen Index, über die der Verbund realisiert wird. Darauffolgend wird Kopf der S-t-tgd gebildet, indem eine neue Relation bzw. ein neues Atom *Res* (*Result*) gebildet wird. Das Atom *Res* erhält alle Attribute, die in der (obersten) SELECT-Klausel angegeben wurden. Abschließend wird die WHERE-Klausel der Anfrage

ausgewertet und dementsprechend nach Bedingungen an die Attribute/Attributwerte gesucht. Ebenfalls wird in diesem Schritt geprüft, ob die Datentypen von dem Attributwerten von ChaTEAU unterstützt werden. Derzeit unterstützt ChaTEAU nur die Datentypen *String*, *Integer* und *Double*. Nach der Auswertung der WHERE-Klausel ist die Konvertierung, der atomaren Anfrage in eine S-t-tgd, abgeschlossen.
Für die Veranschaulichung wird im Folgenden ein Beispiel, anhand der Anfrage 3.2 durchgeführt. In der Anfrage 3.2 wird ein Verbund zwischen den Relationen*Students* und *Grades* realisiert. Infolgedessen werden alle Studierende mit den Attributen *LastName* (kurz: ln) und *FirstName* (kurz: fn), die eine Note von 1.3 geschrieben haben, zurückgeliefert. Als ersten Schritt in der Konvertierung wird die Anfrage, mithilfe vom JSQLParser, verschachtelt und dementsprechend in ihre Teilanfragen aufgeteilt. Das Ergebnis der Verschachtelung ist in Anfrage 3.3 dargestellt. Als nächsten Schritt werden die Teilanfragen konvertiert, beginnend mit der innersten Anfrage. Dementsprechend wird zuerst der Rumpf der Verbund-Anfrage gebildet, indem die FROM-Klausel ausgewertet wird. Demnach ergibt sich folgender Rumpf, mit den zwei Atomen/Relationen *Students* (kurz: St) und *Grades* (kurz: G):

$$St(ma_1, ln_1, fn_1, sc_1) \wedge G(mo_2, ma_2, se_2, g_2).$$

Darauffolgend müssen noch die Indizes der Variablen für den natürlichen Verbund angepasst werden. In diesem Beispiel wird der Verbund über das Verbundattribut *MatricNo* (kurz: ma) realisiert. Folglich erhalten die beiden Variablen *MatricNo* (kurz: ma) die gleiche Indizes:

$$St(ma_1, ln_1, fn_1, sc_1) \wedge G(mo_2, ma_1, se_2, g_2).$$

Im nächsten Schritt wird der Kopf der S-t-tgd gebildet, indem die (oberste) SELECT-Klausel betrachtet wird. Hierbei wird das Atom Res mit den Attributen in der SELECT-Klausel erzeugt. In dem Beispiel erhält das Atom/Relation Res die Attribute *LastName* und *FirstName*.

$$St(ma_1, ln_1, fn_1, sc_1) \wedge G(mo_2, ma_1, se_2, g_2) \rightarrow Res(fn, ln).$$

Schlussendlich werden die Bedingungen der WHERE-Klausel hinzugefügt. Demnach wird der Attributwert vom Attribut *Grade* (kurz: g) auf 1.3 gesetzt. Folglich entspricht die folgende S-t-tgd, dann auch dem Gesamtergebnis der Konvertierung der Anfrage 3.2:

$$St(ma_1, ln_1, fn_1, sc_1) \wedge G(mo_2, ma_1, se_2, 1.3) \rightarrow Res(fn, ln).$$

```
SELECT LastName, FirstName
FROM Students NATURAL JOIN Grades
WHERE Grade = '1.3'
```

Anfrage 3.2: SQL-Beispiel (ähnlich wie in [KRSZ21])

```
SELECT LastName, FirstName
FROM (
        SELECT *
        FROM (
                SELECT *
                FROM Students NATURAL JOIN Grades)
        WHERE Grade = '1.3')
```

Anfrage 3.3: Verschachtelung der Anfrage 3.2 (ähnlich wie in [WWO+21])

3.3.3 Invertierungsmethode

Eine Methode der Invertierung der SQL-Anfragen bzw. der erzeugten S-t-tgd wird im KSWS-Projekt: *Data Provenance* [WWO+21] veranschaulicht, basierend auf der Methodik des SQL-Parsers. Das Grundprinzip der Methode verwendet die Verschachtelung der SQL-Anfrage, die mithilfe des SQL-Parsers erzeugt wird, um damit die einzelnen atomaren Anfragen in eine S-t-tgd zu konvertieren. Darauffolgend werden die einzelnen S-t-tgds invertiert, mithilfe von Regeln für die Invertierung, und anschließend zu einer Inverse zusammengefasst.
Als erster Schritt wird die Verschachtelung der SQL-Anfrage durchgeführt, beispielsweise wie in Abbildung 3.3. Hierbei wird der Verbund im Innersten, die Selektion in der Mitte und die Projektion im Äußeren realisiert. Für die Umsetzung/Implementation dieses Prinzip muss ein SFW-Baum (SELECT-FROM-WHERE) gebildet werden, der die einzelnen Elemente (Projektion, Selektion, Verbund) und die Struktur der Anfrage wiedergibt. Ein Beispiel eines SFW-Baumes, für die Anfrage 3.2, ist in Abbildung 3.5 veranschaulicht.
Im nächsten Schritt müssen die einzelnen (Teil-)Anfragen in S-t-tgds konvertiert werden. Die einzelnen Anfragen befinden sich in jedem Teilbaum des SFW-Baumes. Bei der Konvertierung wird mit der innersten Anfrage, anhand der Verschachtelung, begonnen. Anschließend werden die einzelnen (Teil-)Anfragen von Innen nach Außen konvertiert. Infolgedessen wird der Kopf der letzten konvertierten S-t-tgd, zum Rumpf der nächsten konvertierten S-t-tgd.
Im nächsten Schritt werden die einzelnen S-t-tgds invertiert, indem vordefinierte Grundregeln für die Invertierung der einzelnen Elemente (Projektion, Selektion, Verbund) einer Anfrage/S-t-tgd angewendet werden. Bei der Invertierung werden die einzelnen S-t-tgds in der entgegengesetzte Reihenfolge zu der Konvertierung (vorherigen Schritt) invertiert. Hierbei bildet der Kopf der letzten invertierten S-t-tgd den Rumpf der nächsten invertierten S-t-tgd.
Schließlich wird im letzten Schritt die einzelnen invertierten (Teil-)S-t-tgds, zu einer invertierten (Gesamt-)S-t-tgd zusammengefasst. Folglich wird die Zusammenfassung realisiert, indem der Rumpf der ersten invertierten (Teil-)S-t-tgd mit dem Kopf der letzten invertierten (Teil-)S-t-tgd verbunden wird.

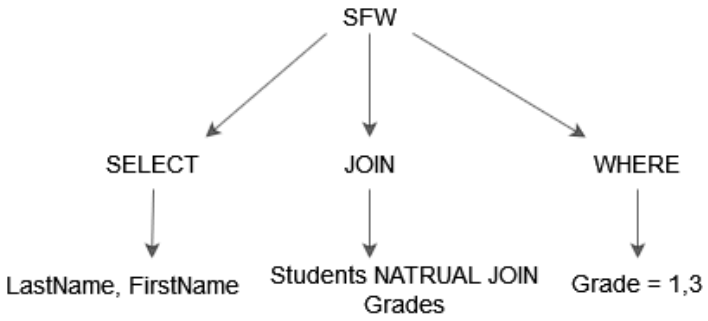

Abbildung 3.5: SFW-Baum für Anfrage 3.2 (ähnlich wie in [WWO+21])

Beispiel. Für die Veranschaulichung der Invertierungsmethode wird im Folgenden die Invertierung der Anfrage 3.2 durchgeführt. Im ersten Schritt wird der SFW-Baum (Abbildung 3.5) bzw. die Verschachtelung (Anfrage 3.3) aufgebaut. Anschließend wird im nächsten Schritt die einzelnen Anfragen konvertiert, in der Reihenfolge Verbund (innerste Anfrage), Selektion und zum Schluss die Projektion (äußerste Anfrage). Um eine Vereinfachung der einzelnen S-t-tgds durchzuführen wird angenommen, dass die Relation *Students* nur die Attribute *MatricNo*, *LastName* und *FirstName* besitzt und die Relation *Grades* ausschließlich aus den Attributen *Grade* und *MatricNo* besteht. Folglich werden folgende S-t-tgds gebildet:

$$St(ma, ln, fn) \wedge G(ma, g) \rightarrow R(ma, fn, ln, g) \qquad \text{Verbund}$$

$$R(ma, fn, ln, g) \wedge g = 1.3 \rightarrow R'(ma, fn, ln, 1.3) \qquad \text{Selektion}$$

$$R'(ma, fn, ln, 1.3) \rightarrow R''(ln, fn) \qquad \text{Projektion}$$

Darauffolgend werden die einzelnen S-t-tgd invertiert, in umgekehrter Reihenfolge, mit den definierten Invertierungsregeln:

Invertierung Projektion

S-t-tgd: $R'(ma, fn, ln, 1.3) \rightarrow R''(ln, fn)$
Regel: $Q(a, b) \rightarrow R(a) \Rightarrow R(a) \rightarrow \exists b : Q(a, b)$
Invertierte S-t-tgd: $R''(ln, fn) \rightarrow \exists ma : R'(ma, fn, ln, 1.3)$

Invertierung Selektion

S-t-tgd: $R(ma, fn, ln, g) \wedge g = 1.3 \rightarrow R'(ma, fn, ln, 1.3)$
Regel: $Q(a, b) \wedge a = c \rightarrow R(c, b) \Rightarrow R(c, b) \rightarrow Q(c, b)$
Invertierte S-t-tgd: $R'(ma, fn, ln, 1.3) \rightarrow \exists ma : R(ma, fn, ln, 1.3)$

Invertierung Verbund

S-t-tgd: $St(ma, ln, fn) \wedge G(ma, g) \rightarrow R(ma, fn, ln, g)$
Regel: $Q(a, b) \wedge S(b, d) \rightarrow R(a, b, d) \Rightarrow R(a, b, d) \rightarrow Q(a, b) \wedge S(b, d)$
Invertierte S-t-tgd: $R(ma, fn, ln, 1.3) \rightarrow \exists ma : St(ma, ln, fn) \wedge G(ma, 1.3)$

Abschließend werden die invertierten (Teil-)S-t-tgds zu einer invertierten (Gesamt-)S-t-tgd zusammengefügt. Bezüglich des Beispiels wird folgende zusammengesetzte S-t-tgd generiert:

$$R''(ln, fn) \rightarrow \exists ma : St(ma, ln, fn) \wedge G(ma, 1.3).$$

In diesem Abschnitt wurde eine hybride Methode für die Invertierung einer SQL-Anfrage betrachtet. Diese hybride Methode erhält als Eingabe eine SQL-Anfrage und liefert als Ausgabe eine invertierte Abhängigkeit (S-t-tgd) zurück. Dabei muss für die Invertierung einer SQL-Anfrage diese Methode in den SQL-Parser integriert werden. Im folgenden Abschnitt wird eine weitere Methode für die Invertierung von Abhängigkeiten vorgestellt. Diese Methode stellt eine allgemeinen Algorithmus für die Invertierung von Abhängigkeiten bzw. von einer Schemaabbildung dar. Entsprechend muss die folgende Methode nicht in den SQL-Parser integriert werden. Außerdem erhält die folgende Methode (AdaptedMaxExtendedRecovery) eine Menge von Abhängigkeiten bzw. eine Schemaabbildung als Eingabe und liefert als Ausgabe eine Menge von invertierten Abhängigkeiten bzw. eine invertierte Schemaabbildung zurück. Folglich müssen für eine Anwendung dieser Methode in ProSA, die gegebenen SQL-Anfragen zuerst in eine Menge von Abhängigkeiten konvertiert werden. Diese Konvertierung würde durch den ProSA-Parser (SQL-Parser) erfolgen.

3.4 AdaptedMaxExtendedRecovery

In der Masterarbeit *Datenintegration durch inverse Schemaabbildungen: Erweiterung der Rostocker GaLVE-Technik* [Zim21], von Jarkob Zimmer, wird ein Algorithmus für die Invertierung von Schemaabbildungen in ChaTEAU konstruiert. Dieser Algorithmus *AdaptedMaxExtendedRecovery*, basiert auf dem *Maximum-Extended-Recovery* [FKPT11a] Verfahren. Hierbei wurde das Verfahren Maximum-Extended-Recovery gewählt, da die Anwendung von Nullwerten in den Quellschematas der Schemaabbildungen erlaubt sind und Nullwerte bei der Datenintegration im Quellschema auftreten können. Allerdings muss diese Verfahren für Anwendung in ChaTEAU angepasst und erweitert werden. Beispielsweise liefert der Maximum-Extended-Recovery-Algorithmus eine S-t-tgd zweiter Ordnung, als Inverse einer Schemaabbildung, als Ausgabe. Allerdings muss für die Verwendung in ChaTEAU eine Inverse, durch eine S-t-tgd erster Ordnung repräsentiert sein. Aus diesem Grund, um eine Anpassung für ChaTEAU vorzunehmen, wurde der AdaptedMaxExtendedRecovery-Algorithmus konstruiert. Für die Veranschaulichung ist der AdaptedMaxExtendedRecovery-Algorithmus, im Algorithmus 3 dargestellt. Als Eingabe erhält der Algorithmus eine Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ eine Menge von Abhängigkeiten in Form von S-t-tgds entspricht. Des weiterem entspricht S einem Quellschema, das Nullwerte enthalten kann, und T einem Zielschema. Dagegen wird als Ausgabe eine invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega')$ von $\mathcal{M}$ geliefert, wobei Ω' eine Menge von Abhängigkeiten in Form von S-t-tgds (erster Ordnung) entspricht.

Diesbezüglich liegen die S-t-tgds in Ω' als *Disjunktive-Mengen* vor, die folglich definiert sind.

Definition 3.1 (Disjunktive Mengen, [Zim21])**.** Ein Disjunktive-Menge ist eine Menge von S-t-tgds, wobei die Rümpfe der S-t-tgds nur aus einem atomaren Ausdruck bestehen. Diese S-t-tgd Rümpfe werden über demselben Relationsschema definiert.

Im ersten Schritt des Algorithmus 3 wird eine Normalisierung der Konjunktion der Köpfe der S-t-tgds vorgenommen. Infolgedessen wird für jedem atomaren Ausdruck im Kopf einer S-t-tgd, eine einzelne S-t-tgd mit nur dem atomaren Ausdruck im Kopf generiert.
Im zweitem Schritt werden dijunktive Tgds erstellt, die als Inversen fungieren, indem eine Abbildung vom Zielschema T zum Quellschema S realisiert wird. Für die Erstellung der disjunktiven Tgds werden alle Relationsschemas R im Zielschema T betrachtet, die im Kopf einer S-t-tgd von Σ auftreten. Dementsprechend wird für jedes Relationsschema $R \in T$,das im Kopf einer S-t-tgd auftritt, eine Tgd erstellt. Ebenfalls werden für jedes betrachtete k-stellige Relationsschema R neue Variablen $z_{R_1}, ..., z_{R_k}$ definiert. Diese neuen Variablen werden mithilfe eines Homomorphismus auf die Variablen $t_1, ..., t_k$ von R abgebildet. Ebenfalls wird eine Menge S_R für jedes verwendete R eingeführt. Die Menge S_R enthält als Elemente die Konjunktion von den Abbildungen der neuen Variablen (z.B. $h(z_{R_1}) = t_1$) mit dem Kopf einer S-t-tgd die R enthält. Anschließend wird aus einer Menge S_R eine disjunktive Tgd gebildet, indem $R(z_{R_1}, ..., z_{R_k})$ den Rumpf der Tgd bildet und der Kopf aus der disjunktiven Verknüpfung der Element aus S_R besteht. Folglich werden alle Variablen die im Kopf und nicht im Rumpf der disjunktiven Tgd vorhanden sind, existenzquantifiziert. Dagegen werden alle anderen Variablen allquantifiziert.
Im drittem Schritt wird eine Normalisierung der Köpfe der disjunktiven Tgds vorgenommen. Dementsprechend wird die Disjunktion in den Tgds aufgelöst, indem neue S-t-tgds generiert werden. Diese neuen S-t-tgds enthalten im Rumpf das Relationsschema der Form $R(z_{R_1}, ..., z_{R_k})$ und im Kopf ein Element aus der Menge S_R. Infolgedessen werden die neuen S-t-tgd in einer Menge $\Omega = \{\omega_1, ..., \omega_q\}$ zusammengefasst, dabei entsteht jedes $\omega_i \in \Omega$ aus einer disjunktiven Tgd.
Schließlich wird im vierten Schritt des Algorithmus die Abbildungen, aus dem zweiten Schritt, in jeder S-t-tgd in einem ω_i angewendet. Darauffolgend bildet die Menge Ω, mit den angewandten Abbildungen, die Menge Ω', die die invertierten S-t-tgd von Σ enthält. Dementsprechend liegen die S-t-tgd in Ω' als Disjunktive-Mengen vor.

Algorithmus 3 : AdaptedMaxExtendedRecovery($\mathcal{M}$) [Zim21]

Input : Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ eine endliche Menge von S-t-tgds ist.

Output : *Adapted Strong Maximum Extended Recovery* $\mathcal{M}' = (T, S, \Omega)$ von $\mathcal{M}$, wobei Ω eine Menge von Mengen von S-t-tgds (Mengen von Disjunktiven-Mengen) in Logik erster Ordnung ist.

1. (Normalisiere die Konjunktionen in den Köpfen der S-t-tgds.) Erstelle Σ' aus Σ, indem jede S-t-tgd $\alpha \rightarrow (\beta_1 \wedge ... \wedge \beta_r)$ in Σ durch S-t-tgds der Form $\alpha \rightarrow \beta_1, ..., \alpha \rightarrow \beta_r$ ersetzt wird, bei denen β_i im Kopf atomar ist.

2. (Erstelle disjunktive Tgds über einem Homomorphismus.) Für jedes Relationsschema $R \in T$ (wobei R k-stellig ist) seien $z_{R_1}, ..., z_{R_k}$ neue, eindeutige Variablen und S_R eine neue leere Menge. Für jede Formel $\alpha(x) \rightarrow R(t_1, ..., t_k)$ in Σ' (wobei der Ausdruck im Kopf über dem Relationsschema definiert ist), füge zu S_R die Formel $\exists x(\alpha(x) \wedge (h(z_{R_1}) = t_1) \wedge ... \wedge (h(z_{R_k}) = t_k))$ hinzu. Sei τ_R die disjunktive Tgd $R(z_{R_1}, ..., z_{R_k}) \rightarrow \bigvee\{\Phi | \Phi \in S_R\}$. Bestehe Σ'' aus den disjunktiven Tgds τ_R für jedes Relationenschema R in T.

3. (Normalisiere die Disjunktionen in den Köpfen der disjunktiven Tgds.) Sei $\Omega = \{\omega_1, ..., \omega_q\}$ eine Menge von Mengen von S-t-tgds (Mengen von Disjunktiven-Mengen), mit $q = |\Sigma''|$. Jedes $\omega_i \in \Omega$ entsteht aus einer disjunktiven Tgd in Σ''. Für eine disjunktive Tgd $R(z_{R_1}, ..., z_{R_k}) \rightarrow \bigvee\{\Phi | \Phi \in S_R\}$, wird für jedes $\Phi_j \in S_R$ eine S-t-tgd der Form $R(z_{R_1}, ..., z_{R_k}) \rightarrow \Phi_j$ zu Φ_j zu ω_i hinzugefügt.

4. (Wende den Homomorphismus an.) Erstelle Ω' aus Ω, indem die Abbildungen in den einzelnen S-t-tgds in jedem $\omega_i \in \Omega$ angewendet werden.

Return : $\mathcal{M}' = (T, S, \Omega')$.

Beispiel. Im Folgendem wird ein Beispiel für den AdaptedMaxExtendedRecovery-Algorithmus durchgeführt, ähnlich wie in [Zim21]. Hierbei dient als Eingabe die Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei $S = \{St = \{ma, fn, ln\}, Pa = \{ma, mo\}, G = \{ma, mo, g, se\}\}$ und $T = \{R = \{ma, ln, mo\}, Se = \{ma, ln, se\}\}$ entspricht. Die Abhängigkeiten Σ sind folgendermaßen, als S-t-tgd, definiert:

$$St(ma, fn, ln) \wedge Pa(ma, mo) \rightarrow R(ma, ln, mo)$$
$$G(ma, mo, g, se) \rightarrow \exists ln : Se(ma, ln, se) \wedge R(ma, ln, mo).$$

In der ersten Abhängigkeit (S-t-tgd) wird ein Verbund zwischen den Relationen *Students* (kurz: St) und *Participants* (kurz: Pa) realisiert, die eine Ergebnisrelation R erzeugt.

Dagegen wird in der zweiten Abhängigkeit die Relation *Grades* (kurz: G) in die Relationen Se (Semester) und R aufgeteilt.
Im ersten Schritt werden die Köpfe der S-t-tgds normalisiert, indem für jedes Prädikatensymbol im Kopf eine eigene S-t-tgd erstellt wird. Dementsprechend wird die Konjunktion in der zweiten S-t-tgd aufgelöst und der Exixtenzquantor auf zwei S-t-tgds aufgeteilt. Folglich bilden sich die normalisierten S-t-tgds:

$$St(ma, fn, ln) \wedge Pa(ma, mo) \rightarrow R(ma, ln, mo)$$
$$G(ma, mo, g, se) \rightarrow \exists ln : Se(ma, ln, se)$$
$$G(ma, mo, g, se) \rightarrow \exists ln : R(ma, ln, mo).$$

Im zweitem Schritt werden die disjunktive Tgds gebildet. Infolgedessen wird für die Relationssymbole $R, Se \in T$ die im Kopf einer S-t-tgd vorkommen, die Mengen S_R und S_{Se} gebildet. Ebenfalls werden neue Variablen z_i eingeführt und ein Homomorphismus umgesetzt, der die neuen Variablen z_i auf die Variablen der Relationen R und Se abbildet. Diesbezüglich sind die Mengen S_R und S_{Se} folglich definiert:

$$S_{Se} = \{(G(ma, mo, g, se) \wedge (h(z_{Se_1}) = ma) \wedge (h(z_{Se_2}) = ln) \wedge (h(z_{Se_3}) = se))\}$$
$$S_R = \{(St(ma, fn, ln) \wedge Pa(ma, mo) \wedge (h(z_{R_1}) = ma) \wedge (h(z_{R_2}) = ln) \wedge (h(z_{R_3}) = mo)),$$
$$(G(ma, mo, g, se) \wedge (h(z_{R_1}) = ma) \wedge (h(z_{R_2}) = ln) \wedge (h(z_{R_3}) = mo))\}.$$

Darauffolgend werden die disjunktive Tgds, mithilfe der Mengen S_R und S_{Se}, erzeugt. Folglich dient die Disjunktive Verknüpfung einer Menge (S_R oder S_{Se}) als Kopf einer disjunktiven Tgd. Der Rumpf der Tgd wird aus den Relationssymbolen R und Se, mit den neuen Variablen z_i, gebildet. Entsprechend besteht Σ'' aus den folgenden disjunktiven Tgds:

$$Se(z_{Se_1}, z_{Se_2}, z_{Se_3}) \rightarrow \exists ma, mo, g, se, ln : (G(ma, mo, g, se) \wedge$$
$$(h(z_{Se_1}) = ma) \wedge (h(z_{Se_2}) = ln) \wedge (h(z_{Se_3}) = se))$$
$$R(z_{R_1}, z_{R_2}, z_{R_3}) \rightarrow \exists ma, mo, ln, fn : (St(ma, fn, ln) \wedge Pa(ma, mo) \wedge$$
$$(h(z_{R_1}) = ma) \wedge (h(z_{R_2}) = ln) \wedge (h(z_{R_3}) = mo)) \vee$$
$$\exists ma, mo, g, se, ln : (G(ma, mo, g, se) \wedge$$
$$(h(z_{R_1}) = ma) \wedge (h(z_{R_2}) = ln) \wedge (h(z_{R_3}) = mo))\}.$$

Im drittem Schritt werden die Köpfe der disjunktiven Tgds normalisiert und die Disjunktion aufgelöst. Dieser Schritt ist notwendig, um im vierten Schritt die Abbildungen des Homomorphismus anzuwenden. Aufgrund von unterschiedlichen Abbildungen von gleichen neuen Variablen, beispielsweise $h(z_{R_3}) = x$ und $h(z_{R_3}) = y$, muss die Disjunktion zuerst aufgelöst werden, um den Homomorphismus umzusetzen. Dementsprechend wird für jede disjunktive Tgd eine Menge $\omega_i \in \Omega$ angelegt, die die normalisierten Tgds

enthält. Im Beispiel werden in diesem Schritt zwei Mengen ω_1 und ω_2 generiert:

$$\omega_1 = \{Se(z_{Se_1}, z_{Se_2}, z_{Se_3}) \rightarrow \exists ma, mo, g, se, ln : (G(ma, mo, g, se) \wedge (h(z_{Se_1}) = ma) \wedge (h(z_{Se_2}) = ln) \wedge (h(z_{Se_3}) = se))\}$$
$$\omega_2 = \{R(z_{R_1}, z_{R_2}, z_{R_3}) \rightarrow \exists ma, mo, ln, fn : (St(ma, fn, ln) \wedge Pa(ma, mo) \wedge (h(z_{R_1}) = ma) \wedge (h(z_{R_2}) = ln) \wedge (h(z_{R_3}) = mo)),$$
$$R(z_{R_1}, z_{R_2}, z_{R_3}) \rightarrow \exists ma, mo, g, se, ln : (G(ma, mo, g, se) \wedge (h(z_{R_1}) = ma) \wedge (h(z_{R_2}) = ln) \wedge (h(z_{R_3}) = mo))\}.$$

Schließlich wird im viertem Schritt die Abbildungen des Homomorphismus umgesetzt. Folglich wird eine Ersetzung der Variablen z_i, im Rumpf der Tgds, vorgenommen:

$$\omega'_1 = \{Se(ma, ln, se) \rightarrow \exists mo, g : (G(ma, mo, g, se)\}$$
$$\omega'_2 = \{R(ma, ln, mo) \rightarrow \exists fn : St(ma, fn, ln) \wedge Pa(ma, mo),$$
$$R(ma, ln, mo) \rightarrow \exists g, se : G(ma, mo, g, se)\}.$$

Abschließend repräsentiert die Menge $\Omega' = \{\omega'_1, \omega'_2\}$ die invertierten Abhängigkeiten Σ, in Form von Disjunktiven-Mengen. Als Ausgabe liefert der Algorithmus die invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega')$ von der ursprünglichen Schemaabbildung $\mathcal{M}$.

Im Rahmen der Masterarbeit [Zim21] wurde der Algorithmus AdaptedMaxExtended-Recovery weiterhin angepasst, dass Ergebnis davon ist der Algorithmus *DirectAdapted-MaxExtendedRecovery*. Dieser Algorithmus wird im Algorithmus 4 veranschaulicht. Der DirectAdaptedMaxExtendedRecovery-Algorithmus besitzt die Besonderheit auf die Bildung von den disjunktiven Tgds und auf die Anwendung eines Homomorphismus zu verzichten. Folglich entfällt der zweite Schritt des AdaptedMaxExtendedRecovery-Algorithmus (Algorithmus 3). Stattdessen wird eine direkte Invertierung durchgeführt, indem eine Vertauschung von dem Kopf und dem Rumpf der S-t-tgds durchgeführt wird und die Existenzquantoren und Allquantoren angepasst werden. Als Eingabe erhält der Algorithmus eine Schemaabbildung $\mathcal{M}$ und als Ausgabe wird eine invertierte Schemaabildung $\mathcal{M}'$ zurückgegeben. Hierbei enthält $\mathcal{M}'$ die Menge der invertierten Abhängigkeiten Ω in Form von Disjunktiven Mengen. Dementsprechend liefern die beiden Algorithmen (3 und 4) das gleiche Ergebnis.
Im erstem Schritt des DirectAdaptedMaxExtendedRecovery-Algorithmus werden die Köpfe der S-t-tgds normalisiert und die Konjunktion im Kopf aufgelöst. Im zweiten Schritt werden die S-t-tgds, durch die Vertauschung von Kopf und Rumpf, invertiert. Dabei werden alle Variablen die im Kopf der invertierten S-t-tgd vorhanden sind, aber nicht im Rumpf vorliegen, existenzquantifiziert. Alle anderen Variablen werden allquantifiziert. Schließlich werden im letzten Schritt die Disjunktiven Mengen $\omega_p \in \Omega$ aus den invertierten S-t-tgd gebildet.

Beispiel. Für die Veranschaulichung der Unterschiede der beiden Algorithmen wird das obere Beispiel noch einmal mit dem DirectAdaptedMaxExtendedRecovery-Algorithmus durchgeführt. Folglich dient als Eingabe eine Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, mit $S = \{St = \{ma, fn, ln\}, Pa = \{ma, mo\}, G = \{ma, mo, g, se\}\}$ und $T = \{R = \{ma, ln, mo\}, Se = \{ma, ln, se\}\}$. Die S-t-tgds in Σ sind folglich definiert:

$$St(ma, fn, ln) \land Pa(ma, mo) \rightarrow R(ma, ln, mo)$$
$$G(ma, mo, g, se) \rightarrow \exists ln : Se(ma, ln, se) \land R(ma, ln, mo).$$

Im ersten Schritt werden die S-t-tgds normalisiert, indem die zweite S-t-tgd aufgeteilt wird. Hierbei existieren keine Unterschiede zum oberen Beispiel, somit ergeben sich folgende normalisierte S-t-tgds:

$$St(ma, fn, ln) \land Pa(ma, mo) \rightarrow R(ma, ln, mo)$$
$$G(ma, mo, g, se) \rightarrow \exists ln : Se(ma, ln, se)$$
$$G(ma, mo, g, se) \rightarrow \exists ln : R(ma, ln, mo).$$

Im zweiten Schritt werden die invertierten S-t-tgds gebildet, indem eine Vertauschung von Rumpf und Kopf innerhalb einer S-t-tgd durchgeführt wird. Ebenfalls wird eine Anpassung der Quantoren realisiert. Infolgedessen wird auf die Bildung von disjunktiven Tgds und dessen Normalisierung verzichtet. Die invertierten S-t-tgds sind folglich definiert:

$$R(ma, ln, mo) \rightarrow \exists fn : St(ma, fn, ln) \land Pa(ma, mo)$$
$$Se(ma, ln, se) \rightarrow \exists g, mo : G(ma, mo, g, se)$$
$$R(ma, ln, mo) \rightarrow \exists g, se : G(ma, mo, g, se).$$

Anschließend werden die Disjunktiven Mengen gebildet, indem die invertierten S-t-tgds mit dem gleichen Relationssymbol $R \in T$ zusammengefasst werden. Diese Disjunktiven Mengen ω_1 und ω_2 bilden die Menge der Abhängigkeiten Ω:

$$\omega_1' = \{Se(ma, ln, se) \rightarrow \exists mo, g : (G(ma, mo, g, se)\}$$
$$\omega_2' = \{R(ma, ln, mo) \rightarrow \exists fn : St(ma, fn, ln) \land Pa(ma, mo),$$
$$R(ma, ln, mo) \rightarrow \exists g, se : G(ma, mo, g, se)\}.$$

Schließlich wird die invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega')$ generiert, die identisch zum Ergebnis des vorherigen Beispiels ist.

In diesem Abschnitt wurden zwei ähnliche Methoden für die Invertierung einer Menge von Abhängigkeiten bzw. einer Schemaabbildung betrachtet. Im folgenden Kapitel wird eine dieser Methoden (DirectAdaptedMaxExtendedRecovery-Algorithmus) verwendet, um eine eigene Invertierungsmethode für ProSA zu entwickeln.

Algorithmus 4 : DirectAdaptedMaxExtendedRecovery($\mathcal{M}$) [Zim21]

Input : Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ eine endliche Menge von S-t-tgds ist.

Output : *Adapted Strong Maximum Extended Recovery* $\mathcal{M}' = (T, S, \Omega)$ von $\mathcal{M}$, wobei Ω eine Menge von Mengen von S-t-tgds (Mengen von Disjunktiven-Mengen) in Logik erster Ordnung ist.

1. (Normalisiere die Konjunktionen in den Köpfen der S-t-tgds.) Erstelle Σ' aus Σ, indem jede S-t-tgd $\alpha \rightarrow (\beta_1 \wedge ... \wedge \beta_r)$ in Σ durch S-t-tgds der Form $\alpha \rightarrow \beta_1, ..., \alpha \rightarrow \beta_r$ ersetzt wird, bei denen β_i im Kopf atomar ist.

2. (Invertiere die S-t-tgds.) Füge für jede normalisierte S-t-tgd $\alpha \rightarrow \beta$ in Σ' die S-t-tgd $\beta \rightarrow \alpha$ zur Menge Σ'' hinzu. In Σ'' wird jede Variable in β zu einer allquantifizierten Variable, während jede Variable in α, die nicht in β vorkommt, zu einer existenzquantifizierten Variable wird.

3. (Sammle die S-t-tgds in Disjuntive-Mengen.) Sei $\Omega = \{\omega_1, ..., \omega_q\}$ eine Menge von Mengen von S-t-tgds (Menge von Disjunktiven-Mengen), mit $q =$ Anzahl der Relationsschemata R, über denen Ausdrücke in den Rümpfen der S-t-tgds in Σ'' existieren. Für jedes R, in den Ausdrücke der Rümpfe der S-t-tgd entsteht ein $\omega_i \in \Omega$, indem die S-t-tgds in Σ, mit im Rumpf-Ausdruck zusammengefasst werden.

Return : $\mathcal{M}' = (T, S, \Omega)$.

4 Konzept

In diesem Kapitel werden die Konvertierung und Invertierung von SQL-Anfragen betrachtet. Hierbei werden die einzelnen Operationen einer SQL-Anfrage und dessen Konvertierung und Invertierung als Menge von Abhängigkeiten (S-t-tgds, Tgds und Egds) aufgeführt (Abschnitt 4.1). Anschließend wird die Vollstängigkeit dieser Operationen geprüft und beschrieben, welche Operationen nicht als Menge von Abhängigkeiten umsetzbar sind (Abschnitt 4.1.4). Ebenfalls wird in diesem Kapitel ein eigener Algorithmus für die Invertierung von Abhängigkeiten entwickelt (Abschnitt 4.2). Dabei wird ein Vergleich von zwei Methoden für die Invertierung, die in Abschnitt 3.3.3 und 3.4 vorgestellt wurden, durchgeführt. Anschließend wird eine dieser Methoden (DirectAdaptedMaxExtendedRecovery-Algorithmus) als Grundlage für den eigenen Invertierungsalgorithmus verwendet.

4.1 SQL-Anfragen als S-t-tgds

Im Folgendem wird eine Betrachtung von SQL-Anfragen, und dessen Darstellung als Menge von Abhängigkeiten in Form von S-t-tgds und Tgds, durchgeführt. Dementsprechend werden die unterschiedlichen Operationen einer SQL-Anfrage und dessen Umsetzung als Menge von Abhängigkeiten aufgeführt (Abschnitt 4.1.1). Ebenfalls wird die invertierte Menge der Abhängigkeiten für die einzelnen Operationen betrachtet und aufgeführt. Für die Veranschaulichung der einzelnen Operationen der SQL-Anfragen dient die Tabelle 4.1. Hierbei wird ein Überblick über die Operationen gegeben und die Machbarkeit als Menge von Abhängigkeiten dargestellt. Beispielsweise ist ersichtlich, dass die Projektion ohne Duplikateneliminierung nicht als Menge von Abhängigkeiten umsetzbar ist, da die Abhängigkeiten (S-t-tgds, Tgds und Egds) mengenbasiert sind. Entsprechend werden in den mengenbasierten Abhängigkeiten die Duplikate immer automatisch eliminiert. In den folgenden Abschnitten werden die Konvertierung und Invertierung der einzelnen Operationen genauer aufgeführt. An dieser Stelle sei anzuführen, dass die folgenden Tabellen 4.1, 4.2 und 4.3 in Zusammenarbeit mit Ivo Kavisanczki, der in seiner Bachelorarbeit: *Erweiterung des ProSA-Parser* [Kav22] die Funktionalität des ProSA-Parser erweitert, erstellt worden sind.

Operation	Fall	Darstellbar	Anmerkung
Projektion	ohne Duplikateliminierung	−	Mengenbasiert
	mit Duplikateliminierung	++	Immer, da mengenbasiert
	SELECT ∗	++	
	Umbenennung in SELECT-Klausel[1]	++	
Selektion	Vergleich mit Konstante	++	
	Vergleich mit Attribut	++	
	AND	++	
	OR	++	
	NOT	+	
	IN/ANY/ALL mit Array	++	
	...		
Verbund	Natürlicher Verbund	++	
	Verbund mit USING	++	
	beliebiges Verbundprädikat	+	In ON-Expression oder WHERE-Klausel, wie bei Selektion
	Innerer Verbund	++	
	Äußerer Verbund	++	
	Kreuzprodukt	++	
Mengenoperationen	UNION	++	
	INTERSECT	++	
	EXCEPT	++	
Aggregation	MIN/MAX	++	
	AVG	++	
	COUNT	++	
	SUM	++	
	logische	?	
	statistische	?	
Geschachtelte Anfrage	EXISTS	++	
	IN	++	entspricht = ANY
	NOT IN	++	entspricht <> ALL
	ANY	++	
	ALL	++	
Sonstige	GROUP BY	?	
	HAVING	+	Ähnlich wie Selektion
	Umbenennung außerhalb der SELECT-Klausel[2]	?	
	Rekursive Anfragen	?	
	Window Functions	?	
	...		

Tabelle 4.1: Überblick, welche Operationen aus SQL inwieweit als S-t-tgds darstellbar sind [Kav22]:
− : SQL-Operation nicht als S-t-tgd darstellbar.
+ : SQL-Operation zum Teilen darstellbar.
++ : SQL-Operation vollständig darstellbar.
? : Möglicherweise darstellbar.

[1] d.h. Spaltenumbenennung im Ergebnis
[2] z.B. Aliasse für Ergebnisse von Unteranfragen oder Verbunden

4.1.1 Operationen

In diesem Abschnitt werden die Konvertierung und Invertierung von SQL-Anfragen dargestellt. Hierbei werden unterschiedliche SQL-Anfragen betrachtet, die in eine Menge von Abhängigkeiten in Form von S-t-tgds und Tgds konvertiert werden. Anschließend wird die Invertierung der Menge der Abhängigkeiten durchgeführt.

Konvertierung. Für die Veranschaulichung der Konvertierung der unterschiedlichen SQL-Anfragen, werden die SQL-Anfragen mit der Darstellung als Menge von Abhängigkeiten in den Tabellen 4.2 und 4.3 aufgeführt. Diese Aufteilung der Tabellen dient für eine bessere Übersicht, wobei die Tabelle 4.2 die Operationen für die Projektion, den Verbund und die Selektion enthält. Die Tabelle 4.3 enthält die Mengenoperationen, die Umbenennung und die verschachtelten Anfragen. Bei der Konvertierung als Menge von Abhängigkeiten bilden die Relationen R und S das Quellschema, welches auf das Zielschema mit der Relation Res abgebildet wird. Im Folgendem werden die einzelnen Operationen und dessen Darstellung als Menge von Abhängigkeiten beispielhaft betrachtet:

Bei der allgemeinen Projektion in Tabelle 4.2 werden ausschließlich die ausgewählten Attribute $a_{i_1}, \ldots, a_{i_k}$ auf die Ergebnisrelation Res abgebildet. Dabei bilden die Attribute $a_{i_1}, \ldots, a_{i_k}$ eine Teilmenge der vorhandenen Attribute $a_1, \ldots, a_n$ in der Relation R. Entsprechend beinhaltet die Relation Res lediglich die Attribute auf die projiziert wurde. Ein einfaches Beispiel für die Projektion wird durch folgende S-t-tgd veranschaulicht:

$$St(ma, ln, fn, sc) \rightarrow Res(ma).$$

Die S-t-tgd realisiert eine Projektion der Relation *Students* (kurz: *St*) auf das Attribut *MatricNo* (kurz: *ma*). Dementsprechend werden die Matrikelnummern (*MatricNo*) aller Studierenden in der Relation *Students* als Ergebnis ausgegeben bzw. auf die Ergebnisrelation *Res* abgebildet.

Im Fall einer Selektion (Tabelle 4.2) werden zusätzliche Vergleiche der Form $a_i \ \theta \ c$ zum Rumpf der S-t-tgd hinzugefügt, wobei $\theta \in \{<, >, \leq, \geq, \neq, =\}$ entspricht. Diese Vergleiche entsprechen den Selektionsbedingungen, wobei ausschließlich die Tupeln zur Ergebnisrelation *Res* hinzugefügt werden, die die Bedingungen erfüllen. Im folgenden Beispiel wird der Fall einer Selektion mit einer Konstante betrachtet, wobei folgende SQL-Anfrage konvertiert wird:

```
SELECT MatricNo
FROM Grades
WHERE Grade ≤ 4.0
```

In dieser SQL-Anfrage wird eine Selektion mit einer Konstante durchgeführt, wobei die Matrikelnummer (*MatricNo*) aller Studierender ausgegeben wird, die eine Note (*Grade*) besser oder gleich einer 4.0 erreicht haben. Folglich wird diese SQL-Anfrage durch

folgende S-t-tgd repräsentiert:

$$G(ma, mo, se, g) \wedge g \leq 4.0 \rightarrow Res(ma).$$

In dieser S-t-tgd wird eine Abbildung von der Relation *Grades* (kurz: G) auf die Ergebnisrelation *Res* durchgeführt, wobei alle Matrikelnummern (kurz: ma) der Studierenden mit einer Note (kurz: g) von 4.0 oder besser in die Ergebnisrelation *Res* aufgenommen werden. Dementsprechend enthält die S-t-tgd einen Vergleich $g \leq 4.0$ im Rumpf, um die Selektion des Attributs *Grade* (kurz: g) mit der Konstante 4.0 zu realisieren.

Im Fall einer Verbundoperation (Tabelle 4.2) wird die Umsetzung des (natürlichen) Verbundes, über eine Konjunktion im Rumpf der Abhängigkeiten realisiert. Dementsprechend wird bei einem natürlichen Verbund der Relation R und S, eine Konjunktion der Relationen R und S im Rumpf der S-t-tgd gebildet. Dabei entsprechen die Attribute d und e, in Tabelle 4.2, den jeweiligen Verbundattributen. Ein Beispiel eines natürlichen Verbundes wird durch folgende SQL-Anfrage dargestellt:

```
SELECT LastName, FirstName, Grade
FROM Students NATURAL JOIN Grades
```

In dieser Anfrage wird ein natürlicher Verbund der Relationen *Students* und *Grades* realisiert, wobei das Attribut *MatricNo* als Verbundattribut fungiert. Infolgedessen werden durch Anwendung dieser SQL-Anfrage, die Studierenden mit Nachname (*LastName*), Vorname *FirstName* und Note (*Grade*) ausgegeben, die an einer Prüfung teilgenommen haben. Diese SQL-Anfrage wird durch folgende S-t-tgd repräsentiert:

$$St(ma, ln, fn, sc) \wedge G(ma, mo, se, g) \rightarrow Res(ln, fn, g).$$

In dieser S-t-tgd wird eine Konjunktion der Relationen *Students* und *Grades* im Rumpf durchgeführt, dabei dient das Attribut *MatricNo* (kurz: ma) als Verbundattribut. Dementsprechend wird eine Verknüpfung von den Tupel der beiden Relationen *Students* und *Grades*, über gleicher Matrikelnummer (*MatricNo*), durchgeführt. Infolgedessen werden ausschließlich die verknüpften Tupel in die Ergebnisrelation *Res* mitaufgenommen.
Für die Umsetzung der äußeren Verbunde werden mehrere S-t-tgds benötigt. Dabei realisiert eine S-t-tgd die Umsetzung des natürlichen Verbundes (innerer Verbund), um eine Kombination der beiden (Verbund-)Relationen durchzuführen. Dagegen realisieren die anderen S-t-tgds die Übernahme der fehlenden Tupel, indem eine Relation auf die Ergebnisrelation abgebildet wird. Beispielsweise wird bei einem LEFT JOIN (Tabelle 4.2), der Form R LEFT JOIN S, die Kombination (natürlicher Verbund) der Relationen R und S in das Ergebnis mitaufgenommen. Des weiterem werden bei dem LEFT JOIN alle Tupel von der Relation R (linke Relation) in das Ergebnis übertragen. Dementsprechend realisiert eine S-t-tgd die Kombination der Relationen R und S und die andere S-t-tgd überträgt die fehlenden Tupel von der Relation R auf die Ergebnisrelation. Für die Veranschaulichung der Umsetzung der äußeren Verbunde wird folgende Anfrage als Beispiel betrachtet:

```
SELECT LastName, FirstName, Grade
FROM Students LEFT JOIN Grades
```

In dieser SQL-Anfrage wird ein LEFT JOIN der beiden Relationen *Students* und *Grades* realisiert. Hierbei werden alle Tupel der Relation *Students* im Anfrageergebnis mitaufgenommen. Ebenfalls wird der (natürliche) Verbund der beiden Relationen *Students* und *Grades* im Anfrageergebnis mitaufgenommen. Entsprechend enthält das Anfrageergebnis alle Studierenden der Relation *Students*, wobei alle Studierenden die an keiner Prüfung teilgenommen haben, einen Nullwert als Note (*Grade*) erhalten. Diese SQL-Anfrage wird durch folgende S-t-tgds repräsentiert:

$$St(ma, ln, fn, sc) \wedge G(ma, mo, se, g) \rightarrow Res(ln, fn, g)$$
$$St(ma, ln, fn, sc) \wedge \neg G(ma, mo, se, g) \rightarrow \exists g : Res(ln, fn, g).$$

In der ersten S-t-tgd wird der natürliche Verbund der beiden Relationen *Students* und *Grades* umgesetzt, wobei alle Tupel mit der gleichen Matrikelnummer (kurz: *ma*) kombiniert werden. Des weiterem werden durch die zweite S-t-tgd alle Tupel der Relation *Students* (kurz: *St*) auf die Ergebnisrelation *Res* abgebildet. Hierbei dient die Negation der Relation *Grades* (kurz: *G*), um zu verhindern, dass Studierende in die Ergebnisrelation *Res* aufgenommen werden, die bereits durch die erste S-t-tgd in die Ergebnisrelation geschrieben worden sind. Dementsprechend wird die Negation der Relation *Grades* als Bedingung verwendet, die verhindert, dass Duplikate in die Ergebnisrelation aufgenommen werden. Hierbei besagt die Bedingung, dass alle Studierende in die Ergebnisrelation aufgenommen werden, die keinen Verbundpartner in der Relation *Grades* besitzen.

Bei der Umbenennung in Tabelle 4.3 wird ein Attribut d in x umbenannt. Hierbei wird die Umsetzung der Umbenennung als S-t-tgd mithilfe eines Gleichheitsprädikat $d = x$ durchgeführt. Dementsprechend wird durch die Anwendung des Gleichheitsprädikat $d = x$ eine Ersetzung des Attributs d durch ein neues Attribut x realisiert.

Im Fall einer verschachtelten Anfrage (Tabelle 4.3) werden Tgds und S-t-tgds verwendet, um die Anfrage als Menge von Abhängigkeiten darzustellen. Hierbei dienen die Tgds für die Umsetzung der Unteranfragen, wobei die Tgds ein Zwischenergebnis liefern. Anschließend wird die (äußere) Anfrage als S-t-tgd umgesetzt, indem das Zwischenergebnis der Unteranfrage/Tgd im Rumpf verwendet wird. Dementsprechend liefert die S-t-tgd das eigentliche Ergebnis der Anfrage, welches mithilfe des Zwischenergebnisses gebildet wird. Eine Ausnahme von dieser Methodik ist die Operation EXISTS, wobei die Umsetzung einer S-t-tgd genügt.
Für die Veranschaulichung der Umsetzung der verschachtelten Anfragen wird im Folgendem ein Beispiel durchgeführt. Hierbei wird folgenden verschachtelte SQL-Anfrage betrachtet:

```
SELECT * FROM Grades
WHERE Grade < ANY(
SELECT Grade FROM Grades
WHERE MatricNo = 1 AND ModulNo = 002)
AND ModulNo = 002
```

Diese verschachtelte Anfrage bestimmt alle Studierenden die eine bessere Note im zweitem Modul ($ModulNo = 002$) besitzen, als der Studierende mit der Matrikelnummer 1 ($MatricNo = 1$). Dementsprechend wird diese SQL-Anfrage durch folgende Menge von Abhängigkeiten umgesetzt:

$$\begin{aligned} &\text{Tgd:} && G(ma, mo, se, g) \wedge ma = 1 \wedge mo = 002 \rightarrow Res'(g) \\ &\text{S-t-tgd:} && G(ma, mo, se, g_j) \wedge Res'(g_i) \wedge g_j < g_i \wedge mo = 002 \rightarrow Res'(ma, mo, se, g_j). \end{aligned}$$

Bei der Umsetzung der verschachtelten Anfragen als Menge von Abhängigkeiten werden die Unteranfragen als Tgd realisiert. Entsprechend wird die Unteranfrage in diesem Beispiel durch die erste Tgd dargestellt. In dieser Tgd wird die Note, die der Studierende mit der Matrikelnummer 1 im zweitem Modul erreicht hat, auf ein Zwischenergebnis Res' übertragen. Anschließend wird das Zwischenergebnis Res' in der S-t-tgd verwendet, um die eigentliche Ergebnisrelation Res zu bilden. Hierbei führt die S-t-tgd den Vergleich der jeweiligen Noten ($g_j < g_i$), aus dem Zwischenergebnis und der Relation *Grades*, durch und übertragt alle Tupel mit einer besseren Note als die vom Zwischenergebnis in die Ergebnisrelation Res.

Bei der Konvertierung der Mengenoperationen (Tabelle 4.3) existiert kein allgemeines Prinzip, welches bei allen Mengenoperationen angewendet werden kann. Entsprechend müssen die jeweiligen Mengenoperationen separat betrachtet werden. Beispielsweise wird bei der Anwendung von der Operation UNION, die eine Vereinigung von zwei Relationen umsetzt, zwei S-t-tgds benötigt. Hierbei dienen die S-t-tgds, um die einzelnen Relationen auf die Ergebnisrelation abzubilden. Ein einfaches Beispiel für die Vereinigung von zwei Relationen ist durch folgende SQL-Anfrage veranschaulicht:

```
SELECT MatricNo, LastName FROM Students
UNION
SELECT MatricNo, Semester FROM Grades
```

Diese SQL-Anfrage realisiert eine Vereinigung der Relationen *Students* und *Grades*. Bei einer Vereinigung müssen beide Relationen die gleiche Anzahl an Attributen besitzen. Außerdem müssen die jeweiligen Attribute den gleichen Datentyp aufweisen. Entsprechend wird bei beiden Relationen eine Projektion durchgeführt, um diese beiden Bedingungen der Vereinigung zu erfüllen. Die Umsetzung dieser SQL-Anfrage als Menge von Abhängigkeiten, ist durch folgende S-t-tgds realisiert:

$$\begin{aligned} St(ma, ln, fn, sc) &\rightarrow Res(ma, ln) \\ G(ma, mo, se, g) &\rightarrow Res(ma, se). \end{aligned}$$

Diesbezüglich wird durch die erste S-t-tgd die Relation *Students* auf die Ergebnisrelation *Res* abgebildet, dabei werden die Attribute *MatricNo* und *LastName* in die Ergebnisrelation *Res* mitaufgenommen. Des weiterem wird durch die Anwendung der zweitem S-t-tgd die Relation *Grades* auf die Ergebnisrelation *Res* abgebildet. Dementsprechend enthält die Ergebnisrelation *Res* nach Anwendung der beiden S-t-tgds die Vereinigung der beiden Relationen, da von beiden Relationen die entsprechenden Tupel bzw. Attribute auf die Ergebnisrelation übertragen wurden.
Für die Umsetzung der Operation EXPECT (Tabelle 4.3), die die Differenz von zwei (Teil-)Anfragen berechnet, wird eine negierte Relation im Rumpf verwendet. Dabei wird bei der Berechnung der Differenz, der Form R EXPECT S, die Relation S negiert. Diese negierte Relation S dient als Bedingung, dass ausschließlich Tupel der Relation R die nicht in der Relation S vorhanden sind, in die Ergebnisrelation *Res* übertragen werden. Als Beispiel wird folgende SQL-Anfrage betrachtet:

```
SELECT MatricNo FROM Students
EXPECT
SELECT MatricNo FROM Grades
```

In diesem Beispiel wird die Differenz der Matrikelnummern (*MatricNo*) von den beiden Relationen *Students* und *Grades* gebildet. Entsprechend werden im Anfrageergebnis alle Matrikelnummern der Studierenden ausgegeben, die keine Prüfung absolviert haben. Folglich wird folgende S-t-tgd gebildet:

$$St(ma, ln, fn, sc) \wedge \neg G(ma, mo, se, g) \rightarrow Res(ma).$$

In dieser S-t-tgd dient die Negation der Relation *Grades* als Bedingung, dass ausschließlich Matrikelnummern in die Ergebnisrelation *Res* aufgenommen werden, die nicht in der Relation *Grades* vorhanden sind. Entsprechend wird durch diese S-t-tgd die Differenz der beiden Relationen *Students* und *Grades* realisiert.
Die Umsetzung der Operation INTERSECT (Tabelle 4.3), die die Berechnung des Durchschnittes von zwei Relationen bzw. Anfragen durchführt, erfolgt analog zu der Umsetzung der Operation EXPECT. Allerdings mit dem Unterschied, dass die zweite Relation nicht negiert wird.

Bei der Anwendung von Operationen auf ein Array (Tabelle 4.3), dass eine Mengen von Konstanten/Werten entspricht, wird das Array als eigene Relation umgesetzt. Dementsprechend wird ein Array $(x_1, ..., x_n)$ als eine Relation X realisiert, wobei die Relation X ausschließlich aus einem Attribut besteht. Infolgedessen repräsentiert jeder einzelne Wert im Array $(x_1, ..., x_n)$ ein Tupel in der Relation X. Anschließend wird mithilfe der Relation X die Operation als S-t-tgd umgesetzt. Ein Beispiel wird durch folgende SQL-Anfrage gegeben:

```
SELECT * FROM Students
WHERE MatricNo IN (1,2,3,4)
```

Bei der Anwendung der SQL-Anfrage werden alle Studierenden mit den Matrikelnummern, die im Array $(1, 2, 3, 4)$ vorhanden sind, ausgeben. Entsprechend wird folgende S-t-tgd verwendet, um diese Anfrage darzustellen:

$$St(ma, ln, fn, sc) \wedge X(ma) \rightarrow Res(ma, ln, fn, sc).$$

Diesbezüglich wird eine neue Relation X erstellt, die als Attribut die Matrikelnummer (kurz: ma) enthält. Ebenfalls enthält die neue Relation X die einzelnen Werte des Arrays $(1, 2, 3, 4)$ als Tupel. Anschließend wird ein Verbund der beiden Relation St und X realisiert, indem eine Konjunktion im Rumpf definiert wird. Entsprechend werden ausschließlich die Matrikelnummern in die Ergebnisrelation Res übernommen, die in beiden Relationen vorhanden sind.

Operation	Anfrage in SQL	Darstellung als S-t-tgds
Projektion:		
Projektion allgemein	SELECT $a_{i_1}, \ldots, a_{i_k}$ FROM R	$R(a_1, \ldots, a_n) \rightarrow Res(a_{i_1}, \ldots, a_{i_k})$
Projektion auf alle Attribute	SELECT $*$ FROM R	$R(a_1, \ldots, a_n) \rightarrow Res(a_1, \ldots, a_n)$
Selektion:		
Vergleich mit Konstante	SELECT $*$ FROM R WHERE $a_i\ \theta\ c$	$R(a_1, \ldots, a_n) \wedge a_i\ \theta\ c$ $\rightarrow Res(a_1, \ldots, a_n)$
Vergleich mit Attribute	SELECT $a_{i_1}, \ldots, a_{i_k}$ FROM R, S WHERE $a_i\,\theta\,b_j$	$R(a_1, \ldots, a_n) \wedge S(b_1, \ldots, b_n) \wedge a_i\,\theta\,b_j$ $\rightarrow Res(a_{i_1}, \ldots, a_{i_k})$
AND	SELECT $*$ FROM R WHERE $a_i\,\theta\,c_1$ AND $a_j\,\theta\,c_2$	$R(a_1, \ldots, a_n) \wedge a_i\,\theta\,c_1 \wedge a_j\,\theta\,c_2$ $\rightarrow Res(a_1, \ldots, a_n)$
OR	SELECT $*$ FROM R WHERE $a_i\,\theta\,c_1$ OR $a_j\,\theta\,c_2$	$R(a_1, \ldots, a_n) \wedge a_i\,\theta\,c_1$ $\rightarrow Res(a_1, \ldots, a_n)$, $R(a_1, \ldots, a_n) \wedge a_j\,\theta\,c_2$ $\rightarrow Res(a_1, \ldots, a_n)$
NOT	SELECT $*$ FROM R WHERE NOT $a_i\,\theta\,c$	$R(a_1, \ldots, a_n) \wedge \neg(a_i\,\theta\,c)$ $\rightarrow Res(a_1, \ldots, a_n)$
BETWEEN	SELECT $*$ FROM R WHERE a_i BETWEEN c_1 AND c_2	$R(a_1, \ldots, a_n) \wedge a_i >= c_1 \wedge a_i <= c_2$ $\rightarrow Res(a_1, \ldots, a_n)$
Verbund:		
natürlicher Verbund	SELECT $*$ FROM R NATURAL JOIN S	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, d)$ $\rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d)$
Kreuzprodukt	SELECT $*$ FROM R CROSS JOIN S	$R(a_1, \ldots, a_n) \wedge S(b_1, \ldots, b_m)$ $\rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m)$
JOIN ON	SELECT $*$ FROM R JOIN S ON $R.d = S.e$	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e)$ $\wedge\, d = e \rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$
USING	SELECT $*$ FROM R JOIN S USING (d_1, d_2)	$R(a_1, \ldots, a_n, d_1, d_2) \wedge S(b_1, \ldots, b_m, d_1, d_2)$ $\rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d_1, d_2)$
LEFT JOIN	SELECT $*$ FROM R LEFT JOIN S ON $R.d = S.e$	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e)$ $\wedge\, d = e \rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$, $R(a_1, \ldots, a_n, d) \wedge \neg S(b_1, \ldots, b_m, d)$ $\rightarrow \exists b_1, \ldots, b_n : Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d)$
RIGHT JOIN	SELECT $*$ FROM R RIGHT JOIN S ON $R.d = S.e$	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e)$ $\wedge\, d = e \rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$, $S(b_1, \ldots, b_m, e) \wedge \neg R(a_1, \ldots, a_n, e)$ $\rightarrow \exists a_1, \ldots, a_n : Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$
FULL OUTER JOIN	SELECT $*$ FROM R FULL OUTER JOIN S ON $R.d = S.e$	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e)$ $\wedge\, d = e \rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$, $R(a_1, \ldots, a_n, d) \wedge \neg S(b_1, \ldots, b_m, d)$ $\rightarrow \exists b_1, \ldots, b_n : Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d)$, $S(b_1, \ldots, b_n, e) \wedge \neg R(a_1, \ldots, a_n, e)$ $\rightarrow \exists a_1, \ldots, a_n : Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$

Tabelle 4.2: Darstellung von Konzepten aus SQL als S-t-tgds

Operation	Anfrage in SQL	Darstellung als S-t-tgds und Tgds
Umbenennung	SELECT $a_1, \ldots, a_n, d$ AS x FROM R	$R(a_1, \ldots, a_n, d) \wedge d = x$ $\rightarrow Res(a_1, \ldots, a_n, x)$
Geschachtelte Anfragen:		
IN	SELECT $*$ FROM R WHERE a_i IN (SELECT b_j FROM S WHERE $b_k \, \theta \, c$)	$S(b_1, \ldots, b_n) \wedge b_k \, \theta \, c \rightarrow Res'(b_j)$, $R(a_1, \ldots, a_n) \wedge Res'(b_j) \wedge a_i = b_j$ $\rightarrow Res(a_1, \ldots, a_n)$
ANY	SELECT $*$ FROM R WHERE $a_i \, \theta$ ANY (SELECT b_j FROM S WHERE $b_k \, \theta \, c$)	$S(b_1, \ldots, b_n) \wedge b_k \, \theta \, c \rightarrow Res'(b_j)$, $R(a_1, \ldots, a_n) \wedge Res'(b_j) \wedge a_i \, \theta \, b_j$ $\rightarrow Res(a_1, \ldots, a_n)$
NOT IN	SELECT $*$ FROM R WHERE a_i NOT IN (SELECT b_j FROM S WHERE $b_k \, \theta \, c$)	$S(b_1, \ldots, b_n) \wedge b_k \, \theta \, c \rightarrow Res'(b_j)$, $R(a_1, \ldots, a_n) \wedge \neg Res'(a_i)$ $\rightarrow Res(a_1, \ldots, a_n)$
EXISTS	SELECT $*$ FROM R WHERE EXISTS (SELECT $*$ FROM S WHERE $R.a_i \, \theta \, S.b_k$)	$R(a_1, ..., a_n) \wedge S(b_1, ..., b_n)$ $\wedge a_i \, \theta \, b_k \rightarrow Res(a_1, ..., a_n)$
Mengenoperationen:		
UNION	SELECT $a_1, \ldots, a_n$ FROM R UNION SELECT $a_1, \ldots, a_n$ FROM S	$R(a_1, \ldots, a_n) \rightarrow Res(a_1, \ldots, a_n)$, $S(a_1, \ldots, a_n) \rightarrow Res(a_1, \ldots, a_n)$
EXCEPT	SELECT $a_1, \ldots, a_n$ FROM R EXCEPT SELECT $a_1, \ldots, a_n$ FROM S	$R(a_1, \ldots, a_n) \wedge \neg S(a_1, \ldots, a_n)$ $\rightarrow Res(a_1, \ldots, a_n)$
INTERSECT	SELECT $a_1, \ldots, a_n$ FROM R INTERSECT SELECT $a_1, \ldots, a_n$ FROM S	$R(a_1, \ldots, a_n) \wedge S(a_1, \ldots, a_n)$ $\rightarrow Res(a_1, \ldots, a_n)$
Arrays:		
IN	SELECT $*$ FROM R WHERE a_i IN $(x_1, ..., x_n)$	$R(a_1, ..., a_n) \wedge X(a_i)$ $\rightarrow Res(a_1, ..., a_n)$
ANY	SELECT $*$ FROM R WHERE $a_i \, \theta$ ANY $(x_1, ..., x_n)$	$R(a_1, ..., a_n) \wedge X(x_i) \wedge a_i \, \theta \, x_i$ $\rightarrow Res(a_1, ..., a_n)$
NOT IN	SELECT $*$ FROM R WHERE a_i NOT IN $(x_1, ..., x_n)$	$R(a_1, ..., a_n) \wedge \neg X(a_i)$ $\rightarrow Res(a_1, ..., a_n)$

Tabelle 4.3: Darstellung von Konzepten aus SQL als S-t-tgds

Invertierung. Für die Veranschaulichung der Invertierung der SQL-Anfragen dienen die Tabellen 4.4 und 4.5. Die Tabelle 4.4 beinhaltet die invertierten Abhängigkeiten (S-t-tgds) für die Projektion, Selektion und für den Verbund. Dagegen beinhaltet die Tabelle 4.5 die invertierten Abhängigkeiten (S-t-tgds) für die Mengenoperationen, für die Umbenennung und für die verschachtelten Anfragen. Im Allgemeinem wird bei der Invertierung eine Rückabbildung von der Ergebnisrelation *Res* auf die ursprünglichen (Quell-)Relationen R und S durchgeführt. Eine allgemeine Methode bei der Invertierung von Abhängigkeiten, ist eine Vertauschung von Rumpf und Kopf durchzuführen. Entsprechend wird die Pfeilrichtung der gegebenen Abhängigkeiten umgekehrt und die Quantoren angepasst, um die invertierten Abhängigkeiten zu generieren. Hierbei werden alle Variablen die im Kopf, aber nicht im Rumpf, einer invertierten Abhängigkeit existenzquantifiziert und die restlichen Variablen werden allquantifiziert. Eine genaue Methode für die Invertierung von Abhängigkeiten wird in Abschnitt 4.2 beschrieben. Im Folgendem wird die Invertierung der Operationen separat betrachtet:

Bei der Invertierung der Projektion (Tabelle 4.4) wird eine einfache Vertauschung von Rumpf und Kopf durchgeführt. Dabei werden alle Variablen, die nicht bei der Projektion ausgewählt worden sind, existenzquantifiziert und die restlichen Variablen werden allquantifiziert. Ein einfaches Beispiel wird durch folgende S-t-tgd dargestellt:

$$St(ma, ln, fn, sc) \rightarrow Res(ma).$$

Folglich wird durch die S-t-tgd eine Projektion auf die Matrikelnummern der Studierenden, in der Relation *Students*, durchgeführt. Die entsprechende invertierte S-t-tgd ist folglich definiert:

$$Res(ma) \rightarrow \exists ln, fn, sc : St(ma, ln, fn, sc).$$

Die invertierte S-t-tgd realisiert eine Rückabbildung von der Ergebnisrelation *Res* auf die Quellrelation *Students*. Hierbei werden alle Variablen, die nicht in der Ergebnisrelation *Res* vorhanden sind, existenzquantifiziert und die restlichen Variablen (*MatricNo*) allquantifiziert. Entsprechend werden die Variablen/Attribute *LastName*, *FirstName* und *StudyCourse* im Kopf der invertierten S-t-tgd existenzquantifiziert. Hinsichtlich einer Berechnung einer minimalen Teildatenbank mithilfe der invertierten S-t-tgd würde ein Verlust der Informationen der existenzquantifizierten Variablen stattfinden, da die Informationen nicht im Rumpf bzw. in der Ergebnisrelation vorhanden sind. Dabei würden die existenzquantifizierten Variablen bei der Berechnung der minimalen Teildatenbank durch Nullwerte ersetzt werden.

Im Fall der Invertierung einer Selektion (Tabelle 4.4) wird ebenfalls eine einfache Vertauschung bzw. die Umkehrung der Pfeilrichtung durchgeführt. Hierbei werden die entsprechende Vergleiche, die in der S-t-tgd die Selektion darstellen, bei der invertierten S-t-tgd in den Kopf mitübernommen. Ein Beispiel wird durch folgende S-t-tgd veranschaulicht:

$$G(ma, mo, se, g) \wedge g \leq 4.0 \rightarrow Res(ma).$$

In dieser S-t-tgd wird eine Selektion des Attributs *Grade* durchgeführt, wobei ausschließlich Matrikelnummern von Studierenden mit einer Note von 4.0 oder besser in der Ergebnisrelation *Res* aufgenommen werden. Folglich bildet die folgende invertierte S-t-tgd das Ergebnis der Invertierung des Beispiels:

$$Res(ma) \rightarrow \exists mo, se, g : G(ma, mo, se, g) \wedge g \leq 4.0$$

Entsprechend wird eine Vertauschung von Rumpf und Kopf und eine Anpassung der Quantoren durchgeführt, um die invertierte S-t-tgd zu bilden. Ebenfalls wird der Vergleich $g \leq 4.0$ im Kopf der invertierten S-t-tgd mitübernommen.

Bei der Invertierung einer Verbundoperation (Tabelle 4.4) erfolgt das gleiche Prinzip, mit der Vertauschung und Umkehrung der Pfeilrichtung, um die invertierten S-t-tgds zu definieren. Als Beispiel wird folgende S-t-tgd betrachtet:

$$St(ma, ln, fn, sc) \wedge G(ma, mo, se, g) \rightarrow Res(ln, fn, g).$$

Diese S-t-tgd realisiert einen natürlichen Verbund der beiden Relationen *Students* und *Grades*, über dem Verbundattribut *MatricNo*. Entsprechend wird durch die Invertierung folgende S-t-tgd gebildet:

$$Res(ln, fn, g) \rightarrow \exists ma, mo, se, g : St(ma, ln, fn, sc) \wedge G(ma, mo, se, g).$$

Diesbezüglich wird eine Rückabbildung von der Ergebnisrelation *Res* auf die beiden Quellrelationen *Students* und *Grades* durchgeführt. Hierbei wird die Variable/Attribut *MatricNo* existenzquantifiziert, da dieses Attribut nicht in der Ergebnisrelation *Res* enthalten ist.
Für die Umsetzung der äußeren Verbunde als Menge von Abhängigkeiten werden mehrere S-t-tgds benötigt. Dementsprechend liefert die Invertierung der äußere Verbunde, ebenfalls mehrere invertierte S-t-tgds zurück. Ein Beispiel eines äußeren Verbundes wird durch folgende S-t-tgds repräsentiert:

$$St(ma, ln, fn, sc) \wedge G(ma, mo, se, g) \rightarrow Res(ln, fn, g)$$
$$St(ma, ln, fn, sc) \wedge \neg G(ma, mo, se, g) \rightarrow \exists g : Res(ln, fn, g).$$

Hierbei wird durch die S-t-tgds die Umsetzung eines LEFT JOIN zwischen den beiden Relationen *Students* und *Grades* realisiert. Infolgedessen führt die erste S-t-tgd den natürlichen Verbund (inneren Verbund) der beiden Relationen *Students* und *Grades* durch. Anschließend wird durch die Anwendung der zweiten S-t-tgds die fehlenden Tupeln, aus der Relation *Students*, in die Ergebnisrelation *Res* übertragen. Bei der Invertierung dieser S-t-tgds werden folgende invertierte S-t-tgds gebildet:

$$Res(ln, fn, g) \rightarrow \exists ma : St(ma, ln, fn, sc) \wedge G(ma, mo, se, g)$$
$$Res(ln, fn, g) \rightarrow \exists ma : St(ma, ln, fn, sc).$$

Bei der Invertierung der ersten S-t-tgd wird eine einfache Vertauschung von Kopf und Rumpf und eine Anpassung der Quantoren durchgeführt. Die zweite S-t-tgd wird ebenfalls durch diese Vertauschung generiert. Allerdings wird bei der zweiten (invertierten) S-t-tgd die negierte Relation G entfernt. Diese Entfernung der negierten Relationen wird durchgeführt, da die negierten Relationen innerhalb einer Abhängigkeit ausschließlich als Bedingung fungieren. Entsprechend können diese Bedingungen/negierten Relationen entfernt werden, ohne einen Informationsverlust bei der Berechnung der minimalen Teildatenbank zu generieren.

Die Invertierung bei einer Umbenennung (Tabelle 4.5) erfolgt ebenfalls durch die Vertauschung von Kopf und Rumpf und durch die Anpassung der Quantoren. Dabei wird der Vergleich, der die Umbenennung realisiert, auch in den Kopf der invertierten S-t-tgd übertragen. Entsprechend erfolgt die Invertierung der Umbenennung analog zur Invertierung einer Selektion.

Bei der Umsetzung der Operationen für die geschachtelten Anfragen (Tabelle 4.5), als Menge von Abhängigkeiten, wird eine Tgd und eine S-t-tgd definiert. Hierbei dienen die Tgds für die Umsetzung der Unteranfragen, wobei ein Zwischenergebnis generiert wird. Anschließend wird dieses Zwischenergebnis in der S-t-tgd verwendet, um das eigentliche Ergebnis zu erzeugen. Bei der Invertierung der Abhängigkeiten für die verschachtelte Anfragen würde eine einfache Vertauschung von Rumpf und Kopf in der S-t-tgd, eine Abbildung von der Ergebnisrelation auf das Zwischenergebnis der Unteranfrage/Tgd realisieren. Allerdings ist diese Abbildung auf ein Zwischenergebnis für die Berechnung der minimalen Teildatenbank nicht sinnvoll, da das Zwischenergebnis nicht im Quellschema enthalten ist. Dementsprechend muss bei der Invertierung immer eine Abbildung vom Anfrageergebnis auf die eigentlichen Quellrelationen realisiert werden. Folglich wird bei der Invertierung der verschachtelten Anfragen ein Rückverfolgung der Tgd auf die eigentlichen Quellrelationen durchgeführt, da die Tgds eine Veränderung der Quellrelation durchführen, um das Zwischenergebnis zu generieren. Anschließend wird eine invertierte S-t-tgd mit den eigentlichen Quellrelationen gebildet. Ein Beispiel für die Invertierung einer verschachtelten Anfrage wird durch folgende SQL-Anfrage gegeben:

```
SELECT * FROM Students
WHERE MatricNo IN (
SELECT MatricNo FROM Grades
WHERE Grade <= 4.0)
```

Bei der Anwendung dieser SQL-Anfrage werden alle Studierende mit einer Note von 4.0 oder besser, als Anfrageergebnis ausgegeben. Folglich wird diese SQL-Anfrage als folgende Menge von Abhängigkeiten umgesetzt:

Tgds: $G(ma, mo, se, g) \wedge g \leq 4.0 \rightarrow Res'(ma)$

S-t-tgds: $St(ma_1, ln, fn, sc) \wedge Res'(ma_2) \wedge ma_1 = ma_2 \rightarrow Res(ma_1, ln, fn, sc).$

In dieser Menge von Abhängigkeiten wird die Umsetzung der Unteranfrage durch die Tgd realisiert und ein Zwischenergebnis Res' gebildet. Anschließend wird in der S-t-tgd das Zwischenergebnis Res' verwendet, um die eigentliche Ergebnisrelation Res zu generieren. Bei der Invertierung der Menge von Abhängigkeiten wird folgende invertierte S-t-tgd gebildet:

$$Res(ma, ln, fn, sc) \rightarrow \exists mo, se, g : St(ma, ln, fn, sc) \wedge G(ma, mo, se, g).$$

Diese invertierte S-t-tgd realisiert eine Rückabbildung von Anfrageergebnis Res auf die eigentlichen Quellrelationen *Students* und *Grades*. Entsprechend wird eine Rückverfolgung der Tgd auf die eigentliche Quellrelation *Grades* durchgeführt. Folglich wird keine Rückabbildung auf das Zwischenergebnis Res' realisiert, da das Zwischenergebnis nicht in der Quellinstanz bzw. in der Datenbank vorhanden ist und keine Relevanz bei der Berechnung der minimalen Teildatenbank besitzt.

Bei der Invertierung der Mengenoperationen (Tabelle 4.5) wird ebenfalls eine einfache Vertauschung von Kopf und Rumpf durchgeführt. Ebenfalls wird bei der Invertierung der Operation EXPECT wieder die negierte Relation, aus dem Kopf der invertierten S-t-tgd, entfernt.

Für die Umsetzung der Operationen auf ein Array (Tabelle 4.5) wird ein neue Relation X in der S-t-tgd erstellt, die die Werte des Arrays als Tupel besitzt. Entsprechend wird diese neue Relation hinzugefügt, um das entsprechende Array in einer S-t-tgd zu verwenden. Diesbezüglich kann die neue Relation X bei der Invertierung aus dem Kopf der invertierten S-t-tgd entfernt werden, da die neue Relation X nicht im Quellschema bzw. in der Datenbank vorhanden ist. Ein Beispiel wird durch folgende S-t-tgd veranschaulicht:

$$St(ma, ln, fn, sc) \wedge X(ma) \rightarrow Res(ma, ln, fn, sc).$$

Diese S-t-tgd entspricht einer Umsetzung der Operation IN auf ein Array. Hierbei werden alle Studierende in die Ergebnisrelation Res aufgenommen, dessen Matrikelnummer im Array bzw. in der Relation X vorhanden sind. Die folgende invertierte S-t-tgd bildet das Ergebnis der Invertierung:

$$Res(ma, ln, fn, sc) \rightarrow St(ma, ln, fn, sc).$$

In dieser invertierten S-t-tgd wurde eine Vertauschung von Kopf und Rumpf vorgenommen, wobei die Relation X aus dem Kopf entfernt wurde. Entsprechend wird in dieser invertierten S-t-tgd alle Studierende in der Ergebnisrelation Res zurück auf die Relation *Students* abgebildet. Folglich besitzt die Relation X bei der Berechnung der minimalen Teildatenbank keine Relevanz, da ausschließlich die Studierende mit einer Matrikelnummer die im Array vorhanden ist, in der Ergebnisrelation Res existieren.

Operation	Darstellung als S-t-tgds	Invertierung der S-t-tgds
Projektion:		
Projektion allgemein	$R(a_1, \ldots, a_n) \rightarrow Res(a_{i_1}, \ldots, a_{i_k})$	$Res(a_{i_1}, \ldots, a_{i_k}) \rightarrow \exists a_i \notin \{a_{i_1}, \ldots, a_{i_k}\} : R(a_1, \ldots, a_n)$
Projektion auf alle Attribute	$R(a_1, \ldots, a_n) \rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow (R(a_1, \ldots, a_n)$
Selektion:		
Vergleich mit Konstante	$R(a_1, \ldots, a_n) \wedge a_i\ \theta\ c$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $R(a_1, \ldots, a_n) \wedge a_i\ \theta\ c$
Vergleich mit Attribute	$R(a_1, \ldots, a_n) \wedge S(b_1, \ldots, b_n) \wedge a_i\, \theta\, b_j$ $\rightarrow Res(a_{i_1}, \ldots, a_{i_k})$	$Res(a_{i_1}, \ldots, a_{i_k}) \rightarrow$ $R(a_1, \ldots, a_n) \wedge S(b_1, \ldots, b_n) \wedge a_i\, \theta\, b_j$
AND	$R(a_1, \ldots, a_n) \wedge a_i\, \theta\, c_1 \wedge a_j\, \theta\, c_2$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $R(a_1, \ldots, a_n) \wedge a_i\, \theta\, c_1 \wedge a_j\, \theta\, c_2$
OR	$R(a_1, \ldots, a_n) \wedge a_i\, \theta\, c_1$ $\rightarrow Res(a_1, \ldots, a_n)$, $R(a_1, \ldots, a_n) \wedge a_j\, \theta\, c_2$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $R(a_1, \ldots, a_n) \wedge a_i\, \theta\, c_1$, $Res(a_1, \ldots, a_n) \rightarrow$ $R(a_1, \ldots, a_n) \wedge a_j\, \theta\, c_2$
NOT	$R(a_1, \ldots, a_n) \wedge \neg(a_i\, \theta\, c)$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $R(a_1, \ldots, a_n) \wedge \neg(a_i\, \theta\, c)$
BETWEEN	$R(a_1, \ldots, a_n) \wedge a_i >= c_1 \wedge a_i <= c_2$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $R(a_1, \ldots, a_n) \wedge a_i >= c_1 \wedge a_i <= c_2$
Verbund:		
natürlicher Verbund	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, d)$ $\rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d)$	$Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d) \rightarrow$ $R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, d)$
Kreuzprodukt	$R(a_1, \ldots, a_n) \wedge S(b_1, \ldots, b_m)$ $\rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m)$	$Res(a_1, \ldots, a_n, b_1, \ldots, b_m) \rightarrow$ $R(a_1, \ldots, a_n) \wedge S(b_1, \ldots, b_m)$
JOIN ON	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e)$ $\wedge\, d = e \rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$	$Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e) \rightarrow$ $R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e) \wedge\ d = e$
USING	$R(a_1, \ldots, a_n, d_1, d_2) \wedge S(b_1, \ldots, b_m, d_1, d_2)$ $\rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d_1, d_2)$	$Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d_1, d_2) \rightarrow$ $R(a_1, \ldots, a_n, d_1, d_2) \wedge S(b_1, \ldots, b_m, d_1, d_2)$
LEFT JOIN	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e)$ $\wedge\, d = e \rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$, $R(a_1, \ldots, a_n, d) \wedge \neg S(b_1, \ldots, b_m, d)$ $\rightarrow \exists b_1, \ldots, b_n : Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d)$	$Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e) \rightarrow$ $R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e) \wedge\ d = e$, $Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d) \rightarrow$ $R(a_1, \ldots, a_n, d)$
RIGHT JOIN	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e)$ $\wedge\, d = e \rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$, $S(b_1, \ldots, b_n, e) \wedge \neg R(a_1, \ldots, a_n, e)$ $\rightarrow \exists a_1, \ldots, a_n : Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$	$Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e) \rightarrow$ $R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e) \wedge\ d = e$, $Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e) \rightarrow$ $S(b_1, \ldots, b_n, e)$
FULL OUTER JOIN	$R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e)$ $\wedge\, d = e \rightarrow Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$, $R(a_1, \ldots, a_n, d) \wedge \neg S(b_1, \ldots, b_m, d)$ $\rightarrow \exists b_1, \ldots, b_n : Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d)$, $S(b_1, \ldots, b_n, e) \wedge \neg R(a_1, \ldots, a_n, e)$ $\rightarrow \exists a_1, \ldots, a_n : Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e)$	$Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e) \rightarrow$ $R(a_1, \ldots, a_n, d) \wedge S(b_1, \ldots, b_m, e) \wedge\ d = e$, $Res(a_1, \ldots, a_n, b_1, \ldots, b_m, d) \rightarrow$ $R(a_1, \ldots, a_n, d)$, $Res(a_1, \ldots, a_n, b_1, \ldots, b_m, e) \rightarrow$ $S(b_1, \ldots, b_n, e)$

Tabelle 4.4: Darstellung der Invertierung der SQL-Operationen

Operation	Darstellung als S-t-tgds	Invertierung der S-t-tgds
Umbenennung	$R(a_1, \ldots, a_n, d) \wedge d = x$ $\rightarrow Res(a_1, \ldots, a_n, x)$	$Res(a_1, \ldots, a_n, x) \rightarrow$ $R(a_1, \ldots, a_n, d) \wedge d = x$
Geschachtelte Anfragen:		
IN	$S(b_1, \ldots, b_n) \wedge b_k \,\theta\, c \rightarrow Res'(b_j)$, $R(a_1, \ldots, a_n) \wedge Res'(b_j) \wedge a_i = b_j$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $\exists b_1, \ldots, b_n : S(b_1, \ldots, b_n) \wedge$ $R(a_1, \ldots, a_n) \wedge b_k \,\theta\, c \wedge a_i = b_j$
ANY	$S(b_1, \ldots, b_n) \wedge b_k \,\theta\, c \rightarrow Res'(b_j)$, $R(a_1, \ldots, a_n) \wedge Res'(b_j) \wedge a_i \,\theta\, b_j$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $\exists b_1, \ldots, b_n : S(b_1, \ldots, b_n) \wedge$ $R(a_1, \ldots, a_n) \wedge b_k \,\theta\, c \wedge a_i \,\theta\, b_j$
NOT IN	$S(b_1, \ldots, b_n) \wedge b_k \,\theta\, c \rightarrow Res'(b_j)$, $R(a_1, \ldots, a_n) \wedge \neg Res'(a_i)$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow R(a_1, \ldots, a_n)$
EXISTS	$R(a_1, ..., a_n) \wedge S(b_1, ..., b_n)$ $\wedge a_i \,\theta\, b_k \rightarrow Res(a_1, ..., a_n)$	$Res(a_1, ..., a_n) \rightarrow \exists b_1, ..., b_n :$ $R(a_1, ..., a_n) \wedge S(b_1, ..., b_n) \wedge a_i \,\theta\, b_k$
Mengenoperationen:		
UNION	$R(a_1, \ldots, a_n) \rightarrow Res(a_1, \ldots, a_n)$, $S(a_1, \ldots, a_n) \rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow R(a_1, \ldots, a_n)$, $Res(a_1, \ldots, a_n) \rightarrow S(a_1, \ldots, a_n)$
EXCEPT	$R(a_1, \ldots, a_n) \wedge \neg S(a_1, \ldots, a_n)$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $R(a_1, \ldots, a_n)$
INTERSECT	$R(a_1, \ldots, a_n) \wedge S(a_1, \ldots, a_n)$ $\rightarrow Res(a_1, \ldots, a_n)$	$Res(a_1, \ldots, a_n) \rightarrow$ $R(a_1, \ldots, a_n) \wedge S(a_1, \ldots, a_n)$
Arrays:		
IN	$R(a_1, ..., a_n) \wedge X(a_i)$ $\rightarrow Res(a_1, ..., a_n)$	$Res(a_1, ..., a_n) \rightarrow$ $R(a_1, ..., a_n)$
ANY	$R(a_1, ..., a_n) \wedge X(x_i) \wedge a_i \,\theta\, x_i$ $\rightarrow Res(a_1, ..., a_n)$	$Res(a_1, ..., a_n) \rightarrow$ $R(a_1, ..., a_n)$
NOT IN	$R(a_1, ..., a_n) \wedge \neg X(a_i)$ $\rightarrow Res(a_1, ..., a_n)$	$Res(a_1, ..., a_n) \rightarrow$ $R(a_1, ..., a_n)$

Tabelle 4.5: Darstellung der Invertierung der SQL-Operationen

4.1.2 Aggregatfunktion

In diesem Abschnitt wird die Darstellung der Aggregatfunktionen als Menge von S-t-tgds und Tgds definiert. Diese Konvertierung der Funktionen stellt eine Besonderheit dar, da für die Darstellung der Funktionen Tgds und S-t-tgds benötigt werden. Ebenfalls wird die Definition von Side-Tables und die Vergabe von eindeutigen Tupel-IDs benötigt, um eine mehrfache Betrachtung der einzelnen Tupeln auszuschließen. Bei der Darstellung der Aggregatfunktionen als Menge von Abhängigkeiten werden die Methodiken und Definitionen von [Aug22a] übernommen. Dabei werden im Folgenden die Funktionen MAX, MIN, COUNT, SUM und AVG betrachtet. Ebenfalls wird die Invertierung der Funktionen beschrieben. Bei der Invertierung wird zwischen der einfachen Invertierung ohne zusätzliche Informationen und der Invertierung mit zusätzlichen Informationen (Provenance-Informationen) in Form von Side-Tables unterschieden.

Konvertierung. Die Darstellung der Aggregatfunktionen als Menge von Tgds und S-t-tgds ist in Tabelle 4.6 veranschaulicht. Diesbezüglich wird eine Aggregatfunktion auf die Relation R umgesetzt, wobei die Variable b jenes Attribut darstellt, auf welches die Funktion angewendet wird. Hierbei enthält jedes Tupel einer Relation R eine eindeutige Tupel-ID t. Für die Konvertierung der Funktionen muss eine Ordnung der Tupel realisiert werden, um eine mehrfache Verwendung der Tupel zu verhindern. Diese Definition einer Reihenfolge bzw. Ordnung der Tupel wird durch die ersten beiden Tgds realisiert, wobei eine Relation R_{ord} definiert wird. Die Relation R_{ord} definiert, nach Ausführung der ersten beiden Tgds, eine aufsteigende Reihenfolge bzw. Nummerierung der Tupel in der Variable x. Im Fall von ProSA ist die Anwendung dieser ersten beiden Tgds nicht notwendig, da in einem vorherigen Schritt von ProSA die Tupel-IDs in aufsteigende Reihenfolge nummeriert werden. Dementsprechend wird in ProSA immer eine Reihenfolge bzw. Nummerierung der Tupeln durchgeführt. Folglich werden in diesem Abschnitt die ersten beiden Tgds nur mitgeführt, um eine allgemeine Darstellung der jeweiligen Funktion als Menge von Abhängigkeiten zu betrachten.
Im nächsten Schritt wird, durch die Anwendung der nächsten beiden Tgds, die eigentliche Berechnung der Funktion durchgeführt. Dabei wird die Relation H definiert, die das Ergebnis der Aggregatfunktion beinhaltet. Diesbezüglich wird eine schrittweise Betrachtung der Tupeln umgesetzt, wobei der Wert b eines Tupels mit der Ordnung x mit dem Zwischenergebnis c des vorherigen Tupels (Ordnung $x-1$) verglichen wird. Hierbei dient die Funktion f der Berechnung des Ergebnisses, indem der Wert b und das vorherige Zwischenergebnis c miteinander verrechnet werden. Eine Ausnahme für diesen Schritt dient die Berechnung der Aggregatfunktion COUNT, da bei dieser Funktion eine zusätzliche Berechnung nicht notwendig ist. Für die Berechnung von COUNT genügt die Bestimmung der maximalen Ordnung bzw. der maximale Wert der Variable x.
Dementsprechend wird in der letzten S-t-tgd von COUNT, der Wert der Variable x als Ergebnis der Aggregatfunktion verwendet, wenn das Tupel mit der Variable x kein nachfolgendes Tupel mit der Variable/Ordnung $x+1$ besitzt. Für die weiteren Aggregatfunk-

tionen wird im letztem Schritt mit der letzten S-t-tgd das Ergebnis, dass von der Relation H beinhaltet wird, auf die Ergebnis Relation Res übertragen. Dementsprechend wird der letzte Wert von H auf die Relation Res übertragen, da dieser letzter Wert das Endergebnis der Aggregatfunktion entspricht. Eine Ausnahme bildet die Funktion für die Berechnung des Durchschnitts AVG, wobei in der letzten S-t-tgd noch eine zusätzliche Berechnung erfolgt. Diese zusätzliche Berechnung dividiert das Ergebnis von H bzw. das Ergebnis der Summierung der Werte durch die Anzahl der Tupel, um den Durchschnitt zu berechnen.

$$R(a_i, b_i, t_i) \land \neg R_{ord}(x_j, a_j, b_j, t_j) \rightarrow S'(t_i) \land R_{ord}(1, a_i, b_i, t_i)$$
$$R(a_i, b_i, t_i) \land \neg S'(t_i) \land R_{ord}(x_j, a_j, b_j, t_j) \land \neg S(t_j) \rightarrow S'(t_i) \land R_{ord}(x_j + 1, a_i, b_i, t_i) \land S(t_j)$$

$$R_{ord}(1, a, b, t) \rightarrow S''(t) \land H(b, 1)$$
$$R_{ord}(x, a, b, t) \land \neg S''(t) \land H(c, x - 1) \rightarrow S''(t) \land H(f(c, g), x) \land f(b, c) = max\{b, c\}$$

$$H(c_i, x) \land \neg H(c_j, x + 1) \rightarrow Res(c_i)$$

(a) MAX(b)/MIN(b)

$$R(a_i, b_i, t_i) \land \neg R_{ord}(x_j, a_j, b_j, t_j) \rightarrow S'(t_i) \land R_{ord}(1, a_i, b_i, t_i)$$
$$R(a_i, b_i, t_i) \land \neg S'(t_i) \land R_{ord}(x_j, a_j, b_j, t_j) \land \neg S(t_j) \rightarrow S'(t_i) \land R_{ord}(x_j + 1, a_i, b_i, t_i) \land S(t_j)$$

$$R_{ord}(x, a, b, t) \land \neg R_{ord}(x + 1, a_j, b_j, t_j) \rightarrow Res(x)$$

(b) COUNT(∗)

$$R(a_i, b_i, t_i) \land \neg R_{ord}(x_j, a_j, b_j, t_j) \rightarrow S'(t_i) \land R_{ord}(1, a_i, b_i, t_i)$$
$$R(a_i, b_i, t_i) \land \neg S'(t_i) \land R_{ord}(x_j, a_j, b_j, t_j) \land \neg S(t_j) \rightarrow S'(t_i) \land R_{ord}(x_j + 1, a_i, b_i, t_i) \land S(t_j)$$

$$R_{ord}(1, a, b, t) \rightarrow S''(t) \land H(b, 1)$$
$$R_{ord}(x, a, b, t)) \land \neg S''(t) \land H(c, x - 1) \rightarrow S''(t) \land H(f(c, g), x) \land f(b, c) = b + c$$

$$H(c_i, x) \land \neg H(c_j, x + 1) \rightarrow Res(c_i)$$

(c) SUM(b)

$$R(a_i, b_i, t_i) \land \neg R_{ord}(x_j, a_j, b_j, t_j) \rightarrow S'(t_i) \land R_{ord}(1, a_i, b_i, t_i)$$
$$R(a_i, b_i, t_i) \land \neg S'(t_i) \land R_{ord}(x_j, a_j, b_j, t_j) \land \neg S(t_j) \rightarrow S'(t_i) \land R_{ord}(x_j + 1, a_i, b_i, t_i) \land S(t_j)$$

$$R_{ord}(1, a, b, t) \rightarrow S''(t) \land H(b, 1)$$
$$R_{ord}(x, a, b, t) \land \neg S''(t) \land H(c, x - 1) \rightarrow S''(t) \land H(f(c, g), x) \land f(b, c) = b + c$$

$$H(c_i, x) \land \neg H(c_j, x + 1) \rightarrow Res(d) \land d = c_i/x$$

(d) AVG(b)

Tabelle 4.6: Aggregatfunktionen als Menge von S-t-tgds und Tgds [Aug22a]

Invertierung. Die Besonderheit bei der Invertierung der Aggregatfunktionen, ohne Verwendung von zusätzlichen Informationen, ist der hohe auftretende Informationsverlust. Dieser Informationsverlust wird durch die fehlenden Attribute der Ausgangsrelation R in der Ergebnisrelation Res erzeugt, da die Ergebnisrelation Res nur das Ergebnis der Aggregatfunktion enthält. Folglich werden bei der Invertierung (fast) alle Attribute der Ausgangsrelation R existenzquantifiziert und bei der Ausführung der Inverse durch Nullwerte ersetzt. Dementsprechend ist die Invertierung der Funktionen ohne zusätzliche Informationen durch folgende S-t-tgd definiert:

$$Res(c) \rightarrow \exists a, t : R(a, c, t).$$

Diese invertierte S-t-tgd gilt für alle Aggregatfunktionen, wobei die Variable c das Ergebnis der Aggregatfunktion beinhaltet. Entsprechend wird bei der Umsetzung dieser invertierten S-t-tgd eine Relation R erzeugt die ausschließlich ein Tupel beinhaltet, da die Relation Res nur ein Tupel mit dem Ergebnis der entsprechenden Funktion enthält. Ebenfalls werden alle Attribute in R existenzquantifiziert, mit Ausnahmen des Attributes b für die Berechnung der Aggregatfunktion. Dieses Attribut b wird durch das Ergebnis c ersetzt.
Für die Vermeidung eines hohen Informationsverlustes bei der Invertierung, können zusätzliche Informationen (Provenance-Informationen) in Form von Side-Tables (zusätzliche Relationen) verwendet werden. Beispielsweise kann durch Anwendung der Why-Provenance, die korrekte Tupelanzahl der verwendeten Tupeln für die Berechnung der Funktion rekonstruiert werden. Eine Invertierung der Aggregatfunktionen, mithilfe der Why-Provenance, ist in Tabelle 4.7 veranschaulicht. Für diese Invertierung wird eine Side-Table S'' benötigt, die die Tupel-IDs von den verwendeten Tupeln aus der Ursprungsrelation R beinhaltet. Diese Side-Table S'' wurde bei der Anwendung der entsprechenden Tgds und S-t-tgd aus Tabelle 4.6 generiert und mit den entsprechenden Tupel-IDs gefüllt.
Im ersten Schritt der Invertierung mit Why-Provenance wird mit der Ausführung der ersten Tgd, ein Zwischenergebnis R' erzeugt, welches für jede Tupel-ID in S'' ein Tupel besitzt. Hierbei wird durch die Negation einer zweiten Side-Table S''' verhindert, dass eine Tupel-ID mehrfach betrachtet wird.
Im zweiten Schritt der Invertierung wird das Zwischenergebnis von R' unverändert auf die Relation R übertragen. Dementsprechend dient die erste Tgd für die separate Berechnung des Ergebnisses der Invertierung und die S-t-tgd nur für die Übertragung des Ergebnisses auf die ursprüngliche Quellrelation R. Diese Unterscheidung von Tgd und S-t-tgd wird umgesetzt, um eine Definition von überflüssigen Relationen (z.B. Side-Tables) im Kopf der invertierten S-t-tgd zu vermeiden.

$$S''(t) \wedge Res(c) \wedge \neg S'''(t) \rightarrow \exists a : R'(a, c, t) \wedge S'''(t)$$
$$R'(a, b, t) \rightarrow R(a, b, t)$$

(a) MAX(b)/MIN(b)

$$S''(t) \wedge Res(x) \wedge \neg S'''(t) \rightarrow \exists a : R'(a, x, t) \wedge S'''(t)$$
$$R'(a, b, t) \rightarrow R(a, b, t)$$

(b) COUNT(b)

$$S''(t) \wedge Res(c) \wedge \neg S'''(t) \rightarrow \exists a : R'(a, c, t) \wedge S'''(t)$$
$$R'(a, b, t) \rightarrow R(a, b, t)$$

(c) SUM(b)

$$S''(t) \wedge Res(d) \wedge \neg S'''(t) \rightarrow \exists a : R'(a, d, t) \wedge S'''(t)$$
$$R'(a, b, t) \rightarrow R(a, b, t)$$

(d) AVG(b)

Tabelle 4.7: Invertierung der Aggregatfunktion mit Why-Provenance [Aug22a]

Beispiel. Für die Veranschaulichung der Umsetzung von Aggregatfunktionen als Menge von Abhängigkeiten wird im Folgenden ein Beispiel durchgeführt, ähnlich wie in [Aug22a]. In diesem Beispiel wird der Durchschnittswert der Noten in der Relation *Grades* berechnet. Diese Berechnung wird durch folgende SQL-Anfrage realisiert:

```
SELECT AVG(Grade)
FROM Grades
WHERE ModulNo = 002
```

Die Konvertierung der SQL-Anfrage für die Berechnung des Notendurchschnitts als Menge von Tgds und S-t-tgds ist im Folgendem dargestellt:

Tgds:
$$G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg G_{ord}(x_j, g_j, id_{g_j}) \rightarrow S'(id_{g_i}) \wedge G_{ord}(1, g_i, id_{g_i})$$
$$G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j}) \rightarrow S'(id_{g_i}) \wedge G_{ord}(x_j + 1, g_i, id_{g_i}) \wedge S(id_{g_j})$$

Tgds:
$$G_{ord}(1, g, id_g) \rightarrow S''(id_g) \wedge H(g, 1)$$
$$G_{ord}(x, g, id_g) \wedge \neg S''(id_g) \wedge H(c, x-1) \rightarrow S''(id_g) \wedge H(f(c, g), x) \wedge f(c, g) = g + c$$

S-t-tgd:
$$H(c_i, x) \wedge \neg H(c_j, x+1) \rightarrow Res(d) \wedge d = c_i/x$$

Diesbezüglich stellt das Attribut ID_{Grades} (kurz: id_g) die eindeutige Tupel-ID dar, die für die Konvertierung und Invertierung der Aggregatfunktion benötigt wird. Für die Vereinfachung der Tgds und S-t-tgds beinhaltet die Relation G_{ord} nicht alle Attribute der Relation $Grades$ (kurz: G), sondern nur die Attribute/Variablen die für die Umsetzung der Durchschnittsfunktion benötigt werden. Dementsprechend enthält die Relation G_{ord} die Tupel-ID id_g, die Note g und die Ordnung der Tupel x.
Für die Umsetzung der Berechnung des Durchschnitts, sei die Relation G beispielsweise mit folgenden Tupeln definiert:

$$G = \{(G_5, 002, 1, WS15/16, 3.7), (G_6, 002, 2, WS15/16, 1.3), (G_7, 002, 3, WS15/16, 2.3)\}.$$

Des weiterem werden die Side-Table S, S' und S'' definiert, und die Relationen für die Zwischenergebnisse G_{ord} und H initialisiert. Im ersten Schritt werden die ersten beiden Tgds ausgeführt, die eine Ordnung/Reihenfolge der Tupeln realisieren. Hierbei wird das Tupel mit der ersten Ordnung ($x = 1$) durch die erste Tgd generiert. Die weiteren Tupel erhalten ihre Ordnung durch die zweite Tgd. Für die Vermeidung einer mehrfachen Betrachtung eines Tupels, dienen die Side-Tables S und S', wobei durch die Negation der Side-Tables eine mehrfache Verwendung eines Tupels ausgeschlossen wird. Dementsprechend werden im Beispiel folgende Tgds ausgeführt:

$$G(G_5, 002, 1, WS15/16, 3.7) \wedge \neg G_{ord}(x, 3.7, G_5) \rightarrow S'(G_5) \wedge G_{ord}(1, 3.7, G_5)$$

$$G(G_6, 002, 2, WS15/16, 1.3) \wedge \neg S'(G_6) \wedge G_{ord}(1, 3.7, G_5) \wedge \neg S(G_5) \rightarrow S'(G_6) \wedge G_{ord}(2, 1.3, G_6) \wedge S(G_5)$$

$$G(G_7, 002, 3, WS15/16, 2.3) \wedge \neg S'(G_7) \wedge G_{ord}(2, 1.3, G_6) \wedge \neg S(G_6) \rightarrow S'(G_7) \wedge G_{ord}(3, 2.3, G_7) \wedge S(G_6).$$

Für die Anwendung der ersten beiden Tgds ergeben sich folgende Relationen, mit den entsprechenden Tupeln:

G:

id_g	mo	ma	se	g
G_5	002	1	$WS15/16$	3.7
G_6	002	2	$WS15/16$	1.3
G_7	002	3	$WS15/16$	2.3

G_{ord}:

x	g	id_g
1	3.7	G_5
2	1.3	G_6
3	2.3	G_7

S' :

id_g
G_5
G_6
G_7

S :

id_g
G_5
G_6

Anschließend erfolgt der zweite Schritt, in welchem die Ausführung der nächsten beiden Tgds erfolgt. In diesem Schritt wird die Summierung der Noten durchgeführt. Dabei wird diese Summierung schrittweise umgesetzt, wobei in einem Schritt ein Tupel betrachtet wird und dessen Note mit dem Zwischenergebnis des letzten Schrittes addiert wird. Diesbezüglich enthält die Relation H alle Zwischenergebnisse und das Endergebnis der Summierung. In diesem zweitem Schritt dient die Side-Table S'' für die Vermeidung der mehrfachen Verwendung eines Tupels. Für die Durchführung der Summierung werden folgende Tgds ausgeführt:

$$G_{ord}(1, 3.7, G_5) \rightarrow S''(G_5) \wedge H(3.7, 1)$$

$$G_{ord}(2, 1.3, G_6) \wedge \neg S''(G_6) \wedge H(3.7, 1) \rightarrow S''(G_6) \wedge H(5.0, 2)$$

$$G_{ord}(3, 2.3, G_7) \wedge \neg S''(G_7) \wedge H(5.0, 2) \rightarrow S''(G_7) \wedge H(7.3, 3)$$

Bei der Anwendung der zweiten und dritten Tgd werden folgende Relationen, mit den entsprechenden Tupeln generiert:

G_{ord}:

x	g	id_g
1	3.7	G_5
2	1.3	G_6
3	2.3	G_7

H:

c	x
3.7	1
5.0	2
7.3	3

S''

id_g
G_5
G_6
G_7

Schließlich wird im letzten Schritt, mit der Umsetzung der S-t-tgd, der Durchschnitt berechnet und das Ergebnis auf die Ergebnisrelation *Res* abgebildet. Folglich wird für die Berechnung des Durchschnitts das Endergebnis der Summierung durch die Anzahl der Tupel dividiert. Dementsprechend wird der letzte Wert c_i, mit der höchsten Ordnung, aus der Relation H betrachtet und dieser Wert wird durch den maximalen Wert der Ordnung x dividiert. Dabei wird folgende S-t-tgd ausgeführt:

$$H(5.0, 3) \wedge \neg H(c_j, 4) \rightarrow Res(1.25).$$

Entsprechend bildet die Relation $Res(1.25)$ das Ergebnis, für die Berechnung des Notendurchschnitts in diesem Beispiel. Für die Invertierung von Aggregatfunktionen existieren zwei Möglichkeiten, die Invertierung ohne Provenance und die Invertierung mit Provenance. Hierbei wird bei der Invertierung ohne Provenance folgende invertierte S-t-tgd generiert:

$$Res(1.25) \rightarrow \exists id_g, mo, ma, se : G(id_g, mo, ma, se, 1.25).$$

Entsprechend wird durch Anwendung dieser invertierten S-t-tgd folgende Relationen erzeugt:

Res:

d
1.25

G:

id_g	mo	ma	se	g
η_1	η_2	η_3	η_4	1.25

Folglich wird bei der Invertierung ohne Provenance eine Relation G generiert, die nur ein einzelnes Tupel enthält, wobei die eigentliche Note durch den Durchschnittswert ersetzt wird. Ebenfalls werden alle Attribute, mit Ausnahme der Note, existentzquantifiziert und durch entsprechende Nullwerte η in dem Tupel ersetzt. Entsprechend wird bei dieser Invertierung einer Aggregatfunktion ein hoher Informationverlust erzeugt. Bei der Invertierung, mithilfe von Why-Provenance, werden für das Beispiel folgende Abhängigkeiten definiert:

Tgd: $$S''(id_g) \wedge Res(d) \wedge \neg S'''(id_g) \rightarrow \exists ma, se, mo : G'(id_g, mo, ma, se, d) \wedge \neg S''(id_g)$$

S-t-tgd: $$G'(id_g, mo, ma, se, g) \rightarrow G(id_g, mo, ma, se, g).$$

Hierbei dienen die zusätzlichen Informationen (Provenance-Informationen) dazu, die Tupelanzahl der verwendeten Tupel aus der ursprünglichen Quellrelation G zu rekonstruieren. Für diese Rekonstruktion beinhaltetet die Side-Table S'' alle Tupel-IDs der Tupel, die an der Berechnung des Durchschnitts beteiligt waren. Ebenfalls wird die aktuell betrachtete Tupel-ID in der Side-Table S''' gespeichert, um eine mehrfache Betrachtung eines Tupels zu vermeiden. Entsprechend werden folgende Tgds und S-t-tgds ausgeführt:

Tgds: $$S''(G_5) \wedge Res(1.25) \wedge \neg S'''(G_5) \rightarrow \exists ma, se, mo : G'(G_5, \eta_1, \eta_2, \eta_3, 1.25) \wedge \neg S''(G_5)$$

$$S''(G_6) \wedge Res(1.25) \wedge \neg S'''(G_6) \rightarrow \exists ma, se, mo : G'(G_6, \eta_4, \eta_5, \eta_6, 1.25) \wedge \neg S''(G_6)$$

$$S''(G_7) \wedge Res(1.25) \wedge \neg S'''(G_7) \rightarrow \exists ma, se, mo : G'(G_7, \eta_7, \eta_8, \eta_9, 1.25) \wedge \neg S''(G_7)$$

S-t-tgds: $$G'(G_5, \eta_1, \eta_2, \eta_3, 1.25) \rightarrow G(G_5, \eta_1, \eta_2, \eta_3, 1.25)$$

$$G'(G_6, \eta_4, \eta_5, \eta_6, 1.25) \rightarrow G(G_6, \eta_4, \eta_5, \eta_6, 1.25)$$

$$G'(G_7, \eta_4, \eta_5, \eta_6, 1.25) \rightarrow G(G_7, \eta_7, \eta_8, \eta_9, 1.25).$$

Diesbezüglich werden folgende Relationen generiert:

Res:

d
1.25

G:

id_g	mo	ma	se	g
G_5	η_1	η_2	η_3	1.25
G_6	η_4	η_5	η_6	1.25
G_7	η_7	η_8	η_9	1.25

S'':

id_g
G_5
G_6
G_7

S'''

id_g
G_5
G_6
G_7

Die Relation G entspricht dem Ergebnis der Invertierung bzw. der minimalen Teildatenbank für die Berechnung des Durchschnittes. Dabei sind die beiden Relationen G und G' identisch, da in der S-t-tgd das Zwischenergebnis G' ohne Veränderungen auf G abgebildet wird. Diese Methodik der Invertierung mit Why-Provenance stellt die richtige Tupelanzahl in der minimalen Datenbank her. Allerdings ist ein hoher Informationsverlust vorhanden, da die meisten Attributwerte durch Nullwerte ersetzt werden. Aufgrund dessen, dass die Ergebnisrelation *Res* nur die Lösung für die Aggregatfunktion enthält, kann der Verlust der Attributwerte nur durch Verwendung einer weiteren Side-Table ausgeglichen werden. Infolgedessen müsste die Side-Table die entsprechende Attributwerte enthalten, um den Informationsverlust auszugleichen. An dieser Stelle sei anzuführen, dass trotz des hohen Informationsverlust bei der Berechnung der minimalen Teildatenbank die Rekonstruktion des Anfrageergebnisses (Ziel von ProSA) möglich ist.

4.1.3 ALL-Operation

Für die Umsetzung einer verschachtelten Anfrage, mithilfe der Operation ALL, als Menge von Abhängigkeiten, ist eine Fallunterscheidung zwischen den einzelnen Vergleichsoperatoren $(<, >, \neq, =, \geq, \leq)$ notwendig. Diese Fallunterscheidung ist in Tabelle 4.8 veranschaulicht. Demnach ist die Bestimmung des maximalen bzw. minimalen Wert notwendig, um die Vergleichsoperationen $(<, >, \geq, \leq)$ bei der Operation ALL anzuwenden. Diese Bestimmung des maximalen bzw. minimalen Wertes wird mit der Anwendung der Tgds und S-t-tgds, die für die Umsetzung der Aggregatfunktionen MAX und MIN (Abschnitt 4.1.2) benötigt werden, realisiert. Hierbei wird in Tabelle 4.9 die Konvertierung der ALL-Operation, mit den unterschiedlichen Vergleichsoperationen, dargestellt. Für die Konvertierung des einzelnen Fälle wurde folgende SQL-Anfrage als Vorlage verwendet:

SELECT $*$ FROM R
WHERE $a_i\,\theta$ ALL
SELECT b_j FROM S
WHERE $b_k\,\theta\,c$

Dementsprechend muss für die Umsetzung der ALL-Operation zuerst die Unteranfrage ausgewertet werden, indem eine Tgd definiert wird. Diese Tgd bildet die Relation S auf ein Zwischenergebnis Res', welches das Ergebnis der Unteranfragen repräsentiert, ab. Anschließend erfolgt für die Vergleichsoperationen $(<, >, \geq, \leq)$ die Anwendung der entsprechenden Aggregatfunktion, dabei wird die Funktion auf das Zwischenergebnis Res' angewendet. Hierbei seien die Relationen MAX und MIN das Ergebnis der Anwendung der entsprechenden Aggregatfunktion. Schließlich erfolgt die S-t-tgd, die den Vergleich zwischen den Attribut a_i und dem Ergebnis der Aggregatfunktion durchführt und das Endergebnis *Res* bestimmt. Dagegen wird bei den Vergleichsoperationen $(=, \neq)$ keine Aggregatfunktion angewendet, sondern diese Vergleiche werden mit der Operation ANY umgesetzt. Hierbei wird die ANY-Operation mit der gegenteiligen Vergleichsoperation angewendet, die Relation ANY sei die Ergebnisrelation dieser Anwendung. Anschließend erfolgt die eigentliche S-t-tgd, die mithilfe der Relation ANY, die Ergebnisrelation

Res bildet. Entsprechend wird ein Attributwert a_i bzw. das dazugehörige Tupel nur in die Ergebnisrelation *Res* aufgenommen, wenn es nicht in der Relation *ANY* vorhanden ist.
Für die Invertierung der Operation ALL wird eine Abbildung von *Res* auf die beiden Ausgangsrelationen R und S realisiert. Die folgende S-t-tgd sei die invertierte S-t-tgd der ALL-Operation, für alle Vergleichsoperationen:

$$Res(a_1, ..., a_n) \rightarrow \exists b_1, ..., b_n : R(a_1, ..., a_n) \wedge S(b_1, ..., b_n).$$

Vergleichsoperation	**Fallunterscheidung**
$a_i >$ALL(...)	Der Wert vom Attribute a_i muss größer, als der maximale Wert in der Menge des ALL-Operators, sein.
$a_i \geq$ ALL(...)	Der Wert vom Attribut a_i muss größer oder gleich, als der maximale Wert in der Menge des ALL-Operators, sein.
$a_i <$ALL(...)	Der Wert vom Attribut a_i muss kleiner, als der minimale Wert in der Menge des ALL-Operators, sein.
$a_i \leq$ ALL(...)	Der Wert vom Attribut a_i muss kleiner oder gleich, als der minimale Wert in der Menge des ALL-Operators sein.
$a_i =$ ALL(...)	Der Wert vom Attribut a_i muss gleich, mit jedem Wert in der Menge des ALL-Operators sein.
$a_i \neq$ ALL(...)	Der Wert vom Attribut a_i muss ungleich, mit jedem Wert in der Menge des ALL-Operators sein.

Tabelle 4.8: Fallunterscheidung ALL-Operator, ähnlich wie in [Tut21]

Vergleichsoperation	**Darstellung als S-t-tgd und Tgd**
$a_i >$ALL(...)	$S(b_1, ..., b_m) \wedge b_j \theta c \rightarrow Res'(b_j)$ $R(a_1, ..., a_n) \wedge MAX(c) \wedge a_i > c \rightarrow Res(a_1, ..., a_n)$
$a_i \geq$ ALL(...)	$S(b_1, ..., b_m) \wedge b_j \theta c \rightarrow Res'(b_j)$ $R(a_1, ..., a_n) \wedge MAX(c) \wedge a_i \geq c \rightarrow Res(a_1, ..., a_n)$
$a_i <$ALL(...)	$S(b_1, ..., b_m) \wedge b_j \theta c \rightarrow Res'(b_j)$ $R(a_1, ..., a_n) \wedge MIN(c) \wedge a_i < c \rightarrow Res(a_1, ..., a_n)$
$a_i \leq$ ALL(...)	$S(b_1, ..., b_m) \wedge b_j \theta c \rightarrow Res'(b_j)$ $R(a_1, ..., a_n) \wedge MIN(c) \wedge a_i \leq c \rightarrow Res(a_1, ..., a_n)$
$a_i =$ ALL(...)	$R(a_1, ..., a_n) \wedge \neg ANY_{\neq}(a_i) \rightarrow Res(a_1, ..., a_n)$
$a_i \neq$ ALL(...)	$R(a_1, ..., a_n) \wedge \neg ANY_{=}(a_i) \rightarrow Res(a_1, ..., a_n)$

Tabelle 4.9: Konvertierung ALL-Operator als S-t-tgd

4.1.4 Vollständigkeit

Um einen Nachweis auf Vollständigkeit der oben aufgeführten Operationen bzw. SQL-Anfragen durchzuführen, wird die Dokumentation des relationalen Datenbanksystems *PostgreSQL* [Gro21] betrachtet. In dieser Dokumentation werden die SQL-Anfragen im zweiten Kapitel aufgeführt, dessen Ausführung in PostgreSQL (Version 14.1) möglich sind. Dementsprechend werden im zweitem Kapitel alle Anfragen aufgelistet, die für die Umsetzung eines aktuellen Datenbanksystems notwendig sind. Folglich wird im Rahmen dieser Arbeit die Betrachtung dieser Anfragen als vollständig angenommen. Für die meisten Anfragen wurde dessen Darstellung als Menge von Abhängigkeiten und die Invertierung dieser Darstellung bereits, in den Abschnitten 4.1.1, 4.1.2 und 4.1.3, definiert. Im Folgenden werden die restlichen Anfragen, dessen Darstellung als Menge von Abhängigkeiten nicht möglich ist oder die einen Sonderfall darstellen, aufgeführt:

Schemaevolutionoperationen. Im Rahmen dieser Arbeit werden ausschließlich die typischen Anfrage-Operationen betrachtet, die ein Anfrage auf ein bestehendes Datenbankschema bzw. Datenbankinstanz ausführen, um ein Anfrageergebnis zu erhalten. Dabei findet keine Betrachtung der Schemaevolutionsoperationen statt, die eine Veränderung des aktuellen Datenbankschema durchführen. Exemplarisch für die Schemaevolutionsoperationen sind die Operationen die neue Tabellen oder Spalten zum aktuellen Schema hinzufügen oder diese entfernen. Dementsprechend wird die Umsetzung dieser Evolutionsoperationen als Menge von Abhängigkeiten nicht weiter beschrieben. Allerdings ist eine genaue Betrachtung der Konvertierung und Invertierung dieser Operationen in [CMZ08] aufgeführt.

GROUP BY. Die Operation GROUP BY realisiert eine Zusammenfassung von gleichen Attributwerten, innerhalb eines Attributs bzw. einer Spalte. Definitionsgemäß basieren die Abhängigkeiten (S-t-tgds, Tgds und Egds) auf Mengen, deshalb ist eine Unterscheidung und Zusammenfassung von Attributwerten nicht möglich. Entsprechend ist eine Umsetzung der Operation GROUP BY nicht oder nur unter Verwendung von vielen Side-Tables möglich. Beispielsweise besteht die Möglichkeit die Operation umzusetzen, wenn für jeden unterschiedlichen Attributwert eine Side-Table gegeben ist. Dabei würde die Zusammenfassung der gleichen Attributwerte, durch die gegebenen Side-Tables realisiert werden. Anschließend müsste die eigentliche Anfrage auf jede Side-Table angewendet werden, um für jeden unterschiedlichen Attributwert ein Zwischenergebnis zu generieren. Schließlich müssten diese Zwischenergebnisse in einer Ergebnisrelation zusammengefasst werden, allerdings unter Betrachtung einer Ordnung, um die Zusammenfassung der gleichen Attributwerte zu erhalten. Ein Beispiel dieser Anwendung der Side-Tables wird durch folgendes Beispiel veranschaulicht. Hierbei wird folgenden SQL-Anfrage betrachtet:

```
SELECT MatricNo, AVG(Grade)
FROM Grades
GROUP BY MatricNo
```

Diese SQL-Anfrage berechnet den Notendurchschnitt für jeden einzelnen Studierenden. Entsprechend wird eine Zusammenfassung und Unterscheidung der Matrikelnummern bzw. der Attributwerte im Attribut *MatricNo* durchgeführt. Für die Umsetzung dieser SQL-Anfragen wird für jeden Studierenden eine Side-Table $S(ma, g)$ benötigt, die die Matrikelnummer (kurz: ma) und die Noten (kurz: g) beinhaltet. Dementsprechend wird die Gruppierung der Attributwerte durch die Side-Tables realisiert bzw. die Side-Tables beinhalten das Ergebnis der Gruppierung. Anschließend würde in diesem Beispiel die Berechnung des Durchschnittes (Abschnitt 4.1.2) erfolgen, wobei für jeden Studierenden bzw. für jede Side-Table $S(ma, g)$ die Berechnung des Durchschnitts einzeln durchgeführt wird.

HAVING. Die HAVING-Klausel realisiert die Umsetzung einer Selektionsbedingung. Dementsprechend ist die Umsetzung der HAVING-Klausel als Menge von Abhängigkeiten mit der Umsetzung der WHERE-Klausel gleichzusetzen. Der einzige Unterschied zwischen diesen beiden Klausel ist, dass die HAVING Operation nur in Zusammenhang mit der Operation GROUP BY verwendet wird. Aufgrund dessen, dass die Operation GROUP BY nicht als Menge von Abhängigkeiten darstellbar ist, wird keine weitere Betrachtung der Operation HAVING durchgeführt. Allerdings kann die Operation HAVING mit dem gleichen Prinzip, wie bei der Selektion (Tabelle 4.2), umgesetzt werden.

Window-Funktionen. Bei den Window-Funktionen werden Funktionen, über gleiche Attributwerte in einer oder mehren Spalten, angewendet. Exemplarisch für eine Window-Funktion ist die Berechnung des Notendurchschnitts für jeden Studierenden, diese Berechnung wird durch folgende SQL-Anfrage realisiert:

```
SELECT MatricNo, AVG(Grade)
OVER (PARTITION BY MatricNo)
FROM Grades
```

Folglich wird durch Anwendung dieser Anfrage der individuelle Notendurchschnitt für jeden Studenten berechnet. Hierbei ist eine Unterscheidung und Zusammenfassung der einzelnen Attributwerte notwendig. Dementsprechend ist die Umsetzung der Window-Funktionen mit der Umsetzung der Operation GROUP BY gleichzusetzen, wobei beide Operationen/Funktionen nicht als Menge von Abhängigkeiten umsetzbar sind bzw. nur mithilfe der Definition von vielen Side-Tables realisierbar sind.

VALUES. In PostgresSQL besteht die Möglichkeit mithilfe der Operation VALUES eine Liste zu erstellen, die eine Tabelle definiert. Ein Beispiel einer Anwendung dieser Operation, ist im Folgendem dargestellt:

```
SELECT * FROM (VALUES (1, 2), (1, 3) (2, 4)).
```

Entsprechend wird eine Liste definiert, wobei die Elemente in der Liste als Tupel einer Tabelle interpretiert werden. Eine Umsetzung dieser Definition einer Liste als Menge von Abhängigkeiten ist möglich, indem die Liste als Relation definiert wird. Dabei wird das gleiche Prinzip angewendet, wie bei der Anwendung einer Operation auf ein Array (Tabelle 4.2). Dementsprechend würde die SQL-Anfrage folgendermaßen als S-t-tgd umgesetzt werden:

$$X(x_1, x_2) \rightarrow Res(x_1, x_2).$$

Folglich wird die Liste der Operation VALUES als neue Relation X in der S-t-tgd umgesetzt, wobei die Relation X die Werte/Elemente der Liste als Tupel besitzt.

Unteranfragen. Für die Umsetzung der verschachtelten Anfragen werden S-t-tgds und Tgds verwendet. Hierbei dienen die Tgds für die Umsetzung der Unteranfragen, wobei jede Tgd eine Unteranfrage repräsentiert. Bei der Konvertierung der verschachtelten Anfragen wird mit der innersten Unteranfrage begonnen, die entsprechend als Tgd umgesetzt wird. Diese Tgd liefert ein Zwischenergebnis, welches im Rumpf der nächsten Unteranfrage verwendet wird. Dementsprechend werden die Unteranfragen von Innen nach Außen konvertiert, wobei das Zwischenergebnis der vorherigen Unteranfrage/Tgd für die Bildung des Zwischenergebnis der aktuell betrachteten Unteranfrage verwendet wird. Anschließend wird die äußerste (Teil-)Anfrage als S-t-tgd dargestellt, die das eigentliche Ergebnis der verschachtelten Anfrage generiert. Ebenfalls wird in der S-t-tgd das Zwischenergebnis der letzten Tgd verwendet, um das eigentliche Ergebnis zu generieren. Dementsprechend wird hierbei das gleiche Prinzip, wie bei den verschachtelten Anfragen in Tabelle 4.3 angewendet. Mit diesem Prinzip können auch allgemeine Unteranfragen, die beispielsweise in der FROM-Klausel definiert sind, als Menge von Abhängigkeiten dargestellt werden. Analog dazu erfolgt die Umsetzung der Operation WITH, wobei komplexe Anfragen in einfachere (Teil-)Anfragen zerlegt werden. Dabei werden die Unteranfragen, die durch Anwendung der Operation WITH definiert werden, als einzelne Tgds umgesetzt. Ein Beispiel für die Umsetzung einer mehrfach verschachtelten Anfrage wird durch folgende SQL-Anfrage veranschaulicht:

```
SELECT * FROM
  (SELECT MatricNo, LastName, FirstName, Grade
   FROM (
       SELECT *
       FROM Students NATURAL JOIN Grades))
WHERE Grade = 1.0
```

In dieser SQL-Anfrage wird ein (natürlicher) Verbund von den Relationen *Students* und *Grades* durchgeführt. Hierbei werden zwei Unteranfragen angewendet, wobei die innerste Unteranfrage den Verbund und die andere Unterfrage eine Projektion realisiert. Die eigentliche (äußerste) Anfrage führt eine Selektion des Attributs *Grade* durch. Diese verschachtelte Anfrage dient ausschließlich als Beispiel, da diese verschachtelte Anfrage

auch als einfache Anfrage ohne Unteranfragen realisierbar wäre. Bei der Umsetzung dieser Anfrage wird folgende Menge von Abhängigkeiten gebildet:

$$\text{Tgds:} \quad St(ma, ln, fn, sc) \wedge G(ma, mo, se, g) \rightarrow Res'(ma, ln, fn, sc, mo, se, g)$$
$$Res'(ma, ln, fn, sc, mo, se, g) \rightarrow Res''(ma, ln, fn, g)$$

$$\text{S-t-tgds:} \quad Res''(ma, ln, fn, g) \wedge g = 1.0 \rightarrow Res(ma, ln, fn, g).$$

Bei der Umsetzung der verschachtelten Anfrage werden die Unteranfragen als Tgds umgesetzt. Hierbei wird mit der innersten Anfrage begonnen. Dementsprechend wird im Beispiel die innerste Unteranfrage, die den Verbund der Relationen *Students* und *Grades* durchführt, durch die erste Tgd umgesetzt. Hierbei werden die Relationen *Students* und *Grades* auf ein Zwischenergebnis Res' abgebildet. Anschließend wird dieses Zwischenergebnis Res' in der zweiten Tgd verwendet, um die nächste Unteranfrage, die die Projektion auf die Attribute $MatricNo$, $LastName$, $FirstName$ und $Grade$ durchführt, umzusetzen. Die zweite Tgd liefert ebenfalls ein Zwischenergebnis Res'', welches in der nächsten Abhängigkeit verwendet wird. Die nächste Abhängigkeit ist die S-t-tgd, die die äußerste Anfrage repräsentiert. Entsprechend wird in der S-t-tgd die Selektion auf das Zwischenergebnis Res'' angewendet, um das (Gesamt-)Ergebnis Res der verschachtelten Anfrage zu generieren.
Bei der Invertierung der verschachtelten Anfrage wird eine Rückabbildung vom Anfrageergebnis auf die Quellrelationen durchgeführt. Infolgedessen wird bei der Invertierung des Beispiels folgende invertierte S-t-tgd erzeugt:

$$Res(ma, ln, fn, g) \rightarrow \exists mo, se, sc : St(ma, ln, fn, sc) \wedge G(ma, mo, se, g).$$

Entsprechend wird das (Gesamt-)Ergebnis Res der verschachtelten Anfrage auf die beiden Quellrelationen *Students* und *Grades* abgebildet. Hierbei werden die Tgds verwendet, um eine Rückverfolgung auf die eigentlichen Quellrelationen durchzuführen. Die Zwischenergebnisse der Tgds werden nicht in das Ergebnis der Invertierung aufgenommen, da diese im Quellschema bzw. in der Datenbankinstanz nicht vorhanden sind. Folglich haben die Zwischenergebisse keine Relevanz bei der Berechnung einer minimalen Teildatenbank. Außerdem würde bei einer erneuten Anwendung der Anfrage auf die minimale Teildatenbank die Zwischenergebnisse wieder automatisch berechnet werden.

In diesem Abschnitt wurde die Konvertierung und Invertierung der SQL-Anfragen betrachtet. Hierbei wurden unterschiedliche SQL-Anfrage und dessen Konvertierung und Invertierung als Menge von Abhängigkeiten aufgeführt. Um die aufgeführten invertierten Abhängigkeiten der SQL-Anfragen, beispielsweise wie in den Tabellen 4.4 und 4.5, automatisch zu generieren, wird im folgenden Abschnitt eine eigene Invertierungsmethode entwickelt. Ebenfalls wird im folgenden Abschnitt eine Methode für die Invertierung der Aggregatfunktionen (Abschnitt 4.1.2) mit Why-Provenance entwickelt.

4.2 Algorithmen

In diesem Abschnitt werden Algorithmen für die Invertierung von Abhängigkeiten betrachtet. Hierbei wird zuerst im Abschnitt 4.2.1 ein einfacher Algorithmus für die Invertierung von einer S-t-tgd skizziert. Anschließend wird ein Vergleich von zwei Invertierungsmethoden (Abschnitt 4.2.2) durchgeführt. Diese Invertierungsmethoden wurden in den Abschnitten 3.3.3 und 3.4 vorgestellt. Darauffolgend wird ein eigener Algorithmus für die Invertierung von Abhängigkeiten entwickelt (Abschnitt 4.2.3). Hierbei wird der DirectAdaptedMaxExtendedRecovery-Algorithmus (Abschnitt 3.4) als Vorlage für den eigenen Invertierungsalgorithmus verwendet und entsprechend erweitert.

4.2.1 Allgemeiner Algorithmus

Ein einfacher (abstrakter) Algorithmus, für die Invertierung einer einzelnen S-t-tgd, wird folglich beschrieben. Als Eingabe für den Algorithmus dient eine S-t-tgd Q, beispielsweise der Form:

$$\forall x, y : (\phi(x, y) \rightarrow \psi(x))$$

Definitionsgemäß entspricht ϕ einer Konjunktion, über einem Quellschema bzw. einer Datenbankinstanz und ψ einer Konjunktion, über einem Zielschema. Als Ausgabe liefert der Algorithmus eine invertierte S-t-tgd, beispielsweise:

$$\forall x : (\psi(x) \rightarrow \exists y : \phi(x, y))$$

Dementsprechend werden im ersten Schritt der Invertierung der Rumpf ϕ und der Kopf ψ (ohne Quantoren) der Eingabe Q miteinander getauscht. Anschließend erfolgt die Anpassung der Quantoren, dabei werden alle Variablen im Rumpf der invertierten S-t-tgd allquantifiziert. Des weiterem werden alle Variablen die im Kopf, aber nicht im Rumpf der invertierten S-t-tgd vorhanden sind, existenzquantifiziert.
Ebenfalls werden für Gleichheitsprädikate der Form $x = c$, wobei x eine Variable über einem Relationssymbol R repräsentiert und c eine Konstante entspricht, eine Anpassung angenommen. Hierbei wird die Variable x in allen Relationen des Kopfes der invertierten S-t-tgd durch die Konstante c ersetzt. Die Variablen die durch eine Gleichung der Form $x = c$ ersetzt werden, werden nicht existenzquantifiziert. Diesbezüglich findet keine Ersetzung bei einer Gleichung der Form $x\theta c$ statt, wobei $\theta \in \{<, >, \leq, \geq, \neq\}$ gilt, da kein genauer Wert für die Variable x bestimmt werden kann. Folglich wird eine Gleichung $x\theta c$ in den Kopf der invertierten S-t-tgd mitübernommen. Schließlich bildet die invertierte S-t-tgd die Ausgabe für den Algorithmus.

4.2.2 Vergleich Algorithmen

In diesem Abschnitt wird ein Vergleich zwischen den Algorithmen, die im Kapitel: *Stand der Technick* (Kapitel 3) vorgestellt worden sind, durchgeführt. Folglich werden

die Gemeinsamkeiten und Unterschiede der Invertierungsmethode des KSWS-Projekts [WWO+21] (Abschnitt 3.3.3) und des DirectAdaptedMaxExtendedRecovery-Algorithmus [Zim21] (Abschnitt 3.4) betrachtet. Für die Veranschaulichung der Gemeinsamkeiten und Unterschiede der beiden Algorithmen werden in Tabelle 4.10 unterschiedliche Aspekte kurz zusammengefasst und gegenübergestellt. Im Folgenden werden die einzelnen Aspekte betrachtet, dabei werden die Aspekte Charakteristik, Eingabe und Ausgabe in einem Absatz zusammengefasst.

Charakteristik. Ein wesentlicher Unterschied der beiden Algorithmen ist die unterschiedliche Herangehensweise bei der Invertierung einer S-t-tgd bzw. einer SQL-Anfrage. Dabei wird beim KSWS-Projekt ein hybrider Algorithmus gewählt, wobei eine SQL-Anfrage in eine S-t-tgd konvertiert wird und anschließend, mithilfe der Konvertierung, eine Invertierung der S-t-tgd erfolgt. Als hybride Algorithmen [WWO+21] werden in diesem Kontext Algorithmen genannt, die eine SQL-Anfrage als Eingabe verwenden und eine invertierte Abhängigkeit (S-t-tgd) als Ausgabe liefern. Neben dieser Methodik der Invertierung, existieren noch Algorithmen die eine S-t-tgd als Eingabe verwenden und als Ausgabe eine invertierte S-t-tgd zurückgeben. Ebenfalls wäre eine Möglichkeit der Invertierung die SQL-Anfrage direkt zu invertieren und darauffolgend in eine S-t-tgd zu konvertieren. Eine Kombination dieser zwei Methodiken bildet einen hybrider Algorithmus. Die Invertierungsmethode des KSWS-Projekt verwendet für die Invertierung einer SQL-Anfrage die Funktionalitäten des SQL-Parser [KRSZ21] (Abschnitt 3.3.2). Dementsprechend muss für die Umsetzung dieser Methode eine Integrierung in den SQL-Parser erfolgen.
Im Gegensatz zum KSWS-Projekt, verwendet der DirectAdaptedMaxExtendedRecovery-Algorithmus eine Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ als Eingabe, wobei eine Menge von Abhängigkeiten Σ in Form von S-t-tgds gegeben ist. Als Ausgabe liefert der Algorithmus eine Schemaabbildung $\mathcal{M} = (T, S, \Omega)$, wobei Ω eine Menge von Abhängigkeiten mit invertierten S-t-tgds, in Form von Disjunktiven-Mengen repräsentiert. Folglich wird durch den DirectAdaptedMaxExtendedRecovery-Algorithmus ein allgemeiner Algorithmus, für die Invertierung von S-t-tgds, realisiert. Für die Invertierung einer SQL-Anfrage in ProSA mit diesem Algorithmus, muss die entsprechende SQL-Anfrage zunächst in eine oder mehrere S-t-tgds konvertiert werden, bevor sie durch den DirectAdaptedMaxExtended-Recovery-Algorithmus verarbeitet werden kann.

Funktionsweise. Ein weiterer Unterschied der beiden Algorithmen besteht in der Funktionsweise, die aus den unterschiedlichen Herangehensweisen resultiert. Bei der Invertierungsmethode des KSWS-Projekts wird eine Verschachtelung der SQL-Anfrage mithilfe des SQL-Parsers umgesetzt. Hierbei muss eine Konstrukt realisiert werden, beispielsweise in Form eines SFW-Baums. Diese Konstrukt muss die Struktur der einzelnen Anfragen, die durch die Verschachtelung geniert worden sind, beinhalten. Außerdem dient das Konstrukt für die Speicherung der Reihenfolge der einzelnen Anfragen, um die Invertierung

der einzelnen Anfragen in der richtigen Reihenfolge zu gewährleisten. Anschließend werden die einzelnen Anfragen mithilfe des SQL-Parsers in S-t-tgds konvertiert. Danach erfolgt die Invertierung der einzelnen S-t-tgd, mithilfe von vordefinierten Regeln. Bei der Invertierung ist die Einhaltung der richtigen Reihenfolge entscheidend. Die einzelnen Anfragen bzw. S-t-tgds müssen von Außen nach Innen anhand der Verschachtelung invertiert werden, um eine Zusammenfassung der einzelnen invertierten (Teil-)S-t-tgds in eine invertierte (Gesamt-)S-t-tgd zu gewährleisten. Dabei dient bei der Invertierung der einzelnen S-t-tgds der Kopf der vorherigen S-t-tgd als Rumpf der aktuellen S-t-tgd. Im Gegensatz dazu ist beim DirectAdaptedMaxExtendedRecovery-Algorithmus keine Aufspaltung/Verschachtelung der Anfrage notwendig, um eine Invertierung der S-t-tgds bzw. der Anfrage durchzuführen. Ebenfalls erfolgt die Invertierung ohne die Betrachtung einer bestimmten Reihenfolge. Als erster Schritt des Algorithmus dient die Normalisierung der Köpfe der S-t-tgds. Darauffolgend werden die S-t-tgds invertiert, indem eine Umkehrung der Pfeilrichtungen erfolgt. Schließlich werden die invertierten S-t-tgds in Disjunktive-Mengen zusammengefasst und ausgegeben.

Vorteile. Ein wichtiger Vorteil bei der Betrachtung der Invertierungsmethode des KSWS-Projekts ist die Integration in den SQL-Parsers, da Funktionalitäten des Parser übernommen werden können, beispielsweise die Verschachtelung der SQL-Anfrage. Dementsprechend wird im KSWS-Projekt eine Invertierungsmethode skizziert, die eine Konvertierung und Invertierung der SQL-Anfrage umsetzt. Dagegen muss beim DirectAdaptedMaxExtendedRecovery-Algorithmus eine vorherige Konvertierung der SQL-Anfrage, in eine oder mehrere S-t-tgds, erfolgen. Allerdings realisiert der DirectAdaptedMaxExtendedRecovery-Algorithmus einen allgemeinen Algorithmus für die Invertierung von S-t-tgds, folglich muss keine Betrachtung und Unterscheidung von einzelnen Anfragearten (z.B. Projektion, Selektion und Verbund) durchgeführt werden. Ein weiterer Vorteil des DirectAdaptedMaxExtendedRecovery-Algorithmus ist, dass der Algorithmus für die Verwendung mit ChaTEAU konstruiert und angepasst worden ist. Infolgedessen besteht die Möglichkeit die resultierenden invertierten S-t-tgd als Parameters des CHASE-Algorithmus, der mithilfe von ChaTEAU realisiert wird, zu verwenden. Diese Anwendung der invertierten S-t-tgds mit dem CHASE-Algorithmus ist für ProSA entscheidend, um eine Berechnung einer minimale Teildatenbank durchzuführen.

Nachteile. Ein Nachteil der Invertierungsmethode des KSWS-Projekts ist die notwendige Einhaltung der Reihenfolge, bei der Invertierung und Zusammenfassung der einzelnen Anfragen/S-t-tgds. Dementsprechend ist keine beliebige Invertierung der S-t-tgds möglich und ein Konstrukt für die Speicherung der Reihenfolge muss realisiert werden. Außerdem muss bei der Methode für jede Anfrageart, beispielsweise Selektion, Projektion und Verbund, eine Regel für die Invertierung aufgestellt werden. Des weiteren muss bei jeder einzelnen Anfrage eine Unterscheidung der unterschiedlichen Anfragearten durchgeführt werden, um die richtige Regel für die Invertierung zu bestimmen. Im Gegensatz

dazu ist ein Nachteil des DirectAdaptedMaxExtendedRecovery-Algorithmus, dass die Ausgabe der invertierten S-t-tgds als Disjunktive-Mengen erfolgt und nicht als einzelne zusammenhängende S-t-tgd. Allerdings können die Disjunktiven-Mengen mit dem Dijunktiven-CHASE [Zim21], der eine Anpassung des CHASE-Algorithmus entspricht und in ChaTEAU realisiert worden ist, verarbeitet werden.

	KSWS-Projekt	**DirectAdaptedMaxExtendedRecovery**
Charakteristik	- Hybrider Algorithmus für Invertierung - Integrierung in den SQL-Parser von ProSA	- Allgemeiner Algorithmus für Invertierung - Abgeleitet vom MaxExtendedRecovery
Eingabe	- SQL-Anfrage	- Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ mit S-t-tgds in Σ
Ausgabe	- invertierte Anfrage in Form von einer S-t-tgd	- invertierte Schemaabbildung $\mathcal{M} = (T, S, \Omega)$ mit S-t-tgds in Ω von Disjunktiven-Mengen
Funktionsweise	- Aufspaltung von SQL-Anfrage in einfachere Anfragen - Invertierung der einfacheren Anfragen durch Anwendung von Regeln - Zusammenfügen der einzelnen invertierten Anfragen	- Normalisierung der S-t-tgds - Invertierung der normalisierten S-t-tgds mit Pfeilumkehrung - Zusammenfassung der invertierten S-t-tgds in Disjunktive-Mengen
Vorteile	- Integration von Algorithmus in den SQL-Parser	- Verwendung mit ChaTEAU - allgemeiner Algorithmus ohne Verwendung von Regeln
Nachteile	- Definition von Regeln für alle einfacheren Anfragen - Betrachtung der Reihenfolge	- Disjunktive-Mengen als Ausgabe (Disjunktiver-CHASE)

Tabelle 4.10: Vergleich KSWS-Algorithmus mit DirectAdaptedMaxExtendedRecovery-Algorithmus

Im Rahmen dieser Arbeit wird für eine weitere Betrachtung der Umsetzung einer Invertierungsmethode für ProSA der DirectAdaptedMaxExtendedRecovery-Algorithmus verwendet. Dieser Algorithmus wird als Vorlage für die Entwicklung einer eigenen Invertierungsmethode verwendet, da in diesem Algorithmus keine Aufteilung/Verschachtelung der Anfrage in einfachere (Teil-)Anfragen notwendig ist. Ebenfalls ist in diesem Algortihmus keine Betrachtung der Reihenfolge der Abhängigkeiten/Anfragen notwendig, d.h.

die Abhängigkeiten können in beliebiger Reihenfolge invertiert werden.
Im folgendem Abschnitt wird eine Erweiterung und Anpassung des DirectAdaptedMaxExtendedRecovery-Algorithmus vorgenommen, um einen eigenen Invertierungsalgorithmus zu entwickeln. Hierbei wird die Eingabe des DirectAdaptedMaxExtendedRecovery-Algorithmus mit Tgds erweitert. Die Tgds werden benötigt, um beispielsweise verschachtelte Anfragen oder Aggregatfunktionen umzusetzen. Eine weitere Anpassung ist die Vermeidung von Disjunktiven Mengen als Ausgabe. Die Disjunktiven Mengen stellen die invertierten Abhängigkeiten bzw. S-t-tgd als mehrere (Teil-)Abhängigkeiten in unterschiedlichen Mengen dar. Im eigenen Invertierungsalgorithmus werden ausschließlich die invertierten (Gesamt-)S-t-tgd als Ausgabe zurückgeliefert. Entsprechend werden die invertierten S-t-tgd nicht in Disjunktive-Mengen aufgeteilt, sondern möglichst weit zusammengefasst und anschließend im Gesamten ausgeben.

4.2.3 Algorithmus Invertierung

In diesem Abschnitt wird ein Algorithmus für die Invertierung einer Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ veranschaulicht, ohne die Betrachtung von Provenance-Informationen. Der Algorithmus erweitert den DirectAdaptedMaxExtendedRecovery-Algorithmus (Abschnitt 3.4) und ist im Algorithmus 5 skizziert. Als Eingabe erhält der Algorithmus eine Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$. Im Gegensatz zum DirectAdaptedMaxExtendedRecovery-Algorithmus wird im Algorithmus 5 eine Menge von S-t-tgds und Tgds als Eingabe betrachtet. Diese Erweiterung um die Tgds in der Eingabe wurde durchgeführt, da für die Umsetzung einiger SQL-Anfragen Tgds benötigt werden. Beispielsweise wird bei der Konvertierung von SQL-Anfragen mit Aggregatfunktionen in eine Menge von Abhängigkeiten ebenfalls Tgds erzeugt, die für die Umsetzung dieser Anfragen entscheidend sind. Hierbei sorgen die Tgds für eine Anpassung des eigentlichen Quellschemas, um ein Teilschema zu generieren, welches anschließend mit S-t-tgds ins Zielschema überführt werden kann. Dementsprechend werden die Tgds in der Invertierung benötigt, um eine Rückverfolgung zum eigentlichen Quellschema zu gewährleisten.
Für eine Unterscheidung von S-t-tgds und Tgds werden beide Abhängigkeitsarten in unterschiedlichen Mengen σ_1, σ_2 gespeichert. Dabei enthält σ_1 die S-t-tgds und σ_2 die Tgds. Als Ausgabe liefert der Algorithmus 5 eine invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$, wobei die Menge Ω die invertierten S-t-tgds enthält. In der Ausgabe werden ausschließlich die invertierten S-t-tgds benötigt, da im Algorithmus 5 die Quellrelationen der Tgds in die invertierten S-t-tgds eingefügt werden (Schritt 4).
Ebenfalls werden die invertierten S-t-tgds in Ω nicht als Disjunktiven Mengen zurückgegeben, im Gegensatz zum DirectAdaptedMaxExtendedRecovery-Algorithmus. Diesbezüglich verfolgt der Algorithmus 5 das Ziel die invertierten S-t-tgds möglichst zusammenzufassen und eine minimale Anzahl als S-t-tgds zurückzugeben. Folglich werden mehrere (Teil-)S-t-tgds, die beispielsweise bei der Konvertierung von geschachtelten SQL-Anfragen auftreten können, zu einer invertierten S-t-tgd zusammengefasst. Die invertierten S-t-tgds werden, wie im DirectAdaptedMaxExtendedRecovery-Algorithmus,

in erster Ordnung zurückgegeben, um eine Anwendung in ChaTEAU zu gewährleisten. Im Folgendem werden die einzelnen Schritte des Algorithmus 5 beschrieben:

Erster Schritt. Im erstem Schritt werden alle S-t-tgds und Tgds in eigne Mengen σ'_{1i} und σ'_{2n} eingeordnet, wobei σ'_{1i} jeweils eine S-t-tgd und σ'_{2n} jeweils eine Tgd enthält. Hierbei dienen die unterschiedlichen Mengen für eine Unterscheidung von S-t-tgd und Tgds, da für die Ausgabe ausschließlich die S-t-tgds relevant sind. Ebenfalls wird diese Einteilung in einzelnen Mengen für die Normalisierung im zweitem Schritt benötigt. Im zweitem Schritt erfolgt eine Aufteilung der S-t-tgd und Tgds in mehrere (Teil-)S-t-tgds bzw. (Teil-)Tgds. Entsprechend werden im zweitem Schritt alle (Teil-)S-t-tgds die zu einer (Gesamt-)S-t-tgd gehören in einer Menge σ'_{1i} gespeichert, um die Zugehörigkeit der (Teil-)S-t-tgds zu einer (Gesamt-)S-t-tgd zu erhalten. Diese Zugehörigkeit wird benötigt, um die (Teil-)S-t-tgds im siebten Schritt wieder zusammenzufassen. Die Speicherung der (Teil-)Tgds erfolgt analog.

Zweiter Schritt. Der zweite Schritt ist vom DirectAdaptedMaxExtendedRecovery-Algorithmus übernommen worden. Dabei wird eine Normalisierung in den Köpfen der S-t-tgds und Tgds vorgenommen. Diesbezüglich werden die Konjunktion in den Köpfen der S-t-tgds aufgelöst, indem die S-t-tgds aufgeteilt werden. Bei dieser Aufteilung wird eine S-t-tgd der Form $\alpha \rightarrow (\beta_1 \wedge ... \wedge \beta_n)$ in mehrere S-t-tgds der Form $\alpha \rightarrow \beta_1$,...,$\alpha \rightarrow \beta_n$ zerteilt. Die entstehenden (Teil-)S-t-tgds werden in der Menge σ'_{1n} der ursprünglichen S-t-tgd gespeichert. Die Normalisierung der Tgds erfolgt analog. Eine Ausnahme bilden S-t-tgds mit Gleichungen der Form $x\theta c$ oder Funktionen im Kopf. Diese Gleichungen oder Funktionen werden in alle entstehende (Teil-)S-t-tgds mitübernommen.

Dritter Schritt. Für die eigentliche Invertierung der Abhängigkeiten dient der dritte Schritt. Dieser wurde ebenfalls vom DirectAdaptedMaxExtendedRecovery-Algorithmus übernommen. In diesem Schritt wird die Pfeilrichtung der Abhängigkeiten umgekehrt bzw. Rumpf und Kopf der Abhängigkeiten werden miteinander vertauscht. An dieser Stelle sei anzuführen, dass im drittem Schritt keine Betrachtung der Quantoren vorgenommen wird, anders als beim DirectAdaptedMaxExtendedRecovery-Algorithmus. Die Anpassung der Quantoren erfolgt erst im letztem Schritt (Schritt 9), da durch die Anwendung der folgenden Schritte sich die Relationen in einer Abhängigkeit verändern können. Hierbei müssten die Quantoren der Abhängigkeiten bei jeder Veränderung der Relationen in einer Abhängigkeit angepasst werden. Entsprechend ist eine Anpassung der Quantoren erst im letzten Schritt sinnvoll, um eine richtige Quantifizierung der Abhängigkeiten zu garantieren.

Vierter Schritt. Im viertem Schritt wird eine Zusammenfassung der invertierten S-t-tgds und Tgds realisiert. Dabei wird eine Ersetzung in den Köpfen der S-t-tgds, in den Mengen ω'_{1n}, durchgeführt. Bei dieser Ersetzung werden S-t-tgds der Form $\beta \rightarrow$

$\alpha_1 \wedge ... \wedge \alpha_n$, durch $\beta \rightarrow \alpha_1 \wedge ... \wedge \gamma \wedge ... \wedge \alpha_n$ ersetzt, sollte eine S-t-tgd oder Tgd der From $\alpha_i \rightarrow \gamma$ existieren. Dieser Schritt dient der Zusammenfassung von (Teil-)S-t-tgds, die beispielsweise bei verschachtelten Anfragen auftreten können, sowie für die Rückverfolgung zum ursprünglichen Quellschema bei der Verwendung von Tgds. Für die Veranschaulichung dieser Rückverfolgung wird im Folgenden ein Beispiel von abstrakten Abhängigkeiten (S-t-tgds und Tgds) betrachtet:

$$\begin{array}{ll} \text{Tgd:} & R(a_1, a_2, a_3) \rightarrow R'(a_1, a_2) \\ \text{S-t-tgd:} & R'(a_1, a_2) \rightarrow Res(a_1, a_2) \end{array}$$

Hierbei sorgt die Tgd für eine Anpassung des eigentlichen Quellschema, indem die Relation R in die Relation R' überführt wird. Darauffolgend wird die S-t-tgd angewendet, um aus der Relation R' das Ergebnis Res abzuleiten. Für die Rückverfolgung bzw. für die Durchführung vom viertem Schritt, muss eine Invertierung der Abhängigkeiten durchgeführt werden (Schritt 3). Entsprechend werden folgende invertierte Abhängigkeiten gebildet:

$$\begin{array}{ll} \text{Tgd:} & R'(a_1, a_2) \rightarrow \exists a_3 : R(a_1, a_2, a_3) \\ \text{S-t-tgd:} & Res(a_1, a_2) \rightarrow R'(a_1, a_2) \end{array}$$

Aus diesen invertierten Abhängigkeiten ist ersichtlich, dass die invertierte S-t-tgd eine Abbildung vom Ergebnis Res auf die angepasste Relation R' darstellt. Allerdings wäre für die Invertierung eine Abbildung vom Ergebnis auf das ursprüngliche Quellschema wünschenswert. Folglich sollte die invertierte S-t-tgd vom Ergebnis Res auf die ursprüngliche Relation R abbilden. An dieser Stelle kommt der vierte Schritt zum Einsatz, indem eine Ersetzung von R' mit R in der S-t-tgd durchgeführt wird, da R' die Relation R impliziert. Dementsprechend wird folgende invertierte S-t-tgd gebildet:

$$Res(a_1, a_2) \rightarrow \exists a_3 : R(a_1, a_2, a_3).$$

Fünfter Schritt. Im fünften Schritt werden Relationen entfernt, die im Kopf einer S-t-tgd mehrfach auftreten und die gleichen Variablen mit gleichen Indizes besitzen. Diese Duplikate können durch die Ersetzung im viertem Schritt entstehen und werden gelöscht, da sie keine zusätzlichen Informationen liefern.

Sechste Schritt. Im sechstem Schritt werden alle negierten Relationen aus den Köpfen der invertierten S-t-tgds entfernt. Im Allgemeinen dient die Anwendung einer negierten Relation in einer S-t-tgd, um eine Bedingung aufzustellen. Diese Bedingung besagt, dass das aktuell betrachtete Tupeln oder ein bestimmter Attributwert nicht in der negierten Relation vorhanden sein darf. Für die Veranschaulichung ist im Folgenden eine S-t-tgd als Beispiel angegeben, die eine negierte Relation als Bedingung verwendet:

$$R(a_1, ..., a_n, d) \wedge \neg S(b_1, ..., b_n, d) \rightarrow Res(a_1, ..., a_n).$$

In dieser S-t-tgd werden nur die Tupel der Relation R auf die Ergebnisrelation Res abgebildet, wenn der entsprechende Attributwert d nicht in der Relation S vorhanden ist. Bei der Invertierung dieser S-t-tgd wird die Bedingung bzw. die negierte Relation entfernt, da diese keine Auswirkung auf die Berechnung der minimalen Teildatenbank hat. Ebenfalls können durch die Anwendung der Ersetzung im viertem Schritt redundante negierte Relationen im Kopf der invertierten S-t-tgd auftreten, die entfernt werden müssen. Folglich wird bei der Invertierung des Beispiels folgende invertierte S-t-tgd generiert:

$$Res(a_1, ..., a_n) \rightarrow R(a_1, ..., a_n).$$

Siebter Schritt. Im siebtem Schritt werden die normalisierten S-t-tgds wieder zusammengefügt. Dabei werden nur S-t-tgds zusammengefügt, die in der gleichen Menge ω'_{1n} vorhanden sind. Dieser Schritt dient dafür die Normalisierung der S-t-tgds wieder aufzulösen, um eine minimale Anzahl von S-t-tgds als Lösung der Invertierung anzugeben. Eine Zusammenfassung der S-t-tgds, ist nur möglich, wenn der gleiche Kopf mit gleichen Variablen in beiden S-t-tgds vorhanden ist.

Achter Schritt. Im achtem Schritt werden Gleichheitsprädikate der Form $x = a$ angewendet. Hierbei erfolgt einer Ersetzung der Variablen x durch eine Konstante oder durch eine Variable a.

Neunter Schritt. Schlussendlich werden im neuntem Schritt die Existenz- und Allquantoren betrachtet und angepasst. In diesem Schritt werden alle Variablen, die im Rumpf aber nicht im Kopf einer S-t-tgd vorkommen, existenzquantifiziert. Eine Ausnahme davon, bilden die Variablen, die im siebten Schritt durch Konstanten ersetzt wurde. Diese Variablen werden nicht existenzquantifiziert, da durch die Ersetzung durch eine Konstante der konkrete (Attribut-)Wert der Variablen vorhanden ist.

Algorithmus 5 : Invertierung

Input : Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ einer Menge von S-t-tgds und Tgds entspricht. Hierbei sei $\{\sigma_1, \sigma_2\} \in \Sigma$, wobei σ_1 die Menge der S-t-tgds entspricht und σ_2 die Menge der Tgds.

Output : Invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$ von $\mathcal{M}$, wobei Ω einer Menge von (invertierten) S-t-tgds entspricht.

1. (Einordnung der S-t-tgd und Tgds in Mengen) Ordne jede S-t-tgd und Tgd eine eigene Menge in Σ' zu. Hierbei seien $\sigma'_{1i} \in \Sigma'$ die Mengen der S-t-tgds und $\sigma'_{2n} \in \Sigma'$ die Mengen der Tgds, wobei $(n, i) \in \mathbb{N}^+$.

2. (Normalisiere die Konjunktion in den Köpfen der S-t-tgd und Tgd) Normalisiere alle S-t-tgd und Tgds in Σ', indem alle Formeln der Form $\alpha \rightarrow (\beta_1 \wedge ... \wedge \beta_n)$ durch die Formeln $\alpha \rightarrow \beta_1$,...,$\alpha \rightarrow \beta_n$ ersetzt werden.

3. (Invertierung der S-t-tgds und Tgds) Invertiere die S-t-tgds und Tgds, indem alle normalisierten S-t-tgds und Tgds $\alpha \rightarrow \beta$ durch $\beta \rightarrow \alpha$ ersetzt werden. Sei die resultierende Menge Ω', die die Mengen ω'_{1n} (invertierte S-t-tgds) und ω'_{2n} (invertierte Tgds) enthält, wobei $(n, i) \in \mathbb{N}^+$ gilt.

4. (Ersetzung der Köpfe in den S-t-tgds) Seien eine S-t-tgd der Form $\beta \rightarrow \alpha_1 \wedge ... \wedge \alpha_n$ und eine S-t-tgd oder Tgd der Form $\alpha_i \rightarrow \gamma$ in Ω' gegeben. Dann ersetzte die beiden Formeln durch $\beta \rightarrow \alpha_1 \wedge ... \wedge \gamma \wedge ... \wedge \alpha_n$. Ebenfalls ersetzte in alle anderen Abhängigkeiten α_i durch γ.

5. (Duplikate im Kopf einer S-t-tgd entfernen) Sei eine S-t-tgd der Form $\beta \rightarrow \alpha_1, ..., \alpha_i, \alpha_i, ..., \alpha_n$ gegeben, wobei α_i mehrmals mit den gleichen Variablen auftaucht, dann entferne diese Duplikate. Dabei sei die S-t-tgd der Form $\beta \rightarrow \alpha_1, ..., \alpha_i, ..., \alpha_n$ die Lösung der Duplikateneliminierung.

6. (Entfernen der negierten Relationen) Sei eine S-t-tgd der Form $\beta \rightarrow \alpha_1 \wedge \cdots \wedge \neg\alpha_i \wedge \cdots \wedge \alpha_n$ in Ω' gegeben, dann entferne die negierte Relation $\neg\alpha_i$ aus dem Kopf der S-t-tgd.

7. (Zusammenfügen der S-t-tgd) Seien zwei S-t-tgds $\beta \rightarrow \alpha$ und $\gamma \rightarrow \alpha$, in einer Menge ω'_{1n} mit dem gleichem Kopf vorhanden, dann ersetzte beide durch $\beta \wedge \gamma \rightarrow \alpha$.

8. (Anwendung der Gleichheitsprädikate) Bei einer S-t-tgd der Form $\beta \rightarrow \alpha \wedge x = a$, ersetze alle Variablen x durch den Wert a bzw. durch die Variable a und entferne das Gleichheitsprädikat $x = a$.

9. (Anpassung der Quantoren) Alle Variablen einer S-t-tgd, die im Kopf, aber nicht im Rumpf vorkommen werden existenzquantifiziert. Ebenfalls werden alle Variablen, die durch einen konkreten Wert ersetzt wurden, nicht quantifiziert. Die restlichen Variablen werden allquantifiziert.

Return : $\mathcal{M}' = (T, S, \Omega)$. Dabei sei Ω die Menge der invertierten S-t-tgds, mit $\omega'_{1n} \in \Omega$.

Beispiel. Folgendes Beispiel soll den Algorithmus 5 veranschaulichen. In diesem Beispiel werden die (Teil-)S-t-tgds aus dem Abschnitt 3.3.3 verwendet, die aus der Verschachtelung der Anfrage 3.2 generiert wurden. Dementsprechend wird die Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, mit $S = \{St = \{ma, ln, fn\}, G = \{ma, g\}\}$ und $T = \{R = \{ma, fn, ln, g\}, R' = \{ma, fn, ln, g\}, R'' = \{ln, fn\}\}$ als Eingabe betrachtet. Für die Veranschaulichung des Beispiels, wurde die Anzahl der Variablen in den Relationen vereinfacht. Folglich wurde die Relation *Students* (kurz: *St*) auf die Attribute/Variablen *MatricNo*, *LastName* und *FirstName* beschränkt und die Relation *Grades* (kurz: *G*) auf die Attribute *MatricNo* und *Grade* vereinfacht. Die verwendeten S-t-tgds in Σ sind folglich definiert:

$$St(ma, ln, fn) \wedge G(ma, g) \rightarrow R(ma, fn, ln, g)$$
$$R(ma, fn, ln, g) \wedge g = 1.3 \rightarrow R'(ma, fn, ln, 1.3)$$
$$R'(ma, fn, ln, 1.3) \rightarrow R''(ln, fn).$$

Im erstem Schritt wird jede S-t-tgd in eine eigene Menge σ_{1n} eingeordnet. In diesem Beispiel sind drei S-t-tgds als Abhängigkeiten gegeben, entsprechend werden folgende drei Mengen definiert:

$$\sigma'_{11} = \{St(ma, ln, fn) \wedge G(ma, g) \rightarrow R(ma, fn, ln, g)\}$$
$$\sigma'_{12} = \{R(ma, fn, ln, g) \wedge g = 1.3 \rightarrow R'(ma, fn, ln, 1.3)\}$$
$$\sigma'_{13} = \{R'(ma, fn, ln, 1.3) \rightarrow R''(ln, fn)\}.$$

Im zweitem Schritt wird eine Normalisierung der Köpfe der S-t-tgds durchgeführt. Allerdings enthalten die S-t-tgds im Beispiel nur ein Relationssymbol im Kopf, dementsprechend ist keine Normalisierung notwendig bzw. die S-t-tgds sind schon normalisiert. Auf die Normalisierung folgt der dritte Schritt, die Invertierung. Folglich werden alle S-t-tgds invertiert, indem die Pfeilrichtung umgekehrt wird und Rumpf und Kopf vertauscht werden. Im Folgenden werden die invertierten S-t-tgds dargestellt, ohne die Betrachtung von Quantoren:

$$\omega'_{11} = \{R(ma, fn, ln, g) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\}$$
$$\omega'_{12} = \{R'(ma, fn, ln, 1.3) \rightarrow R(ma, fn, ln, g) \wedge g = 1.3\}$$
$$\omega'_{13} = \{R''(ln, fn) \rightarrow R'(ma, fn, ln, 1.3)\}.$$

Anschließend wird im viertem Schritt die Ersetzung in den Köpfen der S-t-tgds durchgeführt. Mit diesem Schritt wird die Zusammenfassung der einzelnen invertierten (Teil-)S-t-tgd in eine invertierte (Gesamt-)S-t-tgd realisiert. Im Beispiel wird zuerst die Relation R in der Menge ω'_{12} durch den Kopf der ersten S-t-tgd in der Menge ω'_{11} ersetzt. Diese Ersetzung ist möglich, da die Relation R im Kopf der einen S-t-tgd und im Rumpf der anderen S-t-tgd vorhanden ist. Das Ergebnis der Ersetzung ist im Folgendem dargestellt:

$$\omega'_{12} = \{R'(ma, fn, ln, 1.3) \rightarrow St(ma, ln, fn) \wedge G(ma, g) \wedge g = 1.3\}$$
$$\omega'_{13} = \{R''(ln, fn) \rightarrow R'(ma, fn, ln, 1.3)\}.$$

Hierbei sei anzuführen, dass die S-t-tgd mit der Relation R im Rumpf bei der Ersetzung gelöscht wird, um eine mögliche erneute Ersetzung über dieser S-t-tgd zu verhindern. Infolge des vierten Schrittes wird eine weitere Ersetzung durchgeführt werden. Bei dieser Ersetzung wird die Relation R' in der Menge ω'_{13} durch die Konjunktion der Relationen St und G ersetzt. Diese Konjunktion entspricht dem Kopf der S-t-tgd in der Menge ω'_{12} mit der Relation R' im Rumpf. Das Ergebnis der Ersetzung ist folglich dargestellt:

$$\omega'_{13} = \{R''(ln, fn) \rightarrow St(ma, ln, fn) \wedge G(ma, g) \wedge g = 1.3\}.$$

Hierbei wurde die S-t-tgd mit der Relation im Rumpf, über die die Ersetzung durchgeführt wird, wieder entfernt. Schließlich ist nach dem viertem Schritt eine invertierte (Gesamt-)S-t-tgd, aus den (Teil-)S-t-tgd generiert worden, die das Ergebnis (Relation R'') auf die ursprünglichen Quellrelationen (Konjunktion von St und G) abbildet. Diese (Gesamt-)S-t-tgd enthält keine Duplikate von Relationen und kann nicht weiter zusammengefasst werden. Außerdem enthält die (Gesamt-)S-t-tgd keine negierten Relationen im Rumpf. Entsprechend bewirken der fünfte, sechste und siebte Schritt keine Veränderung an der S-t-tgd. Folglich wird der achte Schritt angewendet, der die Ersetzung des Gleichheitsprädikat $g = 1.3$ umsetzt. Dabei wird die Variable g durch den konkreten Wert 1.3 ersetzt. Das Ergebnis dieses Schrittes ist folglich dargestellt:

$$\omega'_{13} = \{R''(ln, fn) \rightarrow St(ma, ln, fn) \wedge G(ma, 1.3)\}.$$

Schlussendlich wird für die vollständige Invertierung der neunte Schritt angewendet, der die Existenz- und Allquantoren setzt. Demnach wird die Variable ma existenzquantifiziert, da diese Variable im Kopf aber nicht im Rumpf der S-t-tgd auftaucht. Alle anderen Variablen werden allquantifiziert, wobei auch die Variable g allquantifiziert wird, da diese durch einen konkreten Wert ersetzt worden ist. Abschließend wird eine invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$ zurückgegeben mit:

$$\omega'_{13} = \{R''(ln, fn) \rightarrow \exists ma : St(ma, ln, fn) \wedge G(ma, 1.3)\} \in \Omega.$$

Algorithmus 5 betrachtet einige Ausnahmen der Invertierung, die mit zusätzlichen Schritten abgefangen werden. Beispielsweise ist die Ersetzung im viertem Schritt nur bei der Betrachtung von zusätzlichen Tgds notwendig, wie sie etwa bei den Aggregatfunktionen auftreten. Dementsprechend sind bei der Invertierung einer einzelnen S-t-tgd einige Schritte im Algorithmus 5 überflüssig. Für diesem Fall, dass nur die Invertierung einer einzelnen S-t-tgd benötig wird, kann der Algorithmus 5 auf dem Algorithmus 6 vereinfacht werden. Algorithmus 6 veranschaulicht eine Umsetzung des einfachen allgemeinen

Algorithmus (Abschnitt 4.2.1) für die Invertierung. Als Eingabe erhält der Algorithmus eine Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ nur eine einzelne S-t-tgd enthält. Im erstem Schritt wird die S-t-tgd invertiert, indem die Pfeilrichtung umgekehrt wird und Kopf und Rumpf getauscht werden, wie im drittem Schritt des Algorithmus 5. Anschließend erfolgt die Anwendung der Gleichheitsprädikate und die Anpassung der Quantoren, wie im achtem und neunten Schritt im Algorithmus 5. Schlussendlich ist mit Anwendung dieser Schritte, die Invertierung der einzelnen S-t-tgd abgeschlossen. Entsprechend wird als Ausgabe eine Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$ zurückgegeben, mit der invertierten S-t-tgd als Abhängigkeit in Ω.

Algorithmus 6 : VereinfachungInvertierung

Input : Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ eine S-t-tgd entspricht.

Output : Invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$ von $\mathcal{M}$, wobei Ω eine invertierten S-t-tgd entspricht.

1. (Invertierung der S-t-tgds und Tgds) Invertiere die S-t-tgd, indem alle S-t-tgds der Form $\alpha \rightarrow \beta$ durch $\beta \rightarrow \alpha$ ersetzt werden.

2. (Anwendung der Gleichheitsprädikate) Ersetze bei einer S-t-tgd der Form $\beta \rightarrow \alpha \wedge x = a$ alle Variablen x durch den Wert a bzw. durch die Variable a und entferne das Gleichheitsprädikat $x = a$.

3. (Anpassung der Quantoren) Alle Variablen einer S-t-tgd, die im Kopf aber nicht im Rumpf vorkommen, werden existenzquantifiziert. Ebenfalls werden alle Variablen die durch einen konkreten Wert ersetzt wurden nicht quantifiziert. Die restlichen Variablen werden allquantifiziert.

Return : $\mathcal{M}' = (T, S, \Omega)$.

4.2.4 Algorithmus Why-Provenance

Im vorherigen Abschnitt 4.2.3 wurde ein Algorithmus für die Invertierung beschrieben, ohne weitere Betrachtung von Data Provenance. Allerdings ist die Verwendung von Provenance-Informationen für die Invertierung von Anfragen sinnvoll und notwendig, um die invertierten Abhängigkeiten zu erweitern. Diese Erweiterung der invertierten Abhängigkeiten dient einer genaueren Berechnung der minimalen Teilmengen. Diesbezüglich wird in diesem Abschnitt ein Algorithmus für die Invertierung mithilfe von Why-Provenance beschrieben. Die Why-Provenance dient bei der Invertierung der Rekonstruktion der verwendeten Tupelanzahl im Quellschema und spielt bei der Umsetzung der Aggregatfunktionen eine zentrale Rolle. Im Algorithmus 7 ist eine Verwendung der Why-Provenance für die Invertierung veranschaulicht. Die Grundidee des Algorithmus 7 ist die invertierte S-t-tgd ohne Provenance zu berechnen und anschließend mit Provance-Informationen, in Form von einer Side-Table S, zu erweitern. Hierbei dient der Algo-

rithmus 5 für die Berechnung der invertierten S-t-tgd ohne Provenance, dessen Ergebnis mit zusätzlichen Informationen (Side-Tables) erweitert wird. Als Eingabe erhält der Algorithmus 7 eine Schemaabildung $\mathcal{M} = (S, T, \Sigma)$ mit einer Menge von Abhängigkeiten Σ, die S-t-tgds und Tgds enthält. Für die Speicherung von Provenance-Informationen ist es entscheidend, dass alle Tupel im Quellschema S eine eindeutige Tupel-ID besitzen. Im Fall der Why-Provenance werden die Tupel-IDs der Tupels, die bei der Ausführung der Abhängigkeiten benötigt werden, in einer Side-Table S gespeichert. Folglich benötigt der Algorithmus eine Schemaabildung $\mathcal{M} = (S, T, \Sigma)$, mit eindeutigen Tupel-IDs, und eine Side-Table S als Eingabe. Als Ausgabe liefert der Algorithmus eine invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$, wobei Ω die invertierten Abhängigkeiten mit Why-Provenance enthält. Im Gegensatz zum Algorithmus 5 sind in Ω sowohl S-t-tgds als auch Tgds enthalten. In der Ausgabe sind Tgds der Form $S(t) \wedge \beta \wedge \neg S'(t) \rightarrow \alpha' \wedge S'(t)$ enthalten, die ein Zwischenergebnis α' der Invertierung generieren (Schritt 2). Dieses Zwischenergebnis enthält, mithilfe der Anwendung der Side-Tables S und $\neg S'$, die Tupelanzahl der verwendeten Tupeln im Quellschema. Des weiterem enthält die Ausgabe S-t-tgds der Form $\alpha' \rightarrow \alpha$, die ein Zwischenergebnis α' auf ein Ergebnis der Invertierung α abbilden (Schritt 3). Diese Methodik eine zusätzliche Tgd zu definieren, die zuerst ein Zwischenergebnis generiert statt direkt mit einer S-t-tgd auf das Ergebnis abzubilden, dient für die Vermeidung von Side-Tables Definitionen in der S-t-tgd. Im Folgendem werden die einzelnen Schritte des Algorithmus 7 betrachtet:

Erster Schritt. Im ersten Schritt wird die Invertierung der Eingabe ohne die Verwendung von Provenance-Informationen durchgeführt. Für diese Durchführung wird der Algorithmus 5 (Abschnitt 4.2.3) verwendet, der eine invertierte S-t-tgd (ohne Provenance) als Ausgabe liefert.

Zweiter Schritt. Im zweitem Schritt wird die Why-Provenance zu den invertierten S-t-tgds, die im ersten Schritt generiert worden sind, hinzugefügt. Hierbei wird für jede S-t-tgd der Form $\beta \rightarrow \alpha$, eine Tgd der Form $S(t) \wedge \beta \wedge \neg S'(t) \rightarrow \alpha' \wedge S'(t)$ erzeugt. Die Grundidee ist, dass die Tgd neue Tupel in Höhe der Anzahl von Tupel-IDs t in der Side-Table S zum Zwischenergebnis α' hinzufügt. Bei dem Zwischenergebnis α' wird eine Kopie von α aus der ursprünglichen S-t-tgd generiert, indem α' die gleichen Variablen beinhaltet wie α. Um eine doppelte Verwendung von einer Tupel-IDs t zu vermeiden, wird eine neue leere Side-Table $S'(t)$ definiert. In diese neue Side-Table werden alle Tupel-IDs, die bereits betrachtet worden sind, gespeichert. Folglich wird im Rumpf der Tgd mit $\neg S'(t)$ abgefragt, ob eine Verwendung des Tupels mit der Tupel-ID t schon stattgefunden hat. Für den Fall, dass keine Verwendung des Tupels mit der Tupel-ID t stattgefunden hat, wird das Tupel zum Zwischenergebnis α' hinzugefügt und die Tupel-ID t wird in S' aufgenommen. Im Gegensatz dazu, wird im Fall einer erneuten Betrachtung einer Tupel-ID t keine neues Tupel in α' erzeugt, da t schon in S' vorhanden ist und somit $\neg S'(t)$ nicht erfüllt wird.

Dritter Schritt. Im drittem Schritt für jede Tgd der Form $S(t) \wedge \beta \wedge \neg S'(t) \rightarrow \alpha' \wedge S'(t)$ eine S-t-tgd der Form $\alpha' \rightarrow \alpha$ generiert. Hierbei wird eine direkte Abbildung von α' auf α durchgeführt.

Algorithmus 7 : InvertierungWhyProvenance

Input : Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ einer Menge von S-t-tgds und Tgds entspricht. Hierbei sei $\{\sigma_1, \sigma_2\} \in \Sigma$, wobei σ_1 die Menge der S-t-tgds entspricht und σ_2 die Menge der Tgds.

Output : Invertierte Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$ von $\mathcal{M}$, wobei Ω einer Menge von S-t-tgds und Tgds entspricht.

1. (Ausführung der Invertierung). Führe die Invertierung ohne Why-Provnance mit dem Algorithmus 5 durch. Sei die Menge Ω' die Lösung des Algorithmus 5.

2. (Erweiterung mit Why-Provenance). Erstelle aus einer S-t-tgd $\beta \rightarrow \alpha$ von Ω', eine Tgd der Form $S(t) \wedge \beta \wedge \neg S'(t) \rightarrow \alpha' \wedge S'(t)$. Hierbei stellt $S'(t)$ eine leere Side-Table dar und α' eine Kopie von α dar.

3. (Generiere S-t-tgd). Füge eine S-t-tgd der Form $\alpha' \rightarrow \alpha$ für alle Tgds der Form $S(t) \wedge \beta \wedge \neg S'(t) \rightarrow \alpha' \wedge S'(t)$ hinzu.

Return : $\mathcal{M}' = (T, S, \Omega)$

Beispiel. Für die Veranschaulichung des Prinzips von Algorithmus 7 wird im Folgenden ein Beispiel durchgeführt. Als Beispiel für die Invertierung mit Why-Provenance werden die Abhängigkeiten betrachtet, die bei der Durchführung der Aggregatfunktion für die Berechnung des Durchschnittes (AVG) benötigt werden. Infolgedessen wird die Invertierung folgender SQL-Anfrage beispielhaft durchgeführt:

```
SELECT AVG(Grade)
FROM Grades
WHERE ModulNo = '002'
```

In dieser Anfrage wird der Notendurchschnitt bzw. der Durchschnitt des Attributes *Grade* für alle Studierende berechnet, die am zweiten Modul teilgenommen haben. Als Eingabe erhält Algorithmus 7 eine Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei $S = \{G = \{id_g, mo, ma, se, g\}, G_{ord} = \{x, g, id_g\}, H = \{a, b\}\}$ und $T = \{Res = \{d\}\}$ entspricht. Außerdem enthält Σ folgende Abhängigkeiten, die für die Umsetzung der SQL-Anfrage benötigt werden:

Tgds: $$G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg G_{ord}(x_j, g_j, id_{g_j}) \rightarrow S'(id_{g_i}) \wedge G_{ord}(1, g_i, id_{g_i})$$
$$G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j}) \rightarrow S'(id_{g_i}) \wedge G_{ord}(x+1, g_i, id_{g_i}) \wedge S(id_{g_j})$$

Tgds: $$G_{ord}(1, g, id_g) \rightarrow S''(id_g) \wedge H(g, 1)$$
$$G_{ord}(x, g, id_g) \wedge \neg S''(id_g) \wedge H(c, x-1) \rightarrow S''(id_g) \wedge H(f(c, g), x) \wedge f(c, g) = g + c$$

S-t-tgd: $$H(c_i, x) \wedge \neg H(c_j, x+1) \rightarrow Res(d) \wedge d = c_i/x$$

Diese Abhängigkeiten wurden mithilfe der Definitionen bzw. der Methodik in [Aug22a] erstellt. Allerdings wurde für die Vereinfachung die Relation G_{ord}, die eine Ordnung der Tupeln in der Relation $Grades$ (kurz: G) realisiert, gekürzt. Dementsprechend enthält die Relation G_{ord}, nur die Attribute $Grade$, ID_{Grades} und ein Attribut/Variable x für die Ordnung. In der Definition der Abhängigkeiten entspricht die ID_{Grades} (kurz: id_g) die Tupel-IDs für die Umsetzung der Why-Provenance. Ebenfalls erhält der Algorithmus eine Side-Table $S = \{G_5, G_6, G_7, G_8, G_9, G_{10}, G_{11}\}$ als Eingabe, die alle Tupel-IDs der Studierenden im zweitem Modul enthält.
Im erstem Schritt bei der Invertierung der Abhängigkeiten wird der Algorithmus 5, mit der Schemaabbildung $\mathcal{M}$ als Eingabe, ausgeführt. Als erstem Schritt des Algorithmus 5 werden die Tgds und S-t-tgd in unterschiedlichen Mengen eingeteilt, um eine Unterscheidung der beiden Abhängigkeitsarten zu gewährleisten. Das Ergebnis dieser Einteilung ist folglich dargestellt:

$$\sigma'_{21} = \{G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg G_{ord}(x_j, g_j, id_{g_j}) \rightarrow S'(id_{g_i}) \wedge G_{ord}(1, g_i, id_{g_i})\}$$
$$\sigma'_{22} = \{G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j}) \rightarrow S'(id_{g_i}) \wedge G_{ord}(x+1, g_i, id_{g_i}) \wedge S(id_{g_j})\}$$

$$\sigma'_{23} = \{G_{ord}(1, g, id_g) \rightarrow S''(id_g) \wedge H(g, 1)\}$$
$$\sigma'_{24} = \{G_{ord}(x, g, id_g) \wedge \neg S''(id_g) \wedge H(c, x-1) \rightarrow S''(id_g) \wedge H(f(c, g), x) \wedge f(c, g) = g + c\}$$

$$\sigma'_{11} = \{H(c_i, x) \wedge \neg H(c_j, x+1) \rightarrow Res(d) \wedge d = c_i/x\}.$$

Als zweiten Schritt der Invertierung ohne Provenance mit Algorithmus 5 wird die Normalisierung der Köpfe in den Tgds und S-t-tgds durchgeführt. Die Normalisierung liefert folgendes Ergebnis:

$$
\begin{aligned}
\sigma'_{21} = \quad & \{(G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg G_{ord}(x_j, g_j, id_{g_j}) \rightarrow S(id_{g_i})), \\
& (G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg G_{ord}(x_j, g_j, id_{g_j}) \rightarrow G_{ord}(1, g_i, id_{g_i}))\} \\
\sigma'_{22} = \quad & \{(G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j}) \\
& \rightarrow S(id_{g_j})), \\
& (G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j}) \\
& \rightarrow S'(id_{g_i})), \\
& (G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j}) \\
& \rightarrow G_{ord}(x+1, g_i, id_{g_i}))\}
\end{aligned}
$$

$$
\begin{aligned}
\sigma'_{23} = \quad & \{(G_{ord}(1, g, id_g) \rightarrow S''(id_g)), \\
& (G_{ord}(1, g, id_g) \rightarrow H(g, 1))\} \\
\sigma'_{24} = \quad & \{(G_{ord}(x, g, id_g) \wedge \neg S''(id_g) \wedge H(c, x-1) \rightarrow S''(id_g) \wedge f(c, g) = g + c), \\
& (G_{ord}(x, g, id_g) \wedge \neg S''(id_g) \wedge H(c, x-1) \rightarrow H(f(c, g), x) \wedge f(c, g) = g + c)\}
\end{aligned}
$$

$$
\sigma'_{11} = \quad \{(H(c, x) \wedge \neg H(c_j, x+1) \rightarrow Res(d) \wedge d = c/x)\}.
$$

Darauffolgend wird eine Invertierung der normalisierten S-t-tgds und Tgds durchgeführt, indem die Pfeilrichtung umgekehrt wird. Die Invertierung der Tgds und S-t-tgds ist durch folgende Abhängigkeiten definiert:

$$
\begin{aligned}
\omega'_{21} = \quad & \{(S(id_g) \rightarrow G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg G_{ord}(x_j, g_j, id_{g_j})), \\
& (G_{ord}(1, g_i, id_{g_i}) \rightarrow G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg G_{ord}(x_j, g_j, id_{g_j}))\} \\
\omega'_{22} = \quad & \{(S(id_g) \rightarrow \\
& G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j})), \\
& (S'(id_{g_i}) \rightarrow \\
& G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j}), \\
& (G_{ord}(x+1, g_i, id_{g_i}) \rightarrow \\
& G(id_{g_i}, 002, ma_i, se_i, g_i) \wedge \neg S'(id_{g_i}) \wedge G_{ord}(x_j, g_j, id_{g_j}) \wedge \neg S(id_{g_j}))\}
\end{aligned}
$$

$$
\begin{aligned}
\omega'_{23} = \quad & \{(S''(id_g) \rightarrow G_{ord}(1, g, id_g)), \\
& (H(g, 1)) \rightarrow G_{ord}(1, g, id_g)\} \\
\omega'_{24} = \quad & \{(S''(id_g) \wedge f(c, g) = g + c \rightarrow G_{ord}(x, g, id_g) \wedge \neg S''(id_g) \wedge H(c, x-1)), \\
& (H(f(c, g), x) \wedge f(c, g) = g + c \rightarrow G_{ord}(x, g, id_g) \wedge \neg S''(id_g) \wedge H(c, x-1))\}
\end{aligned}
$$

$$
\omega'_{11} = \quad \{(Res(d) \wedge d = c/x \rightarrow H(c, x) \wedge \neg H(c_j, x+1))\}.
$$

Anschließend erfolgt der vierte Schritt vom Algorithmus 5, wobei eine Ersetzung der Relationen durchgeführt wird. Diese Ersetzung verfolgt das Ziel, eine Abbildung von der Relation *Res* auf die ursprüngliche Anfangsrelation G (bzw. *Grades*) zu garantieren. Für die einfachere Veranschaulichung dieses Schrittes wird in diesem Beispiel nur eine Variante mit den entscheidenden Ersetzungen beschrieben. Eine Variante für die Ersetzung

ist die folgende Tgd zu verwenden, um eine Ersetzung von G_{ord} mit G durchzuführen:

$$G_{ord}(1, g, id_g) \rightarrow G(id_g, 002, ma, se, g).$$

Infolgedessen wird G_{ord} in allen Abhängigkeiten durch G ersetzt. Anschließend wird eine weitere Ersetzung durchgeführt, wobei durch folgende Tgd eine Ersetzung von der Relation H mit Relation G realisiert wird:

$$H(g, 1)) \rightarrow G(id_g, 002, ma, se, g).$$

Schlussendlich kann durch diese Ersetzung die Relation H durch die Relation G ersetzt werden, welches zu folgende (Ergebnis-)S-t-tgd führt:

$$Res(d) \wedge d = c/x \rightarrow G(id_g, 002, ma, se, g).$$

Mit diesem Schritt ist die Invertierung ohne Provenance im wesentlichen abgeschlossen. Der einzige Schritt vom Algorithmus 5 der noch durchführbar ist, ist der neunte Schritt. Im neunten Schritt werden die Existenz- und Allquantoren angepasst und definiert. Entsprechend wird folgende S-t-tgd als Ergebnis vom Algorithmus 5 ausgegeben:

$$Res(d) \wedge d = c/x \rightarrow \exists id_g, ma, se, g : G(id_g, 002, ma, se, g).$$

Hiermit ist die Durchführung des ersten Schrittes vom Algorithmus 7 abgeschlossen. Im zweitem Schritt wird für die obige (Ergebnis-)S-t-tgd vom ersten Schritt, folgende Tgd definiert:

$$S(id_g) \wedge Res(d) \wedge d = c/x \wedge \neg S'(id_g) \rightarrow \exists id_g, ma, se, g : G'(id_g, 002, ma, se, g) \wedge S'(id_g).$$

Schlussendlich wird im drittem Schritt folgende S-t-tgd definiert:

$$G'(id_g, 002, ma, se, g) \rightarrow G(id_g, 002, ma, se, g).$$

Als Ausgabe liefert der Algorithmus 7 eine Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$, wobei Ω die Tgd (Schritt 2) und die S-t-tgd (Schritt 3) enthält.

In den vorherigen Abschnitten wurde die automatische Invertierung von Abhängigkeiten betrachtet. Hierbei wurden zwei Invertierungsmethoden (Algorithmus 5 und Algorithmus 7) entwickelt. Bei der Anwendung einer automatischen Invertierung bzw. bei der Berechnung der minimalen Teildatenbank in ProSA ist eine offene Problemstellung aufgetreten. Diese offene Problemstellung wird im folgenden Abschnitt beschrieben.

4.3 Offene Problemstellung

Eine offene Problemstellung bei der Invertierung in ProSA wurde im Rahmen der Bachelorarbeit: *Ein Rahmengerüst für ProSA* [Han22], von Moritz Hanzig, entdeckt. In dieser Arbeit wird ein Gesamtgerüst für ProSA entwickelt, wobei die einzelnen Komponenten von ProSA, beispielsweise der Parser und der Invertierer, zusammengefügt werden. Infolgedessen wurde festgestellt, dass bei der Invertierung einer S-t-tgd Duplikate im Zielschema auftreten können. Diese Duplikate werden durch die Anwendung der Tupel-IDs generiert, die in ProSA für die eindeutige Identifikation der Tupel verwendet werden. In ProSA werden diese Tupel-IDs automatisch zu allen verwendeten Relationen hinzugefügt. Hierbei besteht der Fehler bei der Invertierung daraus, dass die Tupel-IDs nicht als Schlüsselattribut betrachtet werden. Dementsprechend können Duplikate mit der gleichen Tupel-ID bei der Invertierung bzw. bei der Berechnung der minimalen Teildatenbank auftreten, da die Tupel-ID nicht als eindeutiges Attribut betrachtet wird. An dieser Stelle sei anzuführen, dass die Definitionen der invertierte S-t-tgds, die durch die Umsetzung des Algorithmus 5 generiert wurden, korrekt sind. Folglich ist das Problem der Duplikate ein Implementierungsproblem in ProSA, worauf die gewählte Invertierungsmethode (Algorithmus 5) bzw. die Definition der invertierten S-t-tgds keinen Einfluss besitzt.

Beispiel. Für die Veranschaulichung des Problems der Entstehung von Duplikaten in der minimalen Teildatenbank wird im Folgendem ein Beispiel durchgeführt. Hierbei wird die Anwendung der folgenden SQL-Anfrage betrachtet, ähnlich wie in [Han22]:

```
SELECT FirstName, LastName, Grade
FROM Students NATURAL JOIN Grades
WHERE FirstName = 'Max'
```

Diese SQL-Anfrage führt einen (natürlichen) Verbund der beiden Relationen *Students* und *Grades* durch und liefert als Anfrageergebnis die Noten aller Studierender mit dem Vornamen 'Max'. Hierbei wird diese SQL-Anfrage durch folgende S-t-tgd repräsentiert:

$$St(id_s, ma, ln, \text{'Max'}) \wedge G(id_g, ma, mo, g) \rightarrow Res(\text{'Max'}, ln, g, id_s, id_g).$$

In dieser S-t-tgd entsprechen die Attribute/Variablen id_s und id_g der Tupel-IDs, die in ProSA automatisch hinzugefügt werden. In diesem Beispiel werden die Tupel-IDs ebenfalls in die Ergebnisrelation *Res* mitaufgenommen. Für die Umsetzung der S-t-tgds bzw. der SQL-Anfrage werden folgende Relationen als Beispiel betrachtet:

G:

id_g	ma	mo	g
G_{11}	7	002	3.3
G_7	3	002	2.3
G_{13}	3	004	1.3

St:

id_s	ma	ln	fn
S_7	7	Mustermann	Max
S_3	3	Müller	Max

Entsprechend enthält die Relation *Students* (kurz: *St*) zwei Studierende mit dem Vornamen 'Max', die insgesamt an drei Prüfungen in der Relation *Grades* (kurz: *G*) teilgenommen haben. Hierbei hat der Studierende Max Müller an zwei Prüfungen, im zweitem und viertem Modul, teilgenommen und der Studierende Max Mustermann hat ausschließlich an einer Prüfung, im zweitem Modul, teilgenommen. Bei der Anwendung der S-t-tgd auf diese Relationen wird folgende Ergebnisrelation gebildet:

$Res:$

fn	ln	g	id_s	id_g
Max	Mustermann	3.3	S_7	G_{11}
Max	Müller	2.3	S_3	G_7
Max	Müller	1.3	S_3	G_{13}

Diesbezüglich wird durch Anwendung der S-t-tgd eine Ergebnisrelation *Res* gebildet die drei Tupel enthält. Dabei ist der Studierende Max Müller doppelt enthalten, da dieser an zwei Prüfungen teilgenommen hat. Bei der Invertierung der S-t-tgd, beispielsweise durch Anwendung des Algorithmus 5, wird folgende invertierte S-t-tgd erzeugt:

$$Res(\text{'Max'}, ln, g, id_s, id_g) \rightarrow \exists ma, mo : St(id_s, ma, ln, \text{'Max'}) \wedge G(id_g, ma, mo, g).$$

Dementsprechend realisiert die invertierte S-t-tgd eine Rückabbildung von der Ergebnisrelation *Res* auf die Quellrelationen *Students* und *Grades*. Hierbei werden die Attribute *MatricNo* (kurz: *ma*) und *ModulNo* (kurz: *mo*) existenzquantifiziert, da diese Attribute nicht in Ergebnisrelation *Res* vorhanden sind. Folglich werden durch die Anwendung der invertierten folgende Relationen als minimale Teildatenbank gebildet:

G:

id_g	ma	mo	g
G_{11}	η_1	η_2	3.3
G_7	η_3	η_4	2.3
G_{13}	η_5	η_6	1.3

St:

id_s	ma	ln	fn
S_7	η_1	Mustermann	Max
S_3	η_3	Müller	Max
S_3	η_5	Müller	Max

Entsprechend werden durch die Anwendung der invertierten S-t-tgd die zwei Relationen *Students* und *Grades* rekonstruiert, wobei die existenzquantifizierten Variablen/Attribute durch entsprechenden Nullwerte η_n ersetzt werden. Bei dieser Rekonstruktion ist ersichtlich, dass in der Relation *Students* ein Duplikat vorhanden ist. In der Relation *Students* existieren nach der Anwendung der invertierten S-t-tgds zwei Tupel, die die Tupel-ID S_3 aufweisen und den Studierenden Max Müller repräsentieren. Dementsprechend ist einer dieser Tupel redundant, da in der eigentlichen Relation *Students* ausschließlich ein Tupel pro Studierenden vorhanden ist. Folglich sind die beiden Nullwerte η_3 und η_5 gleichzusetzen, da diese Nullwerte beide die gleiche Matrikelnummer repräsentieren. Die Duplikate in der minimalen Teilbanken entstehen dadurch, dass die Tupel-IDs nicht als eindeutige Schlüsselattribute behandelt werden. Infolgedessen können bei der Invertierung, wie in diesem Beispiel, mehrere Tupel mit der gleichen Tupel-ID entstehen.

Lösungsansatz. Ein möglicher Lösungsansatz wird in [Han22] skizziert, wobei zusätzliche Egds hinzugefügt werden. Diese Egds definieren die entsprechenden Tupel-IDs als eindeutige (Schlüssel-)Attribute, indem alle Attribute innerhalb der Tupel mit der gleichen Tupel-ID gleichgesetzt werden. Exemplarisch für diesen Ansatz werden folgende Egds zu der S-t-tgd im obigen Beispiel hinzugefügt:

$$St(id_s, ma_1, ln_1, fn_1) \wedge St(id_s, ma_2, ln_2, fn_2) \rightarrow ma_1 = ma_2, ln_1 = ln_2, fn_1 = fn_2$$
$$G(id_g, ma_1, mo_1, g_1) \wedge G(id_g, ma_2, mo_2, g_2) \rightarrow ma_1 = ma_2, mo_1 = mo_2, g_1 = g_2.$$

Entsprechend werden durch die Anwendung der Egds alle Attribute mit der gleichen Tupel-ID gleichgesetzt. Folglich werden die Tupel-IDs eindeutig und die Duplikate werden entfernt bzw. verhindert. Diese Anwendung der Egds auf das obige Beispiel liefert folgende Relationen als Ergebnis:

G:

id_g	ma	mo	g
G_{11}	η_1	η_2	3.3
G_7	η_3	η_4	2.3
G_{13}	η_5	η_6	1.3

St:

id_s	ma	ln	fn
S_7	η_1	Mustermann	Max
S_3	η_3	Müller	Max

Infolgedessen wird das Duplikate in der Relation *Students* entfernt. Allerdings muss anschließend noch eine Anpassung der Nullwerte erfolgen. Im Beispiel ist noch der Nullwert η_5 als Matrikelnummer enthalten, der durch die Entfernung des Duplikats keinen Verbundpartner in der Relation *Students* mehr besitzt. Entsprechend muss der Nullwert η_5 durch den Nullwert η_3 ersetzt werden, um die eigentlichen Verbundpartner wieder zu rekonstruieren. Diese Anpassung der Nullwerte liefert folgende Relationen für das obige Beispiel:

G:

id_g	ma	mo	g
G_{11}	η_1	η_2	3.3
G_7	η_3	η_4	2.3
G_{13}	η_3	η_6	1.3

St:

id_s	ma	ln	fn
S_7	η_1	Mustermann	Max
S_3	η_3	Müller	Max

In den vorherigen Abschnitten wurden eigene Methoden für die automatische Invertierung von Abhängigkeiten bzw. von Schemaabildungen entwickelt. Diese Methoden sollen in ProSA verwendet werden, um die Invertierung der SQL-Anfragen bzw. der Abhängigkeiten durchzuführen. Dementsprechend wird im folgenden Kapitel die konkrete Implementierung der automatischen Invertierungsmethoden (Algorithmus 5 und Algorithmus 7) betrachtet.

5 Implementierung

In diesem Kapitel wird die Implementierung der Algorithmen für die Invertierung (Algorithmus 5 und Algorithmus 7) von einer Menge von Abhängigkeiten beschrieben. Die Implementierung der Algorithmen wurde in der Programmiersprache *Java* durchgeführt und zum Projekt ProSA [Aug22c] hinzugefügt. Ebenfalls ist der Quelltext für die Umsetzung der beiden Algorithmen im Anhang A dargestellt. Hierbei wurde eine neue Java-Klasse *InvertingTechniques* erstellt, die in der Methode *invert* die Umsetzung des Algorithmus 5 und in der Methode *invertAgg* die Umsetzung des Algorithmus 7 enthält. Als Eingabe verwenden diese Methoden eine Menge von S-t-tgds und eine Menge von Tgds. Als Ausgabe liefern die Methoden eine Menge von invertierten S-t-tgds und im Fall der Methode *invertAgg* eine zusätzliche Tgd zurück. Für die Umsetzung der S-t-tgds und Tgds werden die entsprechenden Klassen von ChaTEAU [Aug22b] übernommen. Der Aufbau dieser Klassen wird im Abschnitt 5.1 veranschaulicht. Des weiterem wird die konkrete Umsetzung des Algorithmus 5 in Abschnitt 5.2 beschrieben und die Umsetzung des Algorithmus 7 wird in Abschnitt 5.3 betrachtet.

5.1 ChaTEAU-Abhängigkeiten

Für die Implementierung der Invertierungsmethoden werden die (Java-)Klassen vom Projekt ChaTEAU [Aug22b] verwendet, die die Abhängigkeiten S-t-tgds und Tgds definieren. In Abbildung 5.1 wird ein Überblick über den Aufbau einer Abhängigkeit in ChaTEAU gegeben. Hierbei bestehen die Abhängigkeiten S-t-tgds und Tgds aus zwei Mengen *head* (Kopf) und *body* (Rumpf) von Relationen. Die Relationen werden durch die Klasse *RelationalAtom* repräsentiert und bestehen aus einem Namen *relationName*, einer Menge von Termen *terms* und einem booleschen Wert *isNegated*. Der boolesche Wert (isNegated) ist wahr, wenn eine negierte Relation vorliegt. Des weiterem definiert die Menge der Terme den eigentlichen Inhalt einer Relation bzw. eines *RelationalAtoms*. Dabei wird ein Term als Konstante, Variable oder als Nullwert definiert. Eine Konstante (*Constant*) definiert einen konkreten Wert, bestehend aus einem Namen *name*, einen Konstanten-Typ *constType* und dem konkreten Wert *value*. Für die Definition des Konstanten-Typen (*ConstType*) existieren drei Möglichkeiten. Somit kann eine Konstante einen String, einen Integer oder einen Double-Wert bestimmen. Dagegen besteht eine Variable aus einem Namen *name*, einem Quantor *variableType* und einem Index *index*. Dementsprechend wird für jede Variable ein Quantor (*VariableType*) benötigt, der durch ein Token *token* und einer Beschreibung *describtion* definiert wird. Bei der

Anwendung der Quantoren besteht die Möglichkeit einen Allquantor (*FOR_ALL*) oder eine Existentzquantor (*EXISTS*) zu verwenden. Hierbei wird ein Allquantor bzw. eine allquantifizierte Variable mit dem Token V gekennzeichnet und ein Existenzquantor wird mit dem Token E beschrieben. Im Gegensatz zu einer Variable und einer Konstante wird bei einem Nullwert nicht zwischen unterschiedlichen Typen unterschieden. Diesbezüglich besteht ein Nullwert ausschließlich aus einem Namen und einem Index. Außerdem werden alle Nullwerte in ChaTEAU durch ein N gekennzeichnet.

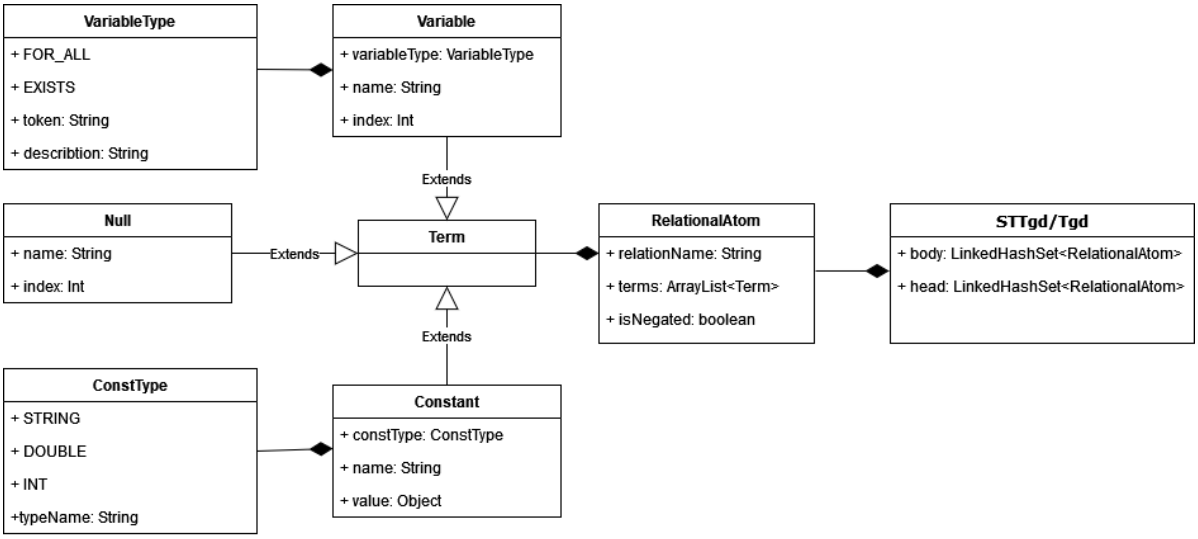

Abbildung 5.1: Aufbau der Abhängigkeiten in ChaTEAU

Beispiel. Für die Veranschaulichung wird im Folgenden eine S-t-tgd als Beispiel, mit den Notationen von ChaTEAU, dargestellt:

$$St(V\_ma\_1, V\_ln\_1, V\_fn\_1) \wedge G(V\_ma\_1, 1.3) \\ \rightarrow Res(V\_ma\_1, V\_ln\_1, E\_mo\_1).$$

In dieser S-t-tgd wird eine Abbildung von den Relationen *Students* (kurz: *St*) und *Grades* (kurz: *G*) auf die Ergebnisrelation *Res* realisiert. Für den Aufbau in ChaTEAU enthält der Rumpf *body* der S-t-tgd, die beiden Relationen bzw. *RelationalAtoms* *St* und *G* und der Kopf *head* enthält die Ergebnisrelation *Res*. Des weiterem werden die einzelnen Relationen aus einer Menge von Termen gebildet, die die einzelnen Variablen bzw. Konstanten definieren. Beispielsweise besteht die Relation *G* aus der Variable *MatricNo* (kurz: *ma*) und einer Konstante *Grade* (kurz: *g*), die einem konkreten Wert von 1.3 entspricht. Außerdem beinhalten alle Variablen eine Quantor und einem Index. Dementsprechend werden alle allquantifizierten Variablen mit einem V und alle existenzquantifizierten mit einem E gekennzeichnet. Folglich ist in der Relation *G* die Variable V_ma_1 eine allquantifizierte Variable mit dem Index 1 und die Variable E_mo_1

(*ModulNo*) ist ein Beispiel für eine exitenzquantifizierte Variable in der Relation *Res*.

In diesem Abschnitt wurde die Implementierung der Abhängigkeiten in ChaTEAU beschrieben. Anschließend werden diese Abhängigkeiten von ChaTEAU verwendet, um die eigene Invertierungsmethode (Algorithmus 5) zu implementieren. Dementsprechend wird im folgenden Abschnitt die Implementierung der einzelnen Schritt des Algorithmus 5 beschrieben.

5.2 Algorithmus Invertierung

Für die Veranschaulichung der Implementierung des Invertierungsalgorithmus (Algorithmus 5) wird im Folgenden eine Betrachtung der einzelnen Schritte des Algorithmus durchgeführt. Hierbei wird die Umsetzung der einzelnen Schritte in der Programmiersprache *Java* aufgeführt.

Eingabe und Ausgabe. Als Eingabe erhält der Invertierungalgorithmus bzw. die Methode *invert* eine Menge von S-t-tgds und eine Menge von Tgds. Dabei werden die Mengen der S-t-tgds und Tgds als *LinkedHashSet* umgesetzt. Dementsprechend nimmt die Methode ein *LinkedHashSet<STTgd>* für die S-t-tgds und ein *LinkedHashSet<Tgd>* für die Ausgabe. Als Ausgabe liefert die Methode die Menge der invertierten S-t-tgds zurück, entsprechend wird ein *LinkedHashSet<STTgd>* zurückgegeben. Die Rückgabe ist auf die Menge der invertierten S-t-tgds beschränkt, da die Menge der invertierten Tgd nach Anwendung des Algorithmus 5, keine Relevanz für die Berechnung der minimalen Teildatenbank besitzt. Folglich enthält die Menge der invertierten S-t-tgds alle Informationen für die Berechnung der minimalen Teildatenbank, da durch die Anwendung der Algorithmus 5 die Informationen/Relationen der (invertierten) Tgds in die invertierten S-t-tgds integriert worden sind.

Erster, Zweiter und Dritter Schritt. Bei der Implementierung des Algorithmus 5 wird der erste, zweite und dritte Schritt zusammengefügt, um eine Vereinfachung der einzelnen Schritte durchzuführen. Im ersten Schritt des Algorithmus werden die S-t-tgds und Tgds in unterschiedliche Mengen unterteilt, um eine Unterscheidung zwischen S-t-tgds und Tgds zu gewährleisten. Des weiterem werden innerhalb dieser Mengen, die einzelne S-t-tgds bzw. Tgds wieder in einzelne Mengen unterteilt. Folglich werden die unterschiedlichen Abhängigkeiten in einer Menge, die wiederum aus den Mengen der einzelnen S-t-tgds bzw. Tgds besteht, gespeichert. Diese Einteilung ist notwendig, um bei der Normalisierung im zweitem Schritt die Zugehörigkeit zu erhalten, welche normalisierten (Teil-)S-t-tgds zusammen zu einer (Gesamt-)S-t-tgd gehören. Diese Zugehörigkeit ist bei der Zusammenfassung der normalisierten S-t-tgds im siebten Schritt notwendig. Diesbezüglich werden für die Umsetzung des ersten Schrittes die Mengen

invertedTgd und *invertedSttgds* angelegt. Diese Mengen werden jeweils durch eine *HashMap<Integer,LinkedHashSet<STTgd>>* bzw. *HashMap<Integer,LinkedHashSet<Tgd>>* repräsentiert. Demnach bestehen die Mengen *invertedTgd* und *invertedSttgds* aus Schlüssel-Werte-Paare, die für einen Schlüssel (Integer) eine Menge von S-t-tgds bzw. Tgds (LinkedHashSet<>) besitzen. Im zweitem Schritt des Algorithmus wird eine Normalisierung der Köpfe der Abhängigkeiten durchgeführt. Entsprechend werden alle Abhängigkeiten der Form $\alpha \rightarrow \beta_1 \wedge ... \wedge \beta_n$ in mehrere Abhängigkeiten $\alpha \rightarrow \beta_1, ..., \alpha \rightarrow \beta_n$ zerlegt. Diese Zerlegung wird durch das Anlegen einer neuen S-t-tgd für jede Relation im Kopf der Abhängigkeit realisiert. Im drittem Schritt wird die eigentliche Invertierung der Abhängigkeiten durchgeführt, indem der Kopf mit dem Rumpf vertauscht wird.
Im Folgenden ist die gemeinsame Durchführung des Einteilung, Normalisierung und Invertierung (Schritt 1, 2 und 3) der S-t-tgds, als Programmiercode in Java veranschaulicht:

```
HashMap<Integer, LinkedHashSet<STTgd>> invertedSttgd = new HashMap<>();
HashMap<Integer, LinkedHashSet<Tgd>> invertedTgd = new HashMap<>();
int key = 0;
for (STTgd sttgdEl : sttgd) {
    LinkedHashSet<STTgd> sttgdList = new LinkedHashSet<>();
      for (RelationalAtom headAtom : sttgdEl.getHead()) {
          LinkedHashSet<RelationalAtom> newHead = new LinkedHashSet<>();
          newHead.add(atom);
          STTgd newSttgd = new STTgd(newHead, sttgdEl.getBody());
          sttgdList.add(newSttgd);
         }
      invertedSttgd.put(key, sttgdList);
      key++;
    }
```

Demnach wird eine einzelne Betrachtung jeder S-t-tgd *sttgdEl* aus der Eingabemenge *sttgd* durchgeführt. Hierbei wird für jede Relation *headAtom* im Kopf der S-t-tgd *sttgdEl* eine neue S-t-tgd *newSttgd* generiert. Diese neue S-t-tgd enthält als Rumpf das eine Kopf-Atom *headAtom* und als Kopf den Rumpf der S-t-tgd *sttgdEL*. Dementsprechend wird durch die neue S-t-tgd *newSttgd* die Normalisierung (Schritt 2) und die Invertierung (Schritt 3) realisiert. Anschließend werden die einzelnen neuen S-t-tgds, die durch die Anwendung der Normalisierung und Invertierung einer S-t-tgd aus der Eingabemenge generiert worden sind, zu einer Menge zusammengefasst (Schritt 1). Schließlich bildet jede neue Mengen einen Wert von den Schlüssel-Wert-Paaren in der HashMap *invertedSttgd*. Die Einteilung, Normalisierung und Invertierung für die Tgds erfolgt analog.
Für die Veranschaulichung wird im Folgenden ein Beispiel für die Einteilung, Normalisierung und eigentliche Invertierung durchgeführt. Herbei werden folgende S-t-tgds, vom Beispiel aus Abschnitt 3, als Eingabemenge betrachtet:

$$St(ma, fn, ln) \wedge Pa(ma, mo) \rightarrow R(ma, ln, mo)$$
$$G(ma, mo, g, se) \rightarrow \exists ln : Se(ma, ln, se) \wedge R(ma, ln, mo).$$

In der ersten S-t-tgd wird eine Abbildung von den Relationen *Students* (kurz: St) und *Participants* (kurz: Pa) auf eine Ergebnisrelation R realisiert. Des weiterem wird in der

zweiten S-t-tgd eine Abbildung der Relation *Grades* (kurz: G) auf die beiden Ergebnisrelationen *Semester* (kurz: Se) und R durchgeführt. Für die Umsetzung der ersten drei Schritte des Algorithmus 5 bzw. des oben dargestellten Programmiercodes wird jede S-t-tgd unabhängig voneinander betrachtet. Die Betrachtung der ersten S-t-tgd liefert folgendes Zwischenergebnis in der HashMap *invertedSttgd*:

$$[0 = \{R(ma, ln, mo) \rightarrow St(ma, fn, ln) \wedge Pa(ma, mo)\}].$$

In der ersten S-t-tgd liefert die Normalisierung des Kopfes eine neue S-t-tgd zurück, da in der ursprünglichen S-t-tgd nur eine Relation im Kopf vorhanden ist. Dementsprechend bewirkt die Normalisierung keine Veränderung der ersten S-t-tgd. Anschließend wird die Invertierung und Einteilung der ersten S-t-tgd durchgeführt. Folglich wird für die eigentliche Invertierung der Rumpf und Kopf der S-t-tgd miteinander vertauscht und eine Einteilung in die HashMap *invertedSttgd* umgesetzt. Demnach besteht das erste Schlüssel-Wert-Paar von *invertedSttgd* aus dem Schlüssel 0 mit der Menge, bestehend aus der einen invertierten S-t-tgd, als Wert. Darauffolgend wird die zweite S-t-tgd betrachtet, wobei im Folgenden das Ergebnis der Einteilung, Normalisierung und Invertierung dargestellt ist:

$$\begin{aligned}[0 = \{R(ma, ln, mo) \rightarrow St(ma, fn, ln) \wedge Pa(ma, mo)\}],\\ [1 = \{R(ma, ln, mo) \rightarrow G(ma, mo, g, se),\\ Se(ma, ln, se) \rightarrow G(ma, mo, g, se)\}].\end{aligned}$$

In der zweiten S-t-tgd ist eine Normalisierung des Kopfes notwendig, da zwei Relationen (Se und R) im Kopf der S-t-tgd vorhanden sind. Entsprechend wird diese S-t-tgd in zwei neue S-t-tgds zerlegt, wobei die Relationen Se und R die einzelnen Köpfe bilden. Anschließend erfolgt die eigentliche Invertierung, indem Kopf und Rumpf miteinander getauscht werden. Diese Invertierung und die Normalisierung bildet die zwei neuen S-t-tgds $R(ma, ln, mo) \rightarrow G(ma, mo, g, se)$ und $Se(ma, ln, se) \rightarrow G(ma, mo, g, se)$. Abschließend werden die zwei neuen S-t-tgds in eine Menge gespeichert und in die HashMap *invertedSttgd* hinzugefügt. Demnach wird ein Schlüssel-Wert-Paar mit dem Schlüssel 1 und die Menge der zwei neuen S-t-tgds als Wert zum Ergebnis hinzugefügt. Darauffolgend ist die Normalisierung, Invertierung und Einteilung (Schritt 1, 2 und 3) der S-t-tgds im Beispiel abgeschlossen.

Vierter Schritt. Im viertem Schritt werden Ersetzungen in den invertierten S-t-tgds und Tgds durchgeführt. Bei diesen Ersetzungen werden zwei Abhängigkeiten der Form $\beta \rightarrow \alpha_1 \wedge \cdots \wedge \alpha_n$ und $\alpha_1 \rightarrow \gamma_1 \wedge \cdots \wedge \gamma_m$, durch eine Abhängigkeit der Form $\beta \rightarrow \gamma_1 \wedge \cdots \wedge \gamma_m \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$ ersetzt. Dementsprechend wird eine Relation α_1 die im Kopf einer Abhängigkeit und im Rumpf einer anderen Abhängigkeit vorkommt ersetzt. Für die Umsetzung dieser Ersetzung muss jede Relation bzw. RelationalAtom im Kopf einer Abhängigkeit mit allem Relationen im Rumpf der anderen Abhängigkeiten verglichen

werden. Hierbei wird im Folgendem die Relation im Kopf als Kopf-Relation und die Relation im Rumpf als Rumpf-Relation bezeichnet. Außerdem wird die Abhängigkeit mit der Kopf-Relation als Kopf-Abhängigkeit und die Abhängigkeit mit der Rumpf-Relation als Rumpf-Abhängigkeit bezeichnet. Bei einer Gleichheit vom Kopf- und Rumpf-Relation wird die entsprechende Ersetzung durchgeführt. Dabei wird die Kopf-Relation in der Kopf-Abhängigkeit durch den Rumpf der Rumpf-Abhängigkeit ersetzt.
Für die Umsetzung ist nur eine Ersetzung innerhalb der S-t-tgds notwendig, da ausschließlich die Menge der invertierten S-t-tgds ausgegeben wird. Demnach werden Ersetzungen innerhalb der Tgds nicht durchgeführt, da diese keinen Einfluss auf die Ausgabe bzw. auf die S-t-tgds besitzen und ausschließlich die invertierten S-t-tgds ausgegeben werden. Diesbezüglich werden bei der Implementierung alle Relationen in den Köpfe der S-t-tgds mit allen Rümpfen der S-t-tgds und Tgds verglichen. Bei einer Gleichheit der Kopf-Relation in einer S-t-tgd (Kopf-Relation) mit einer Rumpf-Relation, in einer Tgd oder S-t-tgd (Rumpf-Relation), wird eine neue S-t-tgd generiert. Diese neue S-t-tgd enthält als Rumpf den Rumpf der Kopf-Abhängigkeit und als Kopf die Vereinigung der Köpfe der Rumpf- und Kopf-Abhängigkeit, ohne die eigentliche Kopf-Relation. Anschließend wird die eigentliche S-t-tgd (Kopf-Abhängigkeit) gelöscht, da diese durch die neue S-t-tgd ersetzt wird. Diese Prinzip mit den Vergleichen der Rumpf- und Kopf-Relationen und den ersetzen innerhalb der S-t-tgds muss solange wiederholt werden, bis keine Veränderungen durch die Ersetzung realisiert wird.
Für die Veranschaulichung dieses Ersetzungsschrittes wird im Folgendem ein Beispiel durchgeführt. Als Beispiel werden folgende Abhängigkeiten betrachtet:

$$\begin{array}{ll} \text{Tgds:} & G(mo, ma, se, g) \rightarrow G'(mo, ma, g), \\ & G'(mo, ma, g) \rightarrow G''(ma, g) \\ \text{S-t-tgd:} & St(ma, ln, fn) \wedge G''(ma, g) \rightarrow Res(ln, fn, g). \end{array}$$

In der ersten Tgd wird die Relation *Grades* (kurz: G) auf ein Zwischenergebnis G' abgebildet, wobei eine Projektion auf die Attribute *ModulNo.*, *MatricNo.* und *Grade* realisiert wird. Anschließend wird die Relation G' auf ein weiteres Zwischenergebnis $G''(ma, g)$ abgebildet, wobei eine Projektion auf die Attribute *MatricNo.* und *Grade* durchgeführt wird. Dieses Zwischenergebnis $G''(ma, g)$ wird in der S-t-tgd zusammen mit der Relation *Students* (kurz: St) auf die Ergebnisrelation *Res* abgebildet. Im Ersetzungsschritt wird das Ziel verfolgt die Ergebnisrelation auf die eigentlichen Quellrelationen abzubilden. Dementsprechend soll im Beispiel eine Abbildung von der Ergebnisrelation *Res* auf die eigentlichen Quellrelationen *Students* und *Grades* durchgeführt werden, ohne die Zwischenergebnisse G' und G'' zu enthalten. Für die Umsetzung des Ersetzungsschrittes müssen zuerst die Normalisierung, Invertierung und Einteilung (Schritt 1, 2 und 3) der Abhängigkeiten erfolgen. Dabei seien folgende Abhängigkeiten bzw. Mengen das Ergebnis dieser Schritte:

$$\begin{aligned}
&\text{invertedTgd:} && [0 = \{G'(mo, ma, g) \rightarrow G(mo, ma, se, g)\}],\\
&&& [1 = \{G''(ma, g) \rightarrow G'(mo, ma, g)\}]\\
&\text{invertedSttgd:} && [0 = \{Res(ln, fn, g) \rightarrow St(ma, ln, fn) \wedge G''(ma, g)\}].
\end{aligned}$$

Bei der Umsetzung des Ersetzungsschrittes wird jede Relation im Kopf der S-t-tgd mit den Relationen im Rumpf der anderen Abhängigkeiten verglichen. Entsprechend wird die Relation St (Kopf-Relation) in der S-t-tgd (Kopf-Abhängigkeit) mit den Relationen im Rumpf (Rumpf-Relationen) der Tgds (Rumpf-Abhängigkeit) verglichen. Hierbei liegt keine Gleichheit der Relationen vorliegt, daher ist keine Ersetzung mit der Relation St möglich. Darauffolgend wird die zweite Relation G'' im Kopf der S-t-tgd betrachtet und mit den Relationen in den Rümpfen der Tgds verglichen. Dabei liegt eine Gleichheit mit der zweiten Tgd vor, da die zweite Tgd die Relation G'' im Rumpf besitzt. Dementsprechend wird eine Ersetzung durchgeführt, wobei eine neue S-t-tgd generiert wird. Diese neue S-t-tgd beinhaltet die Relation Res als Rumpf und die Relationen St und G' als Kopf. Demnach bildeten folgende Mengen das Ergebnis eines Ersetzungsschrittes:

$$\begin{aligned}
&\text{invertedTgd:} && [0 = \{G'(mo, ma, g) \rightarrow G(mo, ma, se, g)\}],\\
&&& [1 = \{G''(ma, g) \rightarrow G'(mo, ma, g)\}]\\
&\text{invertedSttgd:} && [0 = \{Res(ln, fn, g) \rightarrow St(ma, ln, fn) \wedge G'(mo, ma, g)\}].
\end{aligned}$$

Anschließend wird eine Wiederholung dieses Prinzip der Ersetzung durchgeführt, da noch Möglichkeiten für die Ersetzung existieren. Entsprechend werden wieder alle Relationen im Kopf der S-t-tgd mit den Rumpf der Tgds verglichen. Hierbei ist eine Gleichheit mit der Relation G' gegeben, wobei eine Ersetzung mit der S-t-tgd und der ersten Tgd durchgeführt wird. Folglich wird eine neue S-t-tgd generiert, die die Relation Res als Rumpf und die Relationen St und G als Kopf verwendet. Die folgenden Mengen seien das Ergebnis dieses Ersetzungsschrittes:

$$\begin{aligned}
&\text{invertedTgd:} && [0 = \{G'(mo, ma, g) \rightarrow G(mo, ma, se, g)\}],\\
&&& [1 = \{G''(ma, g) \rightarrow G'(mo, ma, g)\}]\\
&\text{invertedSttgd:} && [0 = \{Res(ln, fn, g) \rightarrow St(ma, ln, fn) \wedge G(mo, ma, se, g)\}].
\end{aligned}$$

Schließlich ist die Ersetzung abgeschlossen, da weitere Wiederholungen des Ersetzungsschrittes keine weiteren Veränderungen ergeben. Im Beispiel wurde das Ziel der Ersetzung erreicht, indem die S-t-tgd eine Abbildung der Ergebnisrelation Res auf die eigentlichen Quellrelationen G und S realisiert.

Fünfter Schritt. Im fünftem Schritt des Algorithmus 5 werden die Duplikate aus den Köpfen der Abhängigkeiten entfernt. Folglich sei eine Abhängigkeit der Form $\beta \rightarrow \alpha_1 \wedge \cdots \wedge \alpha_i \wedge \alpha_i \wedge \ldots \alpha_n$ gegeben, die durch eine Abhängigkeit der Form $\beta \rightarrow \alpha_1 \wedge \cdots \wedge \alpha \wedge \ldots \alpha_n$ ersetzt wird. Diese Duplikate in den Köpfen der Abhängigkeiten bzw. S-t-tgds können durch die Anwendung der Ersetzungsschritte (Schritt 4) erzeugt werden.

Allerdings wird bei der Implementierung für die Speicherung der Köpfe der S-t-tgds ein *LinkedHashSet<RelationalAtom>* verwendet. Dieses *LinkedHashSet* bildet die Menge der Relationen/RelationalAtoms im Kopf bzw. Rumpf einer S-t-tgd. Im Allgemeinen sind in einem *LinkedHashSet* keine Duplikate zulässig d.h. das Hinzufügen eines Duplikates führt zu keiner Veränderung der Menge. Folglich bleibt beim Hinzufügen einer Relation, die ein Duplikat einer vorhandenen Relation in der Menge entspricht, nur eine diese Relationen in der Menge erhalten. Dementsprechend wird die Entfernung der Duplikate im Kopf und im Rumpf einer Abhängigkeit durch die Struktur einer S-t-tgd in ChaTEAU und die Verwendung der *LinkedHashSets* realisiert.

Sechster Schritt. Im sechsten Schritt werden alle negierten Relationen aus dem Kopf der S-t-tgds entfernt. Im Allgemeinem werden negierte Relationen ausschließlich als Selektionsbedingugng verwendet. Dementsprechend liefern die negierten Relationen keine wichtigen Informationen für die Berechnung einer minimalen Teildatenbank. Folglich können diese entfernt werden, ohne Einfluss auf die minimale Teildatenbank zu nehmen. Ebenfalls kann bei Anwendung der Ersetzungsschritte (Schritt 4) unnötige negierte Relationen im Kopf einer S-t-tgd auftreten, die auch entfernt werden müssen.
Für die Umsetzung des sechsten Schrittes werden alle Relationen in den Köpfen der S-t-tgds betrachtet. Bei dieser Betrachtung wird der boolesche Wert *isNegated* für jede Relation geprüft. Dabei gibt der boolesche Wert *isNegated* an, ob eine Relation negiert ist. Entsprechend wird eine Relation entfernt, wenn der Wert *isNegated* wahr ist. Demnach werden mit dieser Methodik alle negierten Relationen entfernt.
Für die Veranschaulichung der Entfernung der negierten Relationen wird folgende S-t-tgd als Beispiel verwendet:

$$St(ma, ln, fn) \wedge \neg G(mo, ma, se, g) \rightarrow Res(ma, ln, fn)$$

In diesem Beispiel wird eine Abbildung von den Relationen *Students* und *Grades* auf die Ergebnisrelation *Res* realisiert. Allerdings ist die Relation *Grades* negiert, wobei diese Relation als Selektionsbendingung dient. Folglich werden durch diese Bedingung alle Studenten, dessen Matrikelnummer (kurz: *ma*) nicht in der Relation *Grades* vorhanden ist, in die Ergebnisrelation *Res* mitaufgenommen. Entsprechend enthält die Ergebnisrelation *Res* alle Studenten, die noch keine Note erhalten haben. Für die Umsetzung des sechsten Schrittes müssen zuerst die Umsetzung der vorherigen Schritte erfolgen. Die folgende invertierte S-t-tgd bzw. die Menge, sei das Ergebnis der vorherigen Schritte:

$$\text{invertedSttgd:} \quad [0 = \{Res(ma, ln, fn) \rightarrow St(ma, ln, fn) \wedge \neg G(mo, ma, se, g)\}].$$

Anschließend werden bei der Umsetzung der Entfernung der negierten Relationen, die Relationen (St und G) im Kopf der S-t-tgd betrachtet. Der boolesche Wert *isNegated* ist im Fall der Relation St falsch, entsprechend wird diese Relation nicht entfernt. Dagegen ist der boolesche Wert *isNegated* bei der Relation G wahr. Dementsprechend wird die

Relation G aus dem Kopf der S-t-tgd entfernt. Darauffolgend ist die Durchführung des sechsten Schrittes abgeschlossen, die folgende S-t-tgd sei das Ergebnis dieses Schrittes:

$$\text{invertedSttgd:} \quad [0 = \{Res(ma, ln, fn) \rightarrow St(ma, ln, fn)\}].$$

Siebter Schritt. Im siebten Schritt werden die normalisierten S-t-tgds wieder zusammengefügt. Bei dieser Zusammenführung seien zwei normalisierte S-t-tgds der Form $\beta_1 \rightarrow \alpha$ und $\beta_2 \rightarrow \alpha$ in einer Menge gegeben. Diese zwei S-t-tgds mit dem gleichen Kopf werden zu einer S-t-tgd der Form $\beta_1 \wedge \beta_2 \rightarrow \alpha$ zusammengefügt. Aufgrund dessen, dass nur die invertierten S-t-tgds durch den Algorithmus 5 ausgegeben werden, ist eine Zusammenfassung der Tgds nicht notwendig.
Bei der Umsetzung des siebten Schrittes werden ausschließlich S-t-tgds zusammengefügt, die in einem Schlüssel-Wert-Paar innerhalb der HashMap *invertedSttgd* liegen. Hierbei werden die Köpfe der jeweiligen normalisierten S-t-tgds verglichen und bei Gleichheit werden die S-t-tgds zusammengefasst. Der Rumpf dieser zusammengefasste S-t-tgd besteht aus der Konjunktion der beiden Rümpfe der normalisierten S-t-tgd. Des weiterem besteht der Kopf der zusammengefassten S-t-tgd aus einem Kopf der normalisierten S-t-tgds.
Für die Veranschaulichung wird folgende Menge von invertierten normalisierten S-t-tgds als Beispiel verwendet:

$$\begin{aligned}\text{invertedSttgd:} \quad [0 = \{&R(ma, ln, fn) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\},\\ &\{R'(ln, fn) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\}]\\ [1 = \{&R(ma, g) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\},\\ &\{R'(ln, fn, ma) \rightarrow St(ma, ln, fn)\} \wedge G(ma, g)].\end{aligned}$$

Bei der Anwendung des siebten Schrittes werden die einzelnen Schlüssel-Wert-Paare betrachtet und innerhalb der entsprechenden Werte die S-t-tgds zusammengefügt. Entsprechend wird zuerst das Schlüssel-Wert-Paar mit dem Schlüssel 0 betrachtet. Hierbei sind zwei S-t-tgds im Wert enthalten. Diese zwei S-t-tgds werden zu einer neuen S-t-tgd zusammengefügt, da die beiden S-t-tgds den gleichen Kopf besitzen. Im Folgenden ist das (Zwischen-)Ergebnis dieser Schrittes dargestellt:

$$\begin{aligned}\text{invertedSttgd:} \quad [0 = \{&R(ma, ln, fn) \wedge R'(ln, fn) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\}],\\ [1 = \{&R(ma, g) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\},\\ &\{R'(ln, fn, ma) \rightarrow St(ma, ln, fn)\} \wedge G(ma, g)].\end{aligned}$$

Anschließend wird das zweite Schlüssel-Wert-Paar mit dem Schlüssel 1 betrachtet. Diese Schlüssel-Wert-Paar enthält ebenfalls zwei normalisierte S-t-tgds, die zu einer neuen S-t-tgd zusammengefügt werden. Das Ergebnis dieses Schrittes ist im Folgendem veranschaulicht:

invertedSttgd: $[0 = \{R(ma, ln, fn) \wedge R'(ln, fn) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\}]$,
$[1 = \{R(ma, g) \wedge R'(ln, fn, ma) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\}]$.

Schließlich ist die Durchführung des siebten Schrittes abgeschlossen, da keine weiteren Schlüssel-Wert-Paare im Beispiel vorhanden sind und keine weitere Möglichkeit zum Zusammenfassen der S-t-tgds besteht.

Achter Schritt. Im achtem Schritt des Algorithmus werden alle Gleichheitsprädikate der Form $x = c$ angewendet. Dementsprechend wird bei einer S-t-tgd der Form $\beta \rightarrow \alpha \wedge x = c$ die Variable x durch die Konstante c ersetzt. Die Implementierung dieser Schrittes ist mit der aktuellen Version von ChaTEAU nicht notwendig, da Konstanten als Term in einer Relation dargestellt werden. Folglich werden keine Gleichheitsprädikate der Form $x = c$ beschrieben, sondern die Relationen bzw. *RelationalAtoms* enthalten immer die Konstanten als Term. Dementsprechend ist die Umsetzung dieses Schrittes automatisch in ChaTEAU gegeben.

Neunter Schritt. Im neunten Schritt des Algorithmus 5 werden die Quantoren der einzelnen Variablen angepasst. Hierbei werden alle Variablen existenzquantifiziert die im Kopf einer S-t-tgd auftreten, aber nicht im Rumpf der S-t-tgd vorkommen. Die restlichen Variablen werden allquantifiziert.
Für die Umsetzung der Anpassung der Quantoren muss für jede Relation einer S-t-tgd, die Terme angepasst werden. Dabei wird zuerst von den S-t-tgds bis auf die einzelnen Terme iteriert, mithilfe von Schleifen. Anschließend wird jeder einzelne Term betrachtet. Bei dieser Betrachtung wird zuerst mit der Methode *isVariable* geprüft, ob der Term eine Variable ist. Diese Überprüfung wird durchgeführt, da nur Variablen einen Quantor besitzen. Entsprechend ist bei einer Konstante oder einen Nullwert keine Anpassung der Quantoren möglich. Darauffolgend wird zwischen Terme im Kopf und im Rumpf einer S-t-tgd unterschieden. Aufgrund dessen, dass S-t-tgds definitionsgemäß keine existenzquantifizierte Variablen im Rumpf besitzen, werden alle Terme/Variablen im Rumpf einer S-t-tgd allquantifiziert. Hierbei wird für jeden Term im Rumpf ein neuer Term angelegt, wobei der neue Term den Variablen-Typ *FOR_ALL* enthält, ansonsten wird der Namen und der Index des alten Terms übernommen. Bei der Betrachtung der Terme im Kopf einer S-t-tgd wird ein Vergleich, zwischen den aktuellen betrachteten Term im Kopf und allen Termen im Rumpf einer S-t-tgd, durchgeführt. Dementsprechend wird ein Term im Kopf einer S-t-tgd allquantifiziert, wenn beim Vergleich der Terme eine Gleichheit vorliegt. Dagegen wird ein Term existenzquantifiziert, wenn keine Gleichheit zwischen den Term im Kopf und einem Term im Rumpf der S-t-tgd vorliegt. Die Anpassung der Quantoren erfolgt wieder durch das Anlegen eines neuen Terms, der den entsprechenden Quantor als Variablen-Typ enthält.

Als Beispiel für die Anpassung der Quantoren, sei folgende S-t-tgd gegeben:

$$St(V\_ma\_1, V\_ln\_1, V\_fn\_1) \wedge G(V\_ma\_1, V\_g\_1) \\ \rightarrow Res(V\_ln\_1, V\_fn\_1, V\_g\_1, E\_mo\_1).$$

In dieser S-t-tgd wird eine Abbildung von den Relationen *Students* und *Grades* auf die Ergebnisrelation *Res* realisiert. Für die Darstellung der Quantoren wurden in der S-t-tgd die Notationen von ChaTEAU übernommen. Dementsprechend erhalten alle Variablen zusätzlich einen Quantor und einen Index. Dabei erhalten alle allquantifizierte Variablen ein V vor den Namen und alle existenzquantifizierte Variablen ein E. Beispielsweise ist die Variable *ModulNo* (kurz: *mo*) in der Relation *Res* existenzquantifiziert und entsprechend mit einem E gekennzeichnet. Für die Durchführung der Anpassung der Quantoren müssen zuerst die vorherigen Schritte erfolgen, wobei die folgende Menge das Ergebnis der vorherigen Schritte repräsentiert:

$$\text{invertedSttgd:} \quad [0 = \{Res(V\_ln\_1, V\_fn\_1, V\_g\_1, E\_mo\_1) \rightarrow \\ St(V\_ma\_1, V\_ln\_1, V\_fn\_1) \wedge G(V\_ma\_1, V\_g\_1)\}].$$

Bei der Durchführung der Anpassung der Quantoren werden zuerst alle Terme/Variablen im Rumpf einer S-t-tgd allquantifiziert. Entsprechend wird die Variable *ModulNo* (kurz: *mo*) angepasst, indem ein neuer Term mit dem Variablen-Typ *FOR_ALL*, mit dem Namen *ModulNo* (bzw. *mo*) und den Index 1 angelegt wird. Anschließend wird der alte Term E_mo_1 durch den neuen Term V_mo_1 ersetzt. Bei der Anpassung der Terme im Kopf der S-t-tgd wird der Term *MatricNo* (kurz: *ma*) existenzquantifiziert, da kein Term im Rumpf mit dem gleichen Namen existiert. Ebenfalls wird ein neuer Term angelegt mit dem Variable-Typ *EXISTS*, den Namen *MatricNo* (bzw. *ma*) und dem Index 1. Dieser neue Term V_ma_1 ersetzt wieder den alten Term E_ma_1. Die restlichen Variablen bleiben allquantifiziert. Darauffolgend ist die Durchführung des neunten Schrittes bzw. der Anpassung der Quantoren abgeschlossen, die folgende Menge sei das Ergebnis dieses Schrittes:

$$\text{invertedSttgd:} \quad [0 = \{Res(V\_ln\_1, V\_fn\_1, V\_g\_1, V\_mo\_1) \rightarrow \\ St(E\_ma\_1, V\_ln\_1, V\_fn\_1) \wedge G(V\_ma\_1, V\_g\_1)\}].$$

Beispiel. Für den Vergleich des Konzeptes und der Implementierung wird im Folgenden das gleiche Beispiel, wie in Abschnitt 4.2.3 durchgeführt. Infolgedessen wird folgende Menge von (Teil-)S-t-tgds als Eingabe, für die Methode *invert* bzw. für den Algorithmus 5 betrachtet:

$$St(ma, ln, fn) \wedge G(ma, g) \rightarrow R(ma, fn, ln, g) \\ R(ma, fn, ln, 1.3) \rightarrow R'(ma, fn, ln, 1.3) \\ R'(ma, fn, ln, 1.3) \rightarrow R''(ln, fn)$$

In den ersten drei Schritte des Algorithmus 5 erfolgt eine Einteilung, Normalisierung und Invertierung der S-t-tgds, wobei folgende Menge bzw. *HashMap* das Ergebnis dieser Schritte repräsentiert:

$$\begin{aligned}\text{invertedSttgd:}\quad & [0 = \{R(ma, fn, ln, g) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\}],\\ & [1 = \{R'(ma, fn, ln, 1.3) \rightarrow R(ma, fn, ln, 1.3)\}],\\ & [2 = \{R''(ln, fn) \rightarrow R'(ma, fn, ln, 1.3)\}].\end{aligned}$$

Anschließend erfolgt der vierte Schritt, indem mehrere Ersetzungsschritte durchgeführt werden, die eine Zusammenfassung der (Teil-)S-t-tgd realisieren. Hierbei werden die einzelnen Relationen im Kopf einer S-t-tgd mit alle Relationen im Rumpf der S-t-tgds verglichen. Bei einer Gleichheit der beiden Relationen wird ein Ersetzungsschritt durchgeführt, der eine neue S-t-tgd generiert. Beispielsweise kann im Beispiel ein Ersetzungsschritt zwischen der ersten S-t-tgd (Schlüssel 0) und der zweiten S-t-tgd (Schlüssel 1) durchgeführt werden, da beide die Relation R beinhalten. Des weiterem kann ein Ersetzungsschritt zwischen der zweitem S-t-tgd (Schlüssel 1) und der dritten S-t-tgd (Schlüssel 2) erfolgen, wobei die Relation R' ersetzt wird. Im Folgenden wird die Ergebnismenge der Umsetzung des vierten Schrittes dargestellt:

$$\text{invertedSttgd:}\quad [0 = \{R''(ln, fn) \rightarrow St(ma, ln, fn) \wedge G(ma, g)\}].$$

Dementsprechend wurden die drei (Teil-)S-t-tgd, durch Anwendung des viertem Schrittes, zu einer S-t-tgd zusammengefasst. Diese invertierte S-t-tgd realisiert eine Abbildung der Ergebnisrelation R'' auf die eigentlichen Quellrelationen St und G. Als nächsten Schritt in der Implementierung würde die Entfernung der negierten Relationen (Schritt 6) und die Zusammenfassung der normalisierten S-t-tgds (Schritt 7) erfolgen. Allerdings führen diese Schritte zu keiner Veränderung der obigen S-t-tgd, da die S-t-tgd keine negierten Relationen enthält und nicht weiter zusammengefasst werden kann. Entsprechend erfolgt der letzte Schritt, der die Anpassung der Quantoren durchführt. Dabei werden alle Terme im Rumpf der S-t-tgd allquantifiziert. Ebenfalls wird ein Vergleich zwischen den Termen im Kopf und den Termen in Rumpf durchgeführt. Bei Gleichheit wird der entsprechende Term im Kopf allquantifiziert. Dagegen sollte keine Gleichheit vorliegen wird der Term im Kopf existenzquantifiziert. Im Folgenden ist die Ergebnismenge des letzten Schrittes veranschaulicht, wobei die Notationen von ChaTEAU verwendet wurden, um die Quantoren darzustellen:

$$\begin{aligned}\text{invertedSttgd:}\quad & [0 = \{R''(V\_ln\_1, V\_fn\_1) \rightarrow\\ & St(E\_ma\_1, V\_ln\_1, V\_fn\_1) \wedge G(E\_ma\_1, E\_g\_1)\}].\end{aligned}$$

Schließlich bildet diese Menge das Endergebnis der Durchführung der Methode *invert* bzw. des Algorithmus 5.

In diesem Abschnitt wurden die Umsetzung bzw. Implementierung des Algorithmus 5 beschrieben. Im folgenden Abschnitt wird die Implementierung des Algorithmus 7 betrachtet. Dieser Algorithmus 7 dient für die Invertierung der Aggregatfunktion, wobei die verwendete Anzahl von Tupel in der minimalen Teildatenbank erhalten bleibt. Hierbei wird die Umsetzung des Algorithmus 5 verwendet, um zuerst die Invertierung der Aggregatfunktionen ohne die Rekonstruktion der verwendeten Anzahl an Tupel durchzuführen. Anschließend wird im Algorithmus 7 das Ergebnis des Algorithmus 5, um weitere Abhängigkeit erweitert, die die Anzahl der verwendeten Tupel wiederherstellen.

5.3 Algorithmus Why-Provenance

In diesem Abschnitt wird die Umsetzung der einzelnen Schritte des Algorithmus 7 beschrieben. Der Algorithmus 7 dient für die Invertierung der Aggregatfunktionen, wobei die Anzahl der verwendeten Tupel bei der Invertierung erhalten bleiben soll. Entsprechend wird durch die Anwendung von Side-Tables die verwendete Tupelanzahl, der Tupel die an der Berechnung der Aggregatfunktion beteiligt waren, rekonstruiert. Die Implementierung dieses Algorithmus ist in der Methode *invertAgg*, in der (Java-)Klasse *InvertingTechniques*, umgesetzt. Für die Darstellung und Umsetzung der Abhängigkeiten wurden ebenfalls die ChaTEAU-Klassen (Abschnitt 5.1) verwendet. Im Folgenden ist die Umsetzung der einzelnen Schritte des Algorithmus 7 beschrieben:

Eingabe und Ausgabe. Als Eingabe erhält die Methode *invertAgg* eine Menge von S-t-tgds, eine Menge von Tgd und eine Side-Table. Die Side-Table entspricht einer Relation bzw. *RelationalAtom* die als Variable die Tupel-ID enthält. Hierbei enthält die Side-Table alle Tupel-IDs, der Tupel die an der Berechnung der Aggregatfunktion beteiligt waren und dient somit für die Rekonstruktion der verwendeten Tupelanzahl. Als Ausgabe liefert die Methode eine Menge von S-t-tgds, in Form eines *LinkedHashSet*, und eine Tgd zurück. Diese Ausgabe wird in einem Paar der Form *Pair<LinkedHashSet<STTgd>, Tgd>* zusammengefasst.

Erster Schritt. Im ersten Schritt des Algorithmus 7 wird die Invertierung ohne zusätzliche Side-Tables und ohne die Rekonstruktion der verwendeten Tupelanzahl durchgeführt. Dieser Invertierung wird durch die Anwendung des Algorithmus 5 realisiert. Entsprechend wird bei der Umsetzung dieses Schrittes die Methode *invert* aufgerufen, die die Implementierung des Algorithmus 5 (Abschnitt 5.2) entspricht.

Zweiter Schritt. Im zweitem Schritt wird die Tgd für die Rekonstruktion der Tupelanzahl generiert. Hierbei wird für eine S-t-tgd der Form $\beta \rightarrow alpha$ eine Tgd der Form $S(t) \wedge \beta \wedge \neg S'(t) \rightarrow \alpha' \wedge S'(t)$ hinzugefügt. Dabei sei die Relation S die Side-Table aus der Eingabe, die die Tupel-IDs t enthält.
Für die Umsetzung dieses Schrittes werden zwei neue Side-Tables S' und $\neg S'$ definiert,

die die Tupel-ID t als Variable enthalten. Ebenfalls wird für jede Relation α im Kopf der betrachteten invertierten S-t-tgd eine neue Relation α' angelegt. Anschließend wird die Tgd definiert, indem der Rumpf eine Konjunktion der Side-Tables S und $\neg S'$ mit dem Rumpf der invertierten S-t-tgd β bildet. Dagegen wird im Kopf der Tgd eine Konjunktion der Side-Table S' mit den neuen Relationen α' durchgeführt.
Für die Veranschaulichung der Konstruktion der Tgd, wird folgende invertierte S-t-tgd als Beispiel betrachtet:

$$Res(V\_d\_1) \rightarrow G(V\_id_g\_1, 002, E\_ma\_1, E\_se\_1, E\_g\_1).$$

Diese invertierte S-t-tgd wird als Ausgabe des ersten Schrittes angenommen, wobei diese S-t-tgd die Invertierung der Berechnung des Notendurchschnitts repräsentiert. Hierbei wurden die Notationen von ChaTEAU verwendet, um die Quantoren der Variablen darzustellen. Dementsprechend sind alle allquantifizierten Variablen mit einem V gekennzeichnet und alle existenzquantifizierten Variablen mit einem E. Bei der Durchführung des zweiten Schrittes werden zwei neue Side-Table S' und $\neg S'$, die als Variable die Tupel-ID id_g enthalten. Ebenfalls wird eine neue Relation G' definiert, die eine Kopie von der Relation *Grades* (kurz: G) entspricht, und als Zwischenergebnis in der Tgd dient. Anschließend wird folgende Tgd gebildet, die das Ergebnis dieser Schrittes bildet:

$$S'(V\_id_g\_1) \land Res(V\_d\_1) \land \neg S'(V\_id_g\_1)$$
$$\rightarrow G'(V\_id_g\_1, 002, E\_ma\_1, E\_se\_1, E\_g\_1) \land S'(V\_id_g\_1).$$

Dritter Schritt. Im drittem Schritt wird für die Tgd der Form $S(t) \land \beta \land \neg S'(t) \rightarrow \alpha' \land S'(t)$ eine S-t-tgd der Form $\alpha' \rightarrow \alpha$ generiert. Dementsprechend wird bei der Umsetzung eine neue S-t-tgd erzeugt, die alle neuen Relationen α', die die Zwischenergebnisse der Tgd bilden, auf die Relationen des Kopfes der invertierten S-t-tgd abbildet. Als Beispiel wird die folgende Tgd aus dem vorherigen Schritt betrachtet:

$$S'(V\_id_g\_1) \land Res(V\_d\_1) \land \neg S'(V\_id_g\_1)$$
$$\rightarrow G'(V\_id_g\_1, 002, E\_ma\_1, E\_se\_1, E\_g\_1) \land S'(V\_id_g\_1).$$

Infolgedessen wird eine neue S-t-tgd hinzugefügt die eine Abbildung vom Zwischenergebnis G' auf die eigentliche Ergebnisrelation der Invertierung G abbildet. Entsprechend wird folgendes Ergebnis durch die Anwendung des dritten Schrittes generiert:

$$\begin{aligned}
\text{Tgd:} \quad & S'(V_id_g_1) \land Res(V_d_1) \land \neg S'(V_id_g_1) \\
& \rightarrow G'(V_id_g_1, 002, E_ma_1, E_se_1, E_g_1) \land S'(V_id_g_1) \\
\text{S-t-tgd:} \quad & G'(V_id_g_1, 002, V_ma_1, V_se_1, V_g_1) \rightarrow \\
& G(V_id_g_1, 002, V_ma_1, V_se_1, V_g_1).
\end{aligned}$$

In diesem Kapitel wurde Implementierung der eigenen Invertierungsmethoden beschrieben. Im folgenden Kapitel wird eine kurze Zusammenfassung mit den Ergebnissen, offene Problemstellung und mögliche Erweiterungen dieser Arbeit gegeben.

6 Zusammenfassung

Im Rahmen dieser Arbeit *Inverse Anfragen in ProSA* wurde die Konvertierung und Invertierung von typischen SQL-Anfragen betrachtet. Außerdem wurde ein eigener Algorithmus für die Invertierung eine Menge von Abhängigkeiten, in Form von S-t-tgds und Tgds, entwickelt. In diesem Kapitel wird eine kurze Zusammenfassung (Abschnitt 6.1) und ein Ausblick (Abschnitt 6.2) für mögliche Erweiterungen der Arbeit und offene Fragestellungen gegeben.

6.1 Fazit

Die Ziele dieser Arbeit waren die Überprüfung der Korrektheit und Vollständigkeit der Darstellung der SQL-Anfragen als Menge von Abhängigkeiten und die Entwicklung einer Methode für die Invertierung in ProSA. Im Rahmen des Projekts ProSA [AH19, AH18b] wird die Konvertierung und Invertierung von SQL-Anfragen benötigt, um eine minimale Teildatenbank zu berechnen. Dabei dient die Berechnung der minimalen Teildatenbank, dazu die Rekonstruktion eines Anfrageergebnisses zu gewährleisten.

Für die Betrachtung der Konvertierung von SQL-Anfragen wird eine Auflistung der Darstellung von SQL-Anfragen als Menge von Abhängigkeiten durchgeführt. Hierbei werden unterschiedliche SQL-Anfragen und dessen Umsetzung als Menge von S-t-tgds und Tgds aufgeführt (Abschnitt 4.1). Eine Besonderheit hierbei bilden die Aggregatfunktionen und der ALL-Operator bei denen eine Fallunterscheidung und eine Anwendung von mehreren Tgds benötigt wird. Des weiterem wird eine Methode für die Konvertierung von verschachtelten Anfragen entwickelt. Dabei werden die Unteranfragen der verschachtelten Anfragen als Tgds umgesetzt, wobei von Innen nach Außen konvertiert wird. Entsprechend wird zuerst die innerste Unteranfrage als Tgd umgesetzt. Anschließend wird das Zwischenergebnis einer Unteranfragen verwendet, um die Tgd der nächsten Unteranfrage zu bilden. Abschließend wird das Zwischenergebnis der letzten Unteranfrage in der S-t-tgd, die die eigentliche Anfrage repräsentiert, verwendet, um das eigentliche Ergebnis zu realisieren. Ebenfalls wird eine Überprüfung der Vollständigkeit der aufgelisteten SQL-Anfragen und dessen Darstellung als Menge von Abhängigkeiten durchgeführt. Für die Überprüfung der Vollständigkeit wird die Dokumentation eines aktuellen Datenbankmanagementsystems (*PostgreSQL*) verwendet. Infolgedessen werden alle SQL-Anfragen bzw. Operationen betrachtet die von diesem Datenbanksystem umgesetzt werden. Dementsprechend wurde die Vollständigkeit überprüft, indem alle

Operationen und dessen Darstellung als Menge von Abhängigkeiten betrachtet wurden, die für ein aktuelles Datenbankmanagementsystems benötigt werden. Allerdings ist die Konvertierung in eine Menge von Abhängigkeiten nicht bei allen Anfragen und Operationen möglich. Beispielsweise ist die Umsetzung der Gruppierung als Menge von S-t-tgds und Tgds nicht möglich, da diese Abhängigkeiten mengenbasiert sind. Hierbei ist die Zusammenfassung der Attributwerte, durch die Gruppierung, nicht mengenbasiert umsetzbar. Die einzige Möglichkeit die Gruppierung durch S-t-tgds und Tgds umzusetzen, ist die Hinzunahme von zusätzlichen Informationen in Form von Side-Tables. Diese Side-Tables müssten die Zusammenfassung der Attributwerte enthalten, um die Gruppierung umzusetzen.

Für die automatische Invertierung einer Menge von Abhängigkeiten, in Form von S-t-tgds und Tgds, wurde im Rahmen dieser Arbeit ein Invertierungsalgorithmus entwickelt und implementiert. Hierbei wird ein Algorithmus für die Invertierung (Abschnitt 4.2.3) entwickelt der eine Menge von Abhängigkeiten (S-t-tgds und Tgds) als Eingabe verwendet und als Ausgabe eine Menge von invertierten S-t-tgds zurückliefert. Für die Entwicklung des Invertierungsalgorithmus werden zwei Ansätze bzw. Algorithmen für die Invertierung von Abhängigkeiten miteinander verglichen. Anschließend wird einer dieser Ansätze (*DirectAdaptedMaxExtendedRecovery*) als Vorlage für den eigenen Invertierungsalgorithmus verwendet und entsprechend erweitert. Der DirectAdaptedMaxExtendedRecovery-Algorithmus nimmt als Eingabe eine Schemaabbildung, mit einer Menge von S-t-tgds, und liefert eine invertierte Schemaabbildung zurück, wobei die invertierten S-t-tgds in Form von Disjunktiven-Mengen ausgegeben werden. Entsprechend wird bei der Erweiterung dieses Ansatzes das Ziel verfolgt die Anwendung der Tgds in der Eingabe zu realisieren und die invertierten S-t-tgds nicht als Disjunktiven-Mengen auszugeben. Diese Erweiterung mit den Tgds in der Eingabe wird benötigt, da die Umsetzung einiger SQL-Anfragen Tgds benötigten. Beispielsweise werden Tgds bei der Umsetzung von verschachtelten Anfragen oder für Aggregatfunktionen verwendet. Hierbei dienen die Tgds um eine Veränderung des eigentlichen Quellschemas durchzuführen und ein Zwischenergebnis zu generieren. Anschließend wird dieses Zwischenergebnis in einer S-t-tgd verwendet, um das eigentliche Ergebnis der Anfrage zu bestimmen. Bei der Invertierung dieser S-t-tgds würde eine Abbildung vom Ergebnis auf die Zwischenergebnisse realisiert werden. Allerdings sind diese Zwischenergebnisse für die Berechnung der minimalen Teildatenbank nicht relevant, da diese im Quellschema nicht vorhanden sind. Entsprechend wird im Invertierungsalgorithmus eine Zusammenfassung der Tgds mit den S-t-tgds durchgeführt, wobei eine Abbildung vom Ergebnis auf die eigentlichen Quellrelationen realisiert wird.

6.2 Ausblick

In diesem Abschnitt wird ein kurzer Ausblick gegeben, wobei mögliche Erweiterungen und offene Fragestellungen beschrieben werden. Hierbei wird im Folgenden eine Unterscheidung zwischen Erweiterungen der in dieser Arbeit beschrieben Methode, der Anwendung von anderen Methodiken und die Betrachtung von anderen Themenbereichen für die Invertierung durchgeführt.

Erweiterung dieser Arbeit. Für die Erweiterung der vorliegen Arbeit besteht die Möglichkeit weitere Operationen bzw. SQL-Anfragen und dessen Konvertierung und Invertierung zu betrachten. Insbesondere sind bei dieser Betrachtung die Operationen relevant, deren Umsetzung in einer Menge von Abhängigkeiten nicht realisiert werden konnte. Beispielsweise ist eine Umsetzung der Gruppierung relevant, da in dieser Arbeit keine Möglichkeit für die Umsetzung dieser Operation gefunden worden ist. Allgemein ist die Umsetzung der Zusammenfassung von Attributwerte, die beispielsweise in der Gruppierung oder in den Window-Funktionen benötigt wird, eine offene Fragestellung in dieser Thematik.
Eine weitere Erweiterung der Arbeit wäre die Bestimmung des CHASE-Inversen-Types [FKPT11b, AH18c] bei der Invertierung. Entsprechend würde nach der Invertierung der konkrete CHASE-Inversen-Typ (Abschnitt 2.2.3), anhand der invertierten Menge der Abhängigkeiten, bestimmt werden.
Für die Verbesserung der Berechnung der minimalen Teildatenbank besteht die Möglichkeit bei der Invertierung zusätzliche Informationen zu verwenden. Diese zusätzliche Informationen verringern den Informationsverlust, der bei einer Invertierung eine Menge von Abhängigkeiten auftreten kann. Diesbezüglich können Provenance-Informationen, beispielsweise in Form von Provenance-Polynomen, bei der Invertierung verwendet werden, um eine genauere minimale Teildatenbank zu berechnen.
Eine weitere offene Problemstellung wird im Abschnitt 4.3 erläutert. Hierbei wurde das Problem entdeckt, dass bei der Berechnung der minimalen Teildatenbank in ProSA Duplikate auftreten können. Diese Duplikate resultieren daraus, dass die hinzugefügten Tupel-IDs nicht als eindeutige (Schlüssel-)Attribute erkannt werden. Eine mögliche Lösung dieses Problems ist in [Han22] aufgeführt.

Andere Invertierungsmethoden. Für die Invertierung einer Menge von Abhängigkeiten bzw. einer SQL-Anfrage existieren mehrere Möglichkeiten. In Rahmen dieser Arbeit wurden ein Vergleich zwischen der Methode eines KSWS-Projektes [WWO+21] und des DirectAdaptedMaxExtendedRecovery-Algorithmus [Zim21] durchgeführt. Hierbei dienen beide Methoden für die Invertierung einer Menge von Abhängigkeiten bzw. einer SQL-Anfrage. In dieser Arbeit wurde eine eigene Invertierungsmethode, basiert auf den DirectAdaptedMaxExtendedRecovery-Algorithmus, entwickelt. Dementsprechend wäre ein weiterer Ansatz für die Invertierung in ProSA, die Umsetzung und Erweiterung der Methode des KSWS-Projektes. Diese Methode würde direkt im SQL-Parser (Abschnitt

3.3.2) umgesetzt werden und die Funktionalität des SQL-Parser übernehmen. Dabei wird in dieser Methode des KSWS-Projekts eine Verschachtelung der SQL-Anfrage durchgeführt, um die SQL-Anfrage in einfachere Teilanfragen zu zerlegen. Anschließend werden die Teilanfragen einzeln Konvertiert und Invertierung und abschließend wieder zusammengefasst. Entsprechend wird bei dieser Methode ein Invertierungsalgorithmus realisiert der eine SQL-Anfrage als Eingabe verwendet und eine Menge von invertierte S-t-tgd ausgibt. Entsprechend müsste die KSWS-Methode um die Anwendung von Tgds, beispielsweise für die Umsetzung der Aggregatfunktion, erweitert werden.
Eine weitere Invertierungsmethode wäre die direkte Invertierung der SQL-Anfrage, wie in [WWO+21] angedeutet. Infolgedessen würde zu jeder SQL-Anfrage eine invertierte SQL-Anfrage gebildet werden. Anschließend würde die invertierte SQL-Anfrage in eine Menge von Abhängigkeiten, durch den SQL-Parser, konvertiert werden. Dementsprechend wird keine Invertierung einer Menge von Abhängigkeiten vorgenommen, sondern die SQL-Anfrage wird in eine invertierte SQL-Anfrage transformiert. Allerdings ist diese Methode der direkten Invertierung der Anfrage am schwersten in der Umsetzung. Ansätze für diese Methode sind in [TNRH19] zu finden.

Anderer Themenbereich. In dieser Arbeit wurde die Invertierung von Anfragen untersucht, die ein Anfrageergebnis liefern. Eine andere Anwendung der Invertierung könnte im Bereich der Schemaevolution durchgeführt werden. Im Bereich der Schemaevoltution werden Operationen verwendet die eine Veränderung des aktuellen Datenbankschemas realisieren. Dabei ist eine Schemaevoltionsoperation eine Funktion die als Eingabe ein relationales Schema und eine Datenbank erhält, und ein verändertes Schema und eine veränderte Datenbank als Ausgabe liefert [CMZ08]. Entsprechend werden bei diesen Operationen keine klassischen Anfrageergebnisse gebildet, sondern eine Veränderung der Datenbank durchgeführt. Ein Beispiel für eine Schemaevolutionsoperation ist die Operation CREATE TABLE, die eine neue Tabelle in der Datenbank anlegt. Folglich kann die Invertierung der Schemaevolutionsoperationen benutzt werden, um eine Rekonstruktion eines Datenbankschemas durchzuführen.

Literaturverzeichnis

[ABU79] A. V. Aho, C. Beeri, and J. D. Ullman. The Theory of Joins in Relational Databases. 4(3), 1979.

[AH18a] Tanja Auge and A. Heuer. Inverse im Forschungsdatenmanagement: Eine Kombination aus Provenance Management, Schema- und Daten-Evolution. 2018.

[AH18b] Tanja Auge and Andreas Heuer. Inverses in Research Data Management: Combining Provenance Management, Schema and Data Evolution (Inverse im Forschungsdatenmanagement). In Gerhard Klassen and Stefan Conrad, editors, *Proceedings of the 30th GI-Workshop Grundlagen von Datenbanken, Wuppertal, Germany, May 22-25, 2018*, volume 2126 of *CEUR Workshop Proceedings*, pages 108–113. CEUR-WS.org, 2018.

[AH18c] Tanja Auge and Andreas Heuer. The Theory behind Minimizing Research Data: Result equivalent CHASE-inverse Mappings. In *LWDA*, 2018.

[AH19] Tanja Auge and Andreas Heuer. *ProSA—Using the CHASE for Provenance Management*, pages 357–372. 08 2019.

[Aug17] Tanja Auge. Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen. Master's thesis, Universität Rostock, 2017.

[Aug21a] Tanja Auge. Arbeitsnotizen: Ablauf in ProSA. 2021.

[Aug21b] Tanja Auge. Arbeitsnotizen: Parser: SQL to s-t tgd. 2021.

[Aug22a] Tanja Auge. Arbeitsnotizen: Aggregation. 2022.

[Aug22b] Tanja Auge. ChaTEAU. `https://git.informatik.uni-rostock.de/ta093/chateau`, 2022.

[Aug22c] Tanja Auge. ProSA. `https://git.informatik.uni-rostock.de/ta093/prosa`, 2022.

[BHSS17] Ilvio Bruder, Andreas Heuer, Sebastian Schick, and Sascha Spors. Konzepte für das Forschungsdatenmanagement an der Universität Rostock(Concepts for the Management of Research Data at the University of Rostock). In

Michael Leyer, editor, *Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Rostock, Germany, September 11-13, 2017*, volume 1917 of *CEUR Workshop Proceedings*, page 165. CEUR-WS.org, 2017.

[BKM+17] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the Chase. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, page 37–52. Association for Computing Machinery, 2017.

[CCT09] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in Databases: Why, How, and Where. *Found. Trends Databases*, 1(4):379–474, 2009.

[CMZ08] Carlo Curino, Hyun Jin Moon, and Carlo Zaniolo. Graceful database schema evolution: the PRISM workbench. *Proc. VLDB Endow.*, 1(1):761–772, 2008.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, 1970.

[DH13] Alin Deutsch and Richard Hull. *Provenance-Directed Chase&Backchase*, volume 8000, pages 227–236. 01 2013.

[Fag07] Ronald Fagin. Inverting schema mappings. *ACM Trans. Database Syst.*, 32(4):25, 2007.

[FKPT05] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

[FKPT08] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Quasi-inverses of schema mappings. *ACM Trans. Database Syst.*, 33(2):11:1–11:52, 2008.

[FKPT11a] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Reverse data exchange: Coping with nulls. *ACM Trans. Database Syst.*, 36(2):11:1–11:42, 2011.

[FKPT11b] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Schema Mapping Evolution Through Composition and Inversion. In Zohra Bellahsene, Angela Bonifati, and Erhard Rahm, editors, *Schema Matching and Mapping*, Data-Centric Systems and Applications, pages 191–222. Springer, 2011.

[Gö20] Andreas Görres. Erweiterung des CHASE-Werkzeugs ChaTEAU um ein Terminierungs- kriterium. Master's thesis, Universität Rostock, 2020.

[GMS12] Sergio Greco, Cristian Molinaro, and Francesca Spezzano. *Incomplete Data and Data Dependencies in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[Gro21] The PostgreSQL Global Development Group. PostgreSQL 14.1 Documentation. https://www.postgresql.org/docs/, 2021. Abgerufen am: 04.02.2022.

[Han22] Moritz Hanzig. Ein Rahmengerüst für ProSA. Bachelor's thesis, Universität Rostock, 2022.

[HSS18] Andreas Heuer, Gunter Saake, and Kai-Uwe Sattler. *Datenbanken - Konzepte und Sprachen, 6. Auflage*. MITP, 2018.

[JSQ22] JSQLParser. What is JSqlParser? `https://github.com/JSQLParser/JSqlParser/wiki`, 2022. Abgerufen am: 14.03.2022.

[Jur18] Martin Jurklies. CHASE und BACKCHASE: Entwicklung eines Universal-Werkzeugs für eine Basistechnik der Datenbankforschung. Master's thesis, Universität Rostock, 2018.

[Kav22] Ivo Kavisanczki. Erweiterung des ProSA-Parsers. Bachelor's thesis, Universität Rostock, 2022.

[KRSZ21] Ivo Kavisanczki, Tobias Rudolph, Tom Siegl, and Marian Zuska. Projektdokumentation: sql2sttgd. Technical report, Universität Rostock, Fakultät für Informatik, 2021.

[MMS79] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing Implications of Data Dependencies. 4(4), 1979.

[Pér13] Jorge Pérez. The Inverse of a Schema Mapping. In Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt, editors, *Data Exchange, Integration, and Streams*, volume 5 of *Dagstuhl Follow-Ups*, pages 69–95. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[Sch96] Edwin Schicker. *Datenbanken und SQL - eine praxisorientierte Einführung*. Informatik und Praxis. Teubner, 1996.

[TNRH19] Anika Tabassum, Anannya Islam Nady, and Mohammad Rezwanul Huq. Mathematical Formulation and Implementation of Query Inversion Techniques in RDBMS for Tracking Data Provenance. In *2019 7th International Conference on Information and Communication Technology (ICoICT)*, 2019.

[Tut21] SQL Tutorial. SQL ALL. https://www.sqltutorial.org/sql-all/, 2021. Abgerufen am: 30.01.2022.

[WWO+21] Anja Wolpers, Anne-Sophie Waterstradt, Judith-Henrike Overath, Melinda Heuser, and Leonie Förster. Projektdokumentation: KSWS Data Provenance. Technical report, Universität Rostock, Fakultät für Informatik, 2021.

[Zim20] Jakob Zimmer. Vereinheitlichung des CHASE auf Instanzen und Anfragen am Beispiel ChaTEAU. Bachelor's thesis, Universität Rostock, 2020.

[Zim21] Jakob Zimmer. Datenintegration durch inverse Schemaabbildungen: Erweiterung der Rostocker GaLVE-Technik. Master's thesis, Universität Rostock, 2021.

Eidesstattliche Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Rostock, 15. März 2022

Dennis Spolwind

A Quelltext für Invertierungsmethode

Im Folgendem wird der Quelltext für die Umsetzung der Invertierungmethode bzw. für die Algorithmen 5 und 7 dargestellt. Hierbei wird der Algorithmus 5 in der Methode *invert* und der Algorithmus 7 in der Methode *invertAgg* realisiert.

```
package inverse;

import atom.RelationalAtom;
import integrityConstraint.STTgd;
import integrityConstraint.Tgd;
import org.apache.commons.lang3.tuple.Pair;
import term.Term;
import term.Variable;
import term.VariableType;

import java.util.*;

public class InvertingTechniques {
  public static Pair<LinkedHashSet<STTgd>, Optional<Tgd>> invert(
      LinkedHashSet<STTgd> sttgd, LinkedHashSet<Tgd> tgd) {
     HashMap<Integer, LinkedHashSet<STTgd>> invertedSttgd = new HashMap
         <>();
     HashMap<Integer, LinkedHashSet<Tgd>> invertedTgd = new HashMap<>();
     // Step 1, 2 ,3: Normalised and Inverting

     int key = 0;
     for (STTgd sttgdEl : sttgd) {
        LinkedHashSet<STTgd> sttgdList = new LinkedHashSet<>();
        for (RelationalAtom atom : sttgdEl.getHead()) {
           LinkedHashSet<RelationalAtom> newHead = new LinkedHashSet<>();
           newHead.add(atom);
           STTgd newSttgd = new STTgd(newHead, sttgdEl.getBody());
           sttgdList.add(newSttgd);
         }
      invertedSttgd.put(key, sttgdList);
      key++;

     }
     key = 0;

     for (Tgd tgdEl : tgd) {
       LinkedHashSet<Tgd> tgdList = new LinkedHashSet<>();
       for (RelationalAtom atom : tgdEl.getHead()) {
```

```
        LinkedHashSet<RelationalAtom> newHead = new LinkedHashSet<>();
        newHead.add(atom);
        Tgd newTgd = new Tgd(newHead, tgdEl.getBody());
        tgdList.add(newTgd);

      }
    invertedTgd.put(key, tgdList);
    key++;

}
    // Step 4: Replacing
boolean adjust = true;
while (adjust) {
    adjust = false;
      for (Map.Entry<Integer, LinkedHashSet<STTgd>> entrySttgd :
          invertedSttgd.entrySet()) {
          LinkedHashSet<STTgd> newSttgdSet = new LinkedHashSet<>(
              entrySttgd.getValue());
          // STTgd for Head-Atom
            for (STTgd sttgdHead : entrySttgd.getValue()) {
                STTgd sttgdTemp = sttgdHead;
                 // Replacing in Sttgds
                 for (Map.Entry<Integer, LinkedHashSet<STTgd>>
                     entrySttgdComp : invertedSttgd.entrySet()) {
                      //STTgd for Body-Atom
                      for (STTgd sttgdBody : entrySttgdComp.getValue())
                          {
                           if (!sttgdHead.equals(sttgdBody)) {
                               for (RelationalAtom headAtom : sttgdHead.
                                   getHead()) {
                                    if (sttgdBody.getBody().contains(
                                        headAtom)) {
                                     LinkedHashSet<RelationalAtom>
                                         newHead = new LinkedHashSet<>(
                                         sttgdTemp.getHead());
                                     newHead.remove(headAtom);
                                     newHead.addAll(sttgdBody.getHead()
                                         );

                                     STTgd newSttgd = new STTgd(
                                         sttgdTemp.getBody(), newHead);

                                     //Remove both Sttgds
                                     newSttgdSet.remove(sttgdTemp);
                                     sttgdTemp = newSttgd;

                                     if (!entrySttgd.equals(
                                         entrySttgdComp)) {
                                      LinkedHashSet<STTgd> removeSet
                                          = new LinkedHashSet<>(
```

```
                                        entrySttgdComp.getValue());
                                    removeSet.remove(sttgdBody);
                                    invertedSttgd.put(
                                        entrySttgdComp.getKey(),
                                        removeSet);
                                } else {
                                    newSttgdSet.remove(sttgdBody);
                                }
                                newSttgdSet.add(newSttgd);
                                adjust = true;
                            }

                        }
                    }
                }
                // Replacing in Tgds
                for (Map.Entry<Integer, LinkedHashSet<Tgd>> entryTgd :
                    invertedTgd.entrySet()) {
                    // Body-Atom
                    for (Tgd tgdBody : entryTgd.getValue()) {
                        for (RelationalAtom headAtom : sttgdHead.
                            getHead()) {
                            if (tgdBody.getBody().contains(headAtom))
                                {
                                LinkedHashSet<RelationalAtom> newHead
                                    = new LinkedHashSet<>(sttgdTemp.
                                    getHead());
                                newHead.remove(headAtom);
                                newHead.addAll(tgdBody.getHead());
                                STTgd newSttgd = new STTgd(sttgdTemp.
                                    getBody(), newHead);
                                newSttgdSet.remove(sttgdTemp);
                                newSttgdSet.add(newSttgd);
                                sttgdTemp = newSttgd;
                                adjust = true;

                            }

                        }
                    }


                }
                invertedSttgd.put(entrySttgd.getKey(), newSttgdSet);
            }
        }
    }

    // Step 6: Delete negated Head-Atoms
```

```
for (Map.Entry<Integer, LinkedHashSet<STTgd>> entrySttgd :
    invertedSttgd.entrySet()) {
    // STTgd for Head-Atom
    LinkedHashSet<STTgd> newSttgdSet = new LinkedHashSet<>(
        entrySttgd.getValue());
    for (STTgd sttgdHead : entrySttgd.getValue()) {
        STTgd newSttgd = sttgdHead;
        for (RelationalAtom headAtom : sttgdHead.getHead()) {
            LinkedHashSet<RelationalAtom> newHead = new
                LinkedHashSet<>(newSttgd.getHead());
            if (headAtom.isNegated()) {
                newHead.remove(headAtom);
                newSttgd.setHead(newHead);
            }
        }
        newSttgdSet.remove(sttgdHead);
        newSttgdSet.add(newSttgd);
    }
    invertedSttgd.put(entrySttgd.getKey(), newSttgdSet);
}

// Step 7: Resume Sttgds

for (Map.Entry<Integer, LinkedHashSet<STTgd>> entrySttgd :
    invertedSttgd.entrySet()) {
    for (STTgd sttgdEl : entrySttgd.getValue()) {
        STTgd sttgdTemp = sttgdEl;
        for (STTgd sttgdComp : entrySttgd.getValue()) {
            if (!sttgdEl.equals(sttgdComp)) {
                if (sttgdEl.getHead().containsAll(sttgdComp.
                    getHead())) {
                    LinkedHashSet<RelationalAtom> newBody = new
                        LinkedHashSet<>(sttgdEl.getBody());
                    newBody.addAll(sttgdComp.getBody());
                    LinkedHashSet<STTgd> newSttgdSet = new
                        LinkedHashSet<>(entrySttgd.getValue());
                    STTgd newSttgd = new STTgd(newBody, sttgdEl.
                        getHead());
                    newSttgdSet.remove(sttgdTemp);
                    newSttgdSet.remove(sttgdComp);
                    newSttgdSet.add(newSttgd);
                    sttgdTemp = newSttgd;
                    invertedSttgd.put(entrySttgd.getKey(),
                        newSttgdSet);
                }
            }

        }

    }
```

```
}


// Step 9: Adjusting the quantifiers
for (Map.Entry<Integer, LinkedHashSet<STTgd>> entrySttgd :
    invertedSttgd.entrySet()) {
    LinkedHashSet<STTgd> newSttgdList = new LinkedHashSet<>(
        entrySttgd.getValue());
    for (STTgd sttgdEl : entrySttgd.getValue()) {
        LinkedHashSet<RelationalAtom> newHead = new LinkedHashSet
            <>(sttgdEl.getHead());
        for (RelationalAtom headAtom : sttgdEl.getHead()) {
            ArrayList<Term> newTermListHead = new ArrayList<>(
                headAtom.getTerms());
            for (Term termEl : headAtom.getTerms()) {
                Term termTemp = termEl;
                boolean exists = true;
                if (termEl.isVariable()) {
                    for (RelationalAtom bodyAtom : sttgdEl.getBody
                        ()) {
                        for (Term termComp : bodyAtom.getTerms())
                            {
                            if (exists) {
                                if (termComp.getName().equals(
                                    termEl.getName())) {
                                    Term newTerm = new Variable(
                                        VariableType.FOR_ALL,
                                        termTemp.getName(), ((
                                        Variable) termTemp).
                                        getIndex());
                                    newTermListHead.remove(
                                        termTemp);
                                    newTermListHead.add(newTerm);
                                    termTemp = newTerm;
                                    exists = false;

                                } else {
                                    Term newTerm = new Variable(
                                        VariableType.EXISTS,
                                        termTemp.getName(), ((
                                        Variable) termTemp).
                                        getIndex());
                                    newTermListHead.remove(
                                        termTemp);
                                    newTermListHead.add(newTerm);
                                    termTemp = newTerm;

                                }
                            }
```

```
                        }
                    }
                }
            }
            RelationalAtom newHeadAtom = new RelationalAtom(
                headAtom.getName(), newTermListHead);
            newHead.remove(headAtom);
            newHead.add(newHeadAtom);

        }
        STTgd newSttgd = new STTgd(sttgdEl.getBody(), newHead);
        newSttgdList.remove(sttgdEl);
        newSttgdList.add(newSttgd);
        invertedSttgd.put(entrySttgd.getKey(), newSttgdList);
    }
}

for (Map.Entry<Integer, LinkedHashSet<STTgd>> entrySttgd :
    invertedSttgd.entrySet()) {
    LinkedHashSet<STTgd> newSttgdList = new LinkedHashSet<>(
        entrySttgd.getValue());
    for (STTgd sttgdEl : entrySttgd.getValue()) {
        LinkedHashSet<RelationalAtom> newBody = new LinkedHashSet
            <>(sttgdEl.getBody());
        for (RelationalAtom bodyAtom : sttgdEl.getBody()) {
            ArrayList<Term> newTermListBody = new ArrayList<>(
                bodyAtom.getTerms());
            for (Term termEl : bodyAtom.getTerms()) {
                if(termEl.isVariable()){
                    Term newTerm = new Variable(VariableType.
                        FOR_ALL, termEl.getName(),((Variable)termEl)
                        .getIndex());
                    newTermListBody.remove(termEl);
                    newTermListBody.add(newTerm);

                }
            }
            RelationalAtom newBodyAtom = new RelationalAtom(
                bodyAtom.getName(),newTermListBody);
            newBody.remove(bodyAtom);
            newBody.add(newBodyAtom);
        }
        STTgd newSttgd = new STTgd(newBody, sttgdEl.getHead());
        newSttgdList.remove(sttgdEl);
        newSttgdList.add(newSttgd);
        invertedSttgd.put(entrySttgd.getKey(), newSttgdList);
    }
}

LinkedHashSet<STTgd> result = new LinkedHashSet<>();
```

```
        invertedSttgd.forEach( (k, v) -> {
            result.addAll(v);
        });

        return Pair.of(result,Optional.empty());

    }
    public static Pair<LinkedHashSet<STTgd>, Optional<Tgd>> invertAgg(
        LinkedHashSet<STTgd> sttgd, LinkedHashSet<Tgd> tgd, RelationalAtom
        sideTable) {

        Pair<LinkedHashSet<STTgd>, Optional<Tgd>> invertedSttgd = invert(
            sttgd,tgd);
        Tgd resultTgd = new Tgd(new LinkedHashSet<>(),new LinkedHashSet
            <>());
        STTgd resultSttgd = new STTgd(new LinkedHashSet<>(),new
            LinkedHashSet<>());
        for(STTgd sttgdEl : invertedSttgd.getLeft()){
            RelationalAtom newNegatedSideTable = new RelationalAtom(
                sideTable.getName()+"'",sideTable.getTerms(),true);
            LinkedHashSet<RelationalAtom> newBody = new LinkedHashSet<>(
                sttgdEl.getBody());
            newBody.add(sideTable);
            newBody.add(newNegatedSideTable);
            LinkedHashSet<RelationalAtom> newHead = new LinkedHashSet<>();
            for (RelationalAtom headAtom : sttgdEl.getHead()){
                RelationalAtom newHeadAtom = new RelationalAtom(headAtom.
                    getName()+"'",headAtom.getTerms(),false);
                newHead.add(newHeadAtom);
                resultSttgd.getHead().add(headAtom);
                resultSttgd.getBody().add(newHeadAtom);
            }
            newHead.add(new RelationalAtom(newNegatedSideTable.getName(),
                newNegatedSideTable.getTerms(), false));
            resultTgd.setHead(newHead);
            resultTgd.setBody(newBody);
        }
        LinkedHashSet<STTgd> resultSttgdSet = new LinkedHashSet<>();
        resultSttgdSet.add(resultSttgd);

        return Pair.of(resultSttgdSet,Optional.of(resultTgd));
    }
}
```

B Datenträger

Aufgrund dessen, dass für diese Arbeit eine digitale Abgabe erfolgt, wird kein explizierter Datenträger mitgeliefert. Dementsprechend wird die verwendete Literatur und der erstellte Quelltext für die Invertierung online bereitgestellt.
Um die Verfügbarkeit der verwendeten Literatur zu gewährleisten, wurde die Literatur im Studip der Universität Rostock hochgeladen. Eine Ausnahme davon bilden ganze Bücher, wie beispielsweise [HSS18, Sch96], und Master- und Bachelorarbeiten, wie beispielsweise [Kav22, Han22]. Die restliche verwendete Literatur ist in folgender Studip-Veranstaltung zu finden:

- Veranstaltung: MA: Inverse Anfragen in ProSA
- Link: `https://studip.uni-rostock.de/dispatch.php/course/overview?cid=5b12e4cb5b8e33be6f5f55ca67bf60d1`

Der Programmiercode ist im Projekt: ProSA [Aug22c] verfügbar, wobei folgender Branch verwendet wurde:

- Branch: ds570_inverse
- Link: `https://git.informatik.uni-rostock.de/ta093/prosa/-/tree/ds570_inverse`