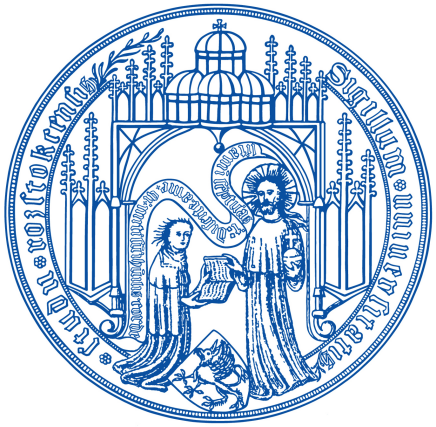
Erweiterung des CHASE-Werkzeugs ChaTEAU um eine BACKCHASE-Phase

Masterarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von: Florian Rose
geboren am: 31.05.1996 in Rostock
Erstgutachter: Prof. Dr. rer. nat. habil. Andreas Heuer
Zweitgutachter: apl. Prof. Dr.-Ing. habil. Meike Klettke
Betreuer: Tanja Auge
Abgabedatum: 23.09.2020

Inhaltsverzeichnis

Korrekturen

Die folgenden Korrekturen wurden nach dem Einreichen und der Bewertung der Arbeit ergänzt. Die jeweiligen Abschnitte sind referenziert.

- Ergänzung von Referenzen für [AH18a] in den Abschnitten 4.3.2 und 4.3.3 des Konzeptkapitels 4.
- Hervorhebung der Abgrenzung des Konzeptes zu den vorangegangenen Arbeiten [AH18a], [AH18b] und [AH19] in Abschnitt 4.3.4 des Kapitels 4.
- Korrigieren eines Tippfehlers von I_{Prov} in Abschnitt 4.3.4 des Kapitels 4.
- Entfernen des rechten Tabellenrandes in den Abbildungen 4.5 und 4.6 im Konzeptkapitel 4.
- Vervollständigung der Literaturangaben zum Studentenprojekt [SRH+20].

Kapitel 1

Einleitung

1.1 Motivation

Datenbanken sind heutzutage der Standard für die persistente Speicherung von Daten. Je nach Verwendungszweck ergeben sich häufig Zusammenhänge zwischen den Datensätzen selbst oder den verschiedenen Eigenschaften eines Datensatzes, die sich aus dem Kontext der Anwendung ergeben. Diese Zusammenhänge müssen über den gesamten Datenbestand konsistent gehalten werden, um Fehler beim Einfügen, Löschen oder Ändern der Daten zu verhindern. Diese Bedingungen bezeichnet man allgemein als Integritätsbedingungen. Dabei können diese Abhängigkeiten verschiedene Formen haben. Sie können die Gleichheit von Werten erzwingen oder die Existenz anderer Daten voraussetzen.

> **Beispiel:** Integritätsbedingungen
> Eine Universität hält Daten über ihre Studenten, die für den Versand der Post, die Ausstellung von Zertifikaten und Ähnlichem benötigt werden. Dazu wird neben dem vollen Namen eines Studenten auch die Matrikelnummer, dessen Anschrift und sein Studiengang gespeichert.
> Klar ist, dass auch ein Student mit Zweitwohnsitz nur an eine Adresse seine Post zugestellt bekommen soll. Folglich ergibt sich hieraus, dass die Matrikelnummer eines Studenten eindeutig die Adresse bestimmen muss (Gleichheit von Werten).
> Zudem muss jeder Student einen Studiengang aus dem Katalog der Universität studieren. Deshalb muss für jeden Studiengang, der in der obigen Tabelle existiert, auch ein Eintrag in der Studiengangstabelle existieren (Existenz von Daten).

Der aus [ABU79] und [MMS79] hervorgegangene CHASE, ein Algorithmus zur Berechnung der Implikation solcher Abhängigkeiten, ist bereits seit langer Zeit Gegenstand der Datenbankforschung. Er kann als eine Art Multifunktionswerkzeug verstanden werden, welches vielseitig im Feld der Datenbankforschung eingesetzt werden kann. Zu den Anwendungsgebieten des CHASE gehören unter anderem das Überprüfen von Anfragen auf Äquivalenz, die Datenintegration und Data-Cleaning [GMPS13]. Für einige Anwendungen liefert der CHASE Zwischenergebnisse, die erst mit weiteren Schritten, der sogenannten BACKCHASE-Phase, zum Ziel führen. Ein Beispiel hierfür ist das Beantworten von Anfragen mit Sichten (*Answering Queries using Views*) [DH13]. Dies ist häufig bei sensiblen Daten interessant, um dem Datenschutz gerecht zu werden, z. B. bei Patientendaten. Weiter kann der CHASE mit anschließender BACKCHASE-Phase genutzt werden, um Anfragen unter Integritätsbedingungen zu optimieren oder Daten bei wissenschaftlichen Auswertungen zurückzuverfolgen (*Provenance*) [AH18a]. Letzteres ist einerseits für die Nachvollziehbarkeit von Forschungsergebnissen wichtig, kann aber auch Forschungseinrichtungen helfen, mit wandelnden Strukturen in Forschungsdaten umzugehen, indem sie wissen, welche Daten aufgehoben werden müssen, um bestimmte Anfragen wiederholen zu können.
Viele Systeme, die den CHASE implementieren, sind auf spezielle Anwendungen zugeschnitten. Sie sind

durch die entsprechenden Anwendungsfälle motiviert, z. B. Datenmigration oder Anfrageoptimierung. Mit ChaTEAU soll ein System mit einem breiten Anwendungsspektrum geschaffen werden. Dieses wurde bereits durch mehrere Abschlussarbeiten erweitert, sodass auch diese Arbeit zur Erweiterung ChaTEAUs beitragen soll.

1.2 Problemstellung

In dieser Arbeit soll ChaTEAU um eine BACKCHASE-Phase erweitert werden. Dazu werden zunächst die verschiedenen Varianten von BACKCHASE-Phasen verglichen. Weiter werden relevante Optimierungen der BACKCHASE-Phase gegenüber den klassischen Ansätzen und deren Einsatzfähigkeit in ChaTEAU diskutiert. Besonderes Augenmerk liegt hierbei auf dem BACKCHASE im Bereich des Provenance-Management. Dort soll durch den BACKCHASE eine Teilinstanz der Datenbank, auf welcher eine Anfrage ausgeführt wurde, aus dem Ergebnis dieser rekonstruiert werden. Ziel dessen ist, die Anfrage auch unter Evolution der Datenbank reproduzierbar zu halten. Dies ist eine wichtige Anforderung an wissenschaftliche Auswertungen.

1.3 Gliederung der Arbeit

In diesem Kapitel wurde die Relevanz des CHASE in der Datenbankforschung sowie seine vielseitigen Einsatzmöglichkeiten aufgezeigt. Zudem wurde die Problemstellung der Arbeit dargestellt. Im folgenden Kapitel 2 werden zunächst die Grundlagen des CHASE und BACKCHASE vorgestellt und wichtige Begriffe definiert. Anschließend werden in Kapitel 3 Stand der Forschung und Stand der Technik gezeigt. Im Stand der Forschung wird dabei besonders auf den BACKCHASE eingegangen, während sich der Stand der Technik vorrangig mit ChaTEAU beschäftigt. Im Kapitel 4 wird ein Konzept für die BACKCHASE-Phase im Provenance-Fall aufgestellt. Im darauffolgenden Kapitel 5 werden die im Konzept vorgestellten Ergebnisse prototypisch implementiert und anschließend in Kapitel 6 bezüglich ihrer Funktionen getestet. Abschließend werden die Ergebnisse dieser Arbeit in Kapitel 7 zusammengefasst und diskutiert. Hier findet sich auch ein Ausblick auf noch offene Fragestellungen.

Kapitel 2

Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen der Arbeit dargestellt. Dazu werden im Folgenden Begriffe so definiert, wie sie im Kontext der Arbeit zu verstehen sind. Weiter werden die Grundlagen für den CHASE und BACKCHASE vorgestellt.

2.1 Definitionen

In diesem Abschnitt werden grundlegende Begriffe definiert, die im Laufe der Arbeit verwendet werden. Hierzu wird neben einer formalen Definition stets auch ein Beispiel zum besseren Verständnis gegeben.

Schema

Schemata beschreiben die Struktur der Daten, die in einer Datenbank abgelegt werden. Sie sagen somit aus, welche Werte gespeichert werden können und was diese bedeuten.

Definition: Relationenschema, Datenbankschema nach [HSS18]
Ein Relationenschema R ist eine endliche, nichtleere Menge von Attributen A_i. Ein Datenbankschema S ist eine Menge von Relationenschemas. Dabei gilt:

$$R = \{A_1, ..., A_n\}, \text{ mit } n \in \mathbb{N}$$
$$S = \{R_1, ..., R_m\}, \text{ mit } m \in \mathbb{N}$$

Beispiel: Relationenschema, Datenbankschema
Gegeben seien die Attribute Matrikelnummer, Studiengang, Vorname und Nachname. Diese charakterisieren einen typischen Studenten. Weiter sei der Studiengang durch seinen Namen und das dazugehörige Institut beschrieben. Ein Datenbankschema mit den dazugehörigen Relationenschemas ist dann:

$$R_{\text{Student}} = \{\text{Matrikelnummer, Studiengang, Vorname, Nachname}\},$$
$$R_{\text{Studiengang}} = \{\text{Studiengangsname, Institut}\},$$
$$S = \{R_{\text{Student}}, R_{\text{Student}}\}.$$

Relation

Relationen sind Instanzen eines Relationenschemas. Sie enthalten daher konkrete Datensätze.

Definition: Relation, Tupel nach [HSS18]
Eine Relation r über einem Relationenschema R ist eine Menge von Tupeln t mit

$$t : R \to \bigcup_{D \in \mathcal{D}} D.$$

Hierbei ist $\mathcal{D}$ jene Menge, welche den Datentypen ihre *Wertebereiche* zuordnet. Der Wertebereich eines Datentyps drückt aus, welche Einträge für diesen gültig sind, z. B. Zeichenketten für den Datentyp VARCHAR. Ein $D \in \mathcal{D}$ heißt *Domäne*. Weiter gilt, dass $t(A) \in dom(A)$, wobei $dom(A)$ die Domäne von A ist. Ein $w \in dom(A)$ heißt Attributwert für A.

Beispiel: Relation
Eine Relation über dem Relationenschema R_{Student} kann wie folgt aussehen:

Matrikelnummer	Studiengang	Vorname	Nachname
0042	Physik	Doug	Adam
3200	Germanistik	Georg	Buches
4711	Informatik	Ronald	Maier

Tabelle 2.1: Studentenrelation

Zudem gilt $dom(\text{Matrikelnummer}) = INTEGER$ und die Domänen der anderen Attribute sind jeweils Zeichenketten ($VARCHAR$).

Relationenalgebra

Die Relationenalgebra ist eine Sprache zur Manipulation von Relationen. Sie kann genutzt werden, um Anfragen an Relationen zu formulieren.

Definition: Relationenalgebra
Die relationale Algebra kennt folgende grundlegende Operatoren:

- Mengenoperationen: $r \cup s, r \cap s, r - s$
- Join-Operatoren
  - Natural Join: $r \bowtie s$
  - Equi-Join: $r \bowtie_{A=B} s$
  - Semi-Join: $r \ltimes s, r \rtimes s$
  - Outer Join: $r \mathbin{⟕} s, r \mathbin{⟖} s, r \mathbin{⟗} s$
- Kartesisches Produkt: $r \times s$
- Projektion: $\pi_A(r)$
- Selektion[1]

[1] komplexere Selektionsprädikate hier nicht betrachtet, boolesche Ausdrücke über Mengenoperationen realisierbar

  - Attribut = Konstante: $\sigma_{A=c}(r)$
  - Attribut = Attribut: $\sigma_{A=B}(r)$
- Umbenennung: $\beta_{s(B_1 \leftarrow A_1, \ldots B_n \leftarrow A_n)}(r)$

Hierbei sind r und s Relationen, A und B Attribute und c eine beliebige Konstante.

Beispiel: Relationenalgebra
Möchten wir aus unseren obigen Beispielrelationen (2.1) die Matrikelnummer aller Studenten an der Fakultät für Informatik und Elektrotechnik (IEF) wissen, lässt sich dies in Relationenalgebra wie folgt formulieren:

$$\pi_{\text{Matrikelnummer}}(r_{\text{Student}} \bowtie_{\text{Studiengang = Studiengangsname}} (\sigma_{\text{Institut='IEF'}}(r_{\text{Studiengang}})))$$

Abhängigkeiten

Abhängigkeiten drücken Beziehungen zwischen Attributen aus. Dabei können diese sowohl über eine Relation oder über mehrere definiert sein. Allgemein werden zwei Arten von Abhängigkeiten unterschieden, *tuple-generating dependencies*(TGDs) und *equality-generating dependencies*(EGDs).
EGDs stellen die Gleichheit bestimmter Werte sicher. Ein besonderer Fall von EGDs sind funktionale Abhängigkeiten (FDs).

Definition: FDs, EGDs
Eine funktionale Abhängigkeit ist eine Formel der Form

$$X \to Y, \text{ mit } X, Y \subseteq R, \text{ wobei}$$
$$X \to Y :\Leftrightarrow \forall t_1, t_2 \in r : (t_1 \neq t_2 \wedge t_1(X) = t_2(X)) \Rightarrow t_1(Y) = t_2(Y).$$

Eine EGD ist eine Formel der Form

$$\forall \vec{x} : F(\vec{x}) \to x_1 = x_2, \text{ mit } x_1, x_2 \in \vec{x}.$$

Hierbei ist $\vec{x}$ ein Variablenvektor. Seine Komponenten heißen $x_1, ..., x_n$ und F eine Konjunktion atomarer Formeln.
Eine *atomare Formel* über einem Datenbankschema ist ein Prädikat $R_i(\vec{x})$, wobei $R_i \in S$ ist und die Anzahl der Komponenten in $\vec{x}$ gleich der Stelligkeit von R_i ist.

Beispiel: EGD
Ein Student kann nur einen Studiengang belegen. Folglich muss über R_{Student} folgende funktionale Abhängigkeit gelten:

$$\text{Matrikelnummer} \to \text{Studiengang}$$

Diese sieht als allgemeine EGD formuliert wie folgt aus:

$$\forall ma, st_1, st_2, vn_1, vn_2, nn_1, nn_2 : (r_{\text{Student}}(ma, st_1, vn_1, nn_1) \wedge r_{\text{Student}}(ma, st_2, vn_2, nn_2)) \to st_1 = st_2$$

Hierbei wurden die Namen der Attribute aus der Studententabelle abgekürzt. Weiter gilt allgemein $\text{Relation}_1(x, y) = TRUE \Leftrightarrow (x, y) \in \text{Relation}_1$.

Definition: JDs, MVDs, TGDs, s-t-TGDs
Verbundabhängigkeiten (JDs für *join dependencies*) und mehrwertige Abhängigkeiten (MVDs für *multivalued dependencies*) sind spezielle TGDs.
Eine JD ist eine Abhängigkeit der Form

$$\bowtie [R_1, ..., R_n], \text{ mit } [R_1, ...R_n] \text{ sei Dekomposition eines Relationenschemas } R.$$

Eine Relation $r(R)$ erfüllt eine Verbundabhängigkeit dann, wenn folgende Bedingung gilt:

$$\bowtie_{i=1}^{n} \pi_{R_i}(r) = r$$

Eine JD drückt folglich aus, dass ein Verbund von $[R_1, ..., R_n]$ keine neuen Tupel generieren kann, die nicht bereits in r enthalten sind.

Eine MVD drückt aus, dass X nicht nur ein Y bestimmt, sondern eine Menge von Werten für Y, unabhängig von den anderen Attributen im Relationenschema. Daraus folgt, dass auch die restlichen Attribute (Z) mehrwertig durch X bestimmt werden. Daher muss für jede Kombination von Y und Z ein Tupel für das jeweilige X in der Relation existieren. Eine MVD ist eine Formel der Form

$$X \twoheadrightarrow Y, \text{ wobei zusätzlich } X \twoheadrightarrow Z, \text{ mit } Z = R - (X \cup Y), \text{ gilt}$$

Daraus folgt, dass eine MVD $X \twoheadrightarrow Y$ eine spezielle JD der Form $\bowtie [XY, XZ]$ ist.

TGDs sind Abhängigkeiten, welche die Generierung neuer Tupel hervorrufen können. Sie haben folgende Form:

$$\forall \vec{x} : F_1(\vec{x}) \rightarrow \exists \vec{y} : F_2(\vec{x}, \vec{y})$$

Hierbei gilt, dass F_1 und F_2 Konjunktionen atomarer Formeln sind, wobei in F_1 jede Komponente aus $\vec{x}$ mindestens einmal vorkommt. Eine TGD, deren *Kopf* (rechte Seite der Abhängigkeit) keinen Existenzquantor enthält, heißt *voll*. Nicht volle TGDs heißen *eingebettet*.
Eine s-t-TGD ist eine TGD, deren *Rumpf* (linke Seite der Abhängigkeit) über einem Quellschema S und Kopf über einem Zielschema S' definiert ist.

Beispiel: TGD
Im obigen Beispiel existiert neben der Studententabelle auch eine Relation für die Studiengänge. Ein Studiengang, der von einem Studenten belegt wird, muss auch in der Studiengangstabelle auftauchen. Diese Integritätsbedingung lässt sich durch folgende TGD ausdrücken:

$$\forall ma, st, vn, nn : r_{\text{Student}}(ma, st, vn, nn) \rightarrow \exists In : r_{\text{Studiengang}}(st, In)$$

Diese Definitionen können als allgemein anerkannt angesehen werden. In [GMS12] wird noch einmal im Detail auf die Definitionen der oben beschriebenen Abhängigkeiten eingegangen. TGDs und EGDs werden dort als Spezialfälle eingebetteter Abhängigkeiten eingeführt.

Anfragen

Eine Anfrage ist eine Folge von Operationen auf den Relationen einer Datenbank, die ihre Datensätze filtern und verknüpfen. Ein Anfrageergebnis kann wieder als Relation interpretiert werden. In dieser Arbeit beschränken wir uns auf konjunktive Anfragen.

Definition: konjunktive Anfrage nach [CM77]
Eine konjunktive Anfrage ist ein Ausdruck der Form

$$\exists \vec{y} : Q(\vec{x}, \vec{y}) \rightarrow x_1, ..., x_n, \text{ mit } x_1, ..., x_n \in \vec{x} \text{ und } Q \text{ ist Konjunktion atomarer Formeln.}$$

Mit ihnen lassen sich Anfragen mit Projektion, Selektion auf Gleichheit, natürlicher Verbund und Umbenennung ausdrücken.

Beispiel: Anfragen
Die obige Beispielanfrage in Relationenalgebra lässt sich als konjunktive Anfrage wie folgt schreiben:

$$\exists St, Vn, Nn : r_{\text{Student}}(ma, St, Vn, Nn) \wedge r_{\text{Studiengang}}(St, \text{'Informatik'}) \rightarrow ma$$

Um Anfragen übersichtlicher zu formulieren, verwenden wir im Folgenden eine Notation, die sich an Datalog orientiert. Per Konvention werden alle allquantifizierten Variablen klein und alle existenzquantifizierten Variablen groß geschrieben. Konstanten werden wie bei konjunktiven Anfragen als Werte, ggf. in einfachen Anführungszeichen, aufgeführt. Das Ergebnisses wird durch den Kopf der Anfrage repräsentiert, während die Operatoren im Rumpf der Regel wiederzufinden sind.

Beispiel: Datalog-Notation
Die vorangegangene Beispielanfrage kann in die Datalog-Notation wie folgt umgeschrieben werden:

$$Q(ma) \text{ :- } r_{\text{Student}}(ma, St, Vn, Nn) \wedge r_{\text{Studiengang}}(St, \text{'Informatik'})$$

2.2 CHASE

Der CHASE ist eine Technik, welche aus der Datenbankforschung als Werkzeug zum Schlussfolgern mit Integritätsbedingungen hervorging. Der CHASE wird auf eine Menge von Abhängigkeiten (Γ) und Objekte (d), z. B. eine Datenbank oder Anfrage, angewandt. Er erweitert diese Fakten durch *Forward-Chaining* so, dass sie die Abhängigkeiten erfüllen [BKM+17]. Der Algorithmus funktioniert grundlegend wie folgt:

```
FOREACH Trigger τ für eine Abhängigkeit γ ∈ Γ DO
  IF τ ist aktiv THEN
    IF γ hat Form ∀x⃗ : F1(x⃗) → ∃y⃗ : F2(x⃗, y⃗) THEN
      F2 anwenden, neue Tupel in d aufnehmen
    ELSE IF γ hat Form ∀x⃗ : F(x⃗) → x1 = x2 THEN
      IF x1, x2 sind Konstanten und x1 ≠ x2 THEN
        CHASE schlägt fehl
      ELSE IF x1 ist Konstante THEN
        x2 ← x1
      ELSE IF x2 ist Konstante ∨ x1, x2 sind Nullwerte THEN
        x1 ← x2
```

Definition: Trigger, aktiver Trigger, CHASE-Varianten
Ein Trigger für eine Abhängigkeit $\gamma \in \Gamma$ existiert, wenn ihr Kopf mittels eines Homomorphismus auf einen Teil des Objektes d abgebildet werden kann. Das heißt, dass das Muster der linken Seite der Abhängigkeit in d vorkommt.
Ein Trigger heißt *aktiv*, wenn durch Anwendung des Kopfes (Ersetzung von Variablen bei EGDs, Anwendung von F_2 bei TGDs) der Abhängigkeit unter dem gleichen Homomorphismus neue Werte

oder Tupel entstehen. Das bedeutet, dass γ noch nicht in d erfüllt ist.
Der hier vorgestellte Algorithmus ist der *Standard*-CHASE. Neben diesem existieren weitere Variationen, wie der *Oblivious*-CHASE, welcher vernachlässigt, ob gefundene Trigger aktiv sind, wodurch potenziell mehr Nullwerte (Datenbanken) oder neue Variablen (Anfragen) entstehen als beim Standard-CHASE. Eine weitere Variante ist der *Skolem*-CHASE, eine Verschärfung des Oblivious-CHASE. Dieser vernachlässigt ebenfalls die Überprüfung auf aktive Trigger, führt Nullwerte oder neue Variablen jedoch als Funktionssymbole ein, deren Parameter die allquantifizierten Variablen der Abhängigkeit sind. Dadurch fallen Nullwerte oder neue Variablen zusammen, falls sie von den gleichen Parametern abhängen.

Der CHASE ist hierbei vielseitig einsetzbar, da sich viele Bedingungen als TGDs oder EGDs formulieren lassen und die Berechnung ihrer Implikation verschiedene Ziele erfüllen kann. So ist es zum Beispiel auch möglich, mit dem CHASE auf Anfragen entsprechende Integritätsbedingungen in diese einzuarbeiten, wie im Fall *Answering Queries using Views*.

Beispiel: CHASE
In unserem laufenden Beispiel legen Studenten auch Prüfungen ab. Ergänzen wir also eine Tabelle für Noten und eine Sicht für die Noten der Informatikstudenten. Diese Sicht existiert aus Datenschutzgründen, damit Mitarbeiter, die für Informatikstudenten zuständig sind, keine Einblicke in die Noten der Studenten anderer Studiengänge bekommen können. Weiter gilt eine TGD, welche ausdrückt, dass Noten von Informatikstudenten auch in $V_{NoteInf}$ auftauchen müssen.

$$r_{Note}(Matrikelnummer, Modulnummer, Note),$$
$$v_{NoteInf}(Matrikelnummer, Modulnummer, Note),$$
$$\forall ma, vn, nn, mo, no : r_{Student}(ma, \text{'Informatik'}, vn, nn) \wedge r_{Note}(ma, mo, no)$$
$$\rightarrow v_{NoteInf}(ma, mo, no).$$

Weiter seien folgende Daten in den Relationen abgelegt:

Matrikelnummer	**Studiengang**	**Vorname**	**Nachname**
0042	Physik	Doug	Adam
3200	Germanistik	Georg	Buches
4711	Informatik	Ronald	Maier
9001	Informatik	Goetz	Kuhlmann

Tabelle 2.2: Studentenrelation $R_{Student}$

Matrikelnummer	**Modulnummer**	**Note**
0042	00001	1.3
3200	10001	2.0
4711	20001	1.0
4711	20002	1.7
9001	20001	1.3
9001	20003	1.0

Tabelle 2.3: Notentabelle R_{Note}

Matrikelnummer	Modulnummer	Note
4711	20001	1.0

Tabelle 2.4: Notenrelation für Informatikstudenten $R_{NoteInf}$

Offensichtlich ist $R_{NoteInf}$ noch nicht vollständig unter Beachtung der obigen TGD. Für die Vervollständigung der Relation kann der CHASE genutzt werden. Dafür suchen wir zunächst nach allen Triggern für die TGD. Es existieren genau vier Trigger mit den folgenden Homomorphismen:

Attribut	h_1	h_2	h_3	h_4
ma	4711	4711	9001	9001
vn	'Ronald'	'Ronald'	'Goetz'	'Goetz'
nn	'Maier'	'Maier'	'Kuhlmann'	'Kuhlmann'
mo	20001	20002	20001	20003
no	1.0	1.7	1.3	1.0

Tabelle 2.5: $r_{NoteInf}$ nach dem CHASE

Daraus ergeben sich folgende Formeln. Hierbei ist die linke Seite jeweils der Rumpf der TGD und die rechte Seite der Trigger für die Abhängigkeit und den entsprechenden Homomorphismus.

$$h_1 : r_{Student}(ma, \text{'Informatik'}, vn, nn) \wedge r_{Note}(ma, mo, no) \rightarrow$$
$$r_{Student}(4711, \text{'Informatik'}, \text{'Ronald'}, \text{'Maier'}) \wedge r_{Note}(4711, 20001, 1.0)$$
$$h_2 : r_{Student}(ma, \text{'Informatik'}, vn, nn) \wedge r_{Note}(ma, mo, no) \rightarrow$$
$$r_{Student}(4711, \text{'Informatik'}, \text{'Ronald'}, \text{'Maier'}) \wedge r_{Note}(4711, 20002, 1.7)$$
$$h_3 : r_{Student}(ma, \text{'Informatik'}, vn, nn) \wedge r_{Note}(ma, mo, no) \rightarrow$$
$$r_{Student}(9001, \text{'Informatik'}, \text{'Goetz'}, \text{'Kuhlmann'}) \wedge r_{Note}(9001, 20001, 1.3)$$
$$h_4 : r_{Student}(ma, \text{'Informatik'}, vn, nn) \wedge r_{Note}(ma, mo, no) \rightarrow$$
$$r_{Student}(9001, \text{'Informatik'}, \text{'Goetz'}, \text{'Kuhlmann'}) \wedge r_{Note}(9001, 20003, 1.0)$$

Anschließend wird für jeden Trigger überprüft, ob er aktiv ist. Falls ja, wird der Kopf der Abhängigkeit in die Datenbank aufgenommen. Die TGD ist in diesem Fall voll, sodass hierbei keine neuen Nullwerte entstehen können. Der einzige inaktive Trigger ist der für h_1, da für diesen Homomorphismus bereits ein Tupel in $r_{NoteInf}$ existiert, welches mit h_1 auf den Kopf der TGD abgebildet werden kann. Durch die anderen Trigger wird die Notenrelation für Informatikstudenten schrittweise wie folgt erweitert:

	Matrikelnummer	Modulnummer	Note
	4711	20001	1.0
h_2	4711	20002	1.7
h_3	9001	20001	1.3
h_4	9001	20003	1.0

Tabelle 2.6: $r_{NoteInf}$ nach dem CHASE

2.3 BACKCHASE

Unter dem BACKCHASE versteht man eine weitere Phase, die dem CHASE in manchen Anwendungsfällen folgt. Die Idee ist, dass der CHASE ein Zwischenergebnis für die eigentliche Anwendung liefert,

und das endgültige Ergebnis durch den BACKCHASE angewandt auf das Resultat des CHASE erreicht wird.
Im Fall des Beantwortens von Anfragen mittels Sichten (AQuV) dient der CHASE dazu, alle relevanten Relationen und Sichten in einem Universalplan zusammenzufassen. Hierbei ist das CHASE-Objekt die Anfrage, welche umgeschrieben werden soll. Mit dem BACKCHASE wird das CHASE-Ergebnis, in diesem Fall der Universalplan, zurück auf ein modifiziertes CHASE-Objekt der ursprünglichen CHASE-Phase, hier also eine Anfrage, transformiert. Dazu werden die Teilmengen des Universalplans, also beliebige Kombinationen von Relationen oder Views, hingehend ihrer Äquivalenz zur ursprünglichen Anfrage getestet. Da in diesem Fall Sichten genutzt werden, macht es Sinn, den Universalplan auf die darin enthaltenen Sichten einzuschränken. Dies hängt aber davon ab, ob man *Anwering Queries using Views* so versteht, dass ausschließlich Sichten in der umgeschriebenen Anfrage vorkommen sollen, oder ob man eine Anfrage sucht, welche möglichst viele Sichten enthält. Ungeachtet dessen, werden dann minimale äquivalente Rewritings der ursprünglichen Anfrage gesucht, welche sich aus dem ggf. modifizierten Universalplan bilden lassen.

2.4 Provenance

Provenance beschreibt den Ursprung bzw. die Herkunft der Daten einer wissenschaftlichen Auswertung und ist damit bedeutend für die Nachvollziehbarkeit von Statistiken und Aussagen in Publikationen. Generell wird zwischen vier Abstraktionsstufen der Provenance unterschieden:

Definition: Where-Provenance
Die Where-Provenance gibt an, aus welchen Quellen die Ergebnisdaten stammen. Dies wird mit der Angabe der Relationen und Datenbankschemata, welche an der Anfrage beteiligt waren, beantwortet [CCT09].

Definition: Why-Provenance
Die Why-Provenance einer Anfrage sagt aus, welche Daten zur Bildung des Ergebnisses beigetragen haben. Beantwortet wird die Why-Provenance durch die Ausgabe der Tupel, die das Ergebnis gebildet haben. Diese nennt man *Zeugenbasis*. Eine minimale Zeugenbasis ist die kleinste Menge von Tupeln, die benötigt wird, um das Anfrageergebnis zu bilden [BKT01].

Definition: How-Provenance
Die How-Provenance beschreibt, wie das Anfrageergebnis zustande kam. Hierzu werden sogenannte Provenance-Polynome angegeben. Diese Provenance-Polynome sagen aus, wie Tupel der Zeugenbasis kombiniert wurden, um das Ergebnis zu bilden [GKT07].

Definition: Why-Not-Provenance
Die Why-Not-Provenance zu einer Anfrage Q und einem Tupel t gibt an, warum dieses Tupel nicht Teil der Auswertung war. Um dies zu beantworten, können Eigenschaften von t angegeben werden, die bei einer Operation im Anfrageplan zum Herausfallen des Tupels führten [CJ09]. Alternativ kann auch eine minimal modifizierte Anfrage Q' angegeben werden, wobei $t \in Q'$ und $Q \subset Q'$ gilt. Minimal bedeutet in diesem Kontext, dass die Anfrage möglichst wenig verändert wurde. Dies bezieht sich sowohl auf das Ergebnis als auch die Operatoren. Q' sollte daher nicht viel mehr Tupel als Q enthalten. Umgekehrt, sollte die geänderte Anfrage aber auch nicht auf das fehlende Tupel zugeschnitten sein. Eine Anfrage Q' mit $Q' - Q = t$ lässt sich einfach konstruieren, indem man Q um $\vee\, TID = tid(t)$ erweitert. TID steht hierbei für *Tuple-Identifier*, ein für jedes Tupel eindeutiger Wert.

Die Techniken und Systeme, die im nachfolgenden Kapitel 3 vorgestellt werden, bauen auf diesen Grundlagen zum CHASE, BACKCHASE und zur Provenance auf. Dort stellen wir eine Auswahl an CHASE- und Provenance-Tools vor und betrachten, wie der CHASE für die verschiedenen Anwendungsfälle eingesetzt wird. Besonders gehen wir dabei auf den Provenance-Fall ein.

Kapitel 3

Stand der Technik & Forschung

In diesem Kapitel werden aktuelle Erkenntnisse aus der Forschung, welche über die Grundlagen hinausgehen, vorgestellt. Im Abschnitt zum Stand der Forschung werden neuere Ansätze für den CHASE und BACKCHASE behandelt. Dabei wird besonders auf diejenigen Algorithmen und Forschungsergebnisse eingegangen, welche speziell für den Provenance-Fall relevant sind. Im Teil zum Stand der Technik wird ChaTEAU als System vorgestellt und andere Systeme, die den CHASE und BACKCHASE implementieren, verglichen. Die CHASE-Tools können hier Anregungen für zukünftige Funktionalitäten ChaTEAUs geben, während das Anwendungsproblem der Provenance-Tools, obwohl sie nicht unbedingt den CHASE implementieren, ein Teilproblem des Problems ist, welches in Abschnitt 1.2 vorgestellt wurde.

3.1 Stand der Technik

In diesem Abschnitt werden ChaTEAU sowie einige weitere CHASE- und Provenance-Tools vorgestellt. Dabei wird auf den Hintergrund zu ChaTEAU und den Aufbau des Systems eingegangen. Zusätzlich werden die verschiedenen Tools auf ihre unterstützten Anwendungsfälle und die algorithmische Umsetzung verglichen.

3.1.1 ChaTEAU

Viele Tools, die den CHASE implementieren, entstanden aus bestimmten Anwendungsfällen heraus und sind auf diese deshalb optimiert. Dadurch besitzen diese Tools keine breiten Anwendungsgebiete. Daher wird an der Universität Rostock das Universalwerkzeug ChaTEAU (*Chase for Transforming, Evolving and Adapting databases, Universal approach*) entwickelt. Diese Java-Anwendung soll allgemeine CHASE-Anwendungen unterstützen. Die Notwendigkeit eines solchen Tools wurde in [AH19] aufgezeigt.
ChaTEAU wird in Etappen weiterentwickelt. Erste programmiertechnische Ansätze entstanden in der Masterarbeit [Jur18]. Hier wurden die theoretischen Grundlagen aus [BKM+17] in Java implementiert. Das Tool wurde dann durch weitere studentische Arbeiten erweitert. Zunächst wurde mit [Ren19] die Verarbeitung von Anfragen vorbereitet und das System um eine XML-basierte Ein- und Ausgabe erweitert, welche sich strukturell an der von PDQ, einem anderen CHASE-Tool, orientiert. Anschließend wurde durch [Zim20] der CHASE auf Instanzen und Anfragen in ChaTEAU vereinheitlicht. Zudem wurde das System auch um die Unterstützung von s-t-TGDs erweitert. Dies ist besonders für diese Arbeit wichtig, da diese Abhängigkeiten sehr relevant für Anwendungen im Bereich der Provenance sind. Hier werden sie vor allem im Kontext der Schemaevolution und Invertierung von wissenschaftlichen Auswertungen genutzt. Darauf wurde mit [Gö20] eine Erweiterung um Terminierungskriterien eingepflegt. Da der CHASE nicht zwingend terminiert, ist dies besonders entscheidend für eine potenzielle Verwendung von ChaTEAU mit realen Datenquellen.

3.1.2 CHASE-Tools

Im Rahmen eines Studentenprojekts an der Universität Rostock wurden verschiedene CHASE-Tools auf ihre Funktionalitäten untersucht. Die im Folgenden vorgestellten Ergebnisse beruhen auf den Erkenntnissen aus dieser Projektarbeit [ABG+20].

ChaseFun

ChaseFun ist ein System für den Datenaustausch[1]. Dabei implementiert ChaseFun den Oblivious Chase, welcher Trigger nicht darauf überprüft, ob sie aktiv sind. ChaseFun unterstützt allerdings nur s-t-TGDs und funktionale Abhängigkeiten auf der Zieldatenbank.

ChaseTEQ

ChaseTEQ[2] ist ein System, welches erlaubt, unvollständige Datenbanken zu reparieren und Anfragen auf ihnen zu formulieren. "TEQ" steht hierbei für die drei Hauptfunktionen der Anwendung, welche in Modulen vorliegen. ChaseT erlaubt die Definition von Abhängigkeiten und Überprüfung auf Terminierung des CHASE basierend auf verschiedenen Kriterien. ChaseE ist das Modul, welches den CHASE auf die Datenbank anwendet, um diese basierend auf den definierten Abhängigkeiten zu reparieren. Dabei entstehende Nullwerte werden gelabelt, um gleiche Werte erkennbar zu machen. ChaseQ erlaubt das Anfragen der reparierten Datenbank durch eine eingeschränkte SQL-Syntax. Zusätzlich bietet ChaseTEQ Funktionalitäten für die graphische Darstellung von Abhängigkeiten, verschiedene CHASE-Varianten und eine Schnittstelle für MySQL-Datenbanken.

Graal

Graal[3] ist ein Tool für das Beantworten von Anfragen unter Integritätsbedingungen. Dabei bietet Graal eine Daten-Schicht für die zugrundeliegenden Fakten. Diese können auch durch Schnittstellen zu verschiedenen Systemen eingespeist werden. Die in der Ontologie-Schicht definierten Regeln lassen sich durch einen integrierten Analysator auf Entscheidbarkeit und Vollständigkeit überprüfen lassen. Graal verwendet dabei sowohl Forward- als auch Backward-Chaining, welche als Methoden zur Schlussfolgerung auch in der logischen Programmierung verwendet werden. EGDs werden allerdings nicht unterstützt. Damit implementiert Graal nicht den CHASE sondern kann lediglich TGDs in die Fakten einarbeiten.

Llunatic

Llunatic[4] ist eine Anwendung für Data-Cleaning und Schema-Mapping. Dabei werden sowohl EGDs als auch TGDs und s-t-TGDs unterstützt. Llunatic beschränkt sich in den Anwendungsfällen auf das Reparieren von Datenbeständen mit EGDs und den Datenaustausch durch s-t-TGDs sowie TGDs und EGDs auf der Zieldatenbank. Weiter wird auch eine Anwendung unterstützt, in welcher nach dem Datenaustausch die Zieldatenbank bereinigt wird. Das Bereinigen der Zieldatenbank durch bestimmte EGDs ist z. B. nützlich, wenn mehrere Quelldatenbanken in eine Zieldatenbank integriert werden. So kann gezielt bestimmt werden, welche Quelle sich durchsetzen soll, wenn unterschiedliche Quellwerte auf einen Zielwert abgebildet werden sollen.

[1] Projektseite: github.com/dbunibas/chasebench/tree/master/tools/chasefun
[2] Projektseite: wwwinfo.deis.unical.it/chaseteq
[3] Projektseite: graphik-team.github.io/graal/
[4] Projektseite: github.com/donatellosantoro/Llunatic

PDQ

PDQ (*Proof-Driven Querying*)[5] ist ein Tool zur Anfrageoptimierung. Dabei werden sowohl Abhängigkeiten ausgenutzt, als auch die Kosten für Zugriffe berücksichtigt. PDQ generiert hierbei Anfragepläne für eine Anfrage Q über dem Schema S, indem nach Beweisen dafür, dass Q unter S beantwortbar ist, gesucht werden. Jeder Beweis stellt dabei einen Anfrageplan dar.

Vergleich

Die vorgestellten CHASE-Tools werden basierend auf ihren Funktionalitäten, vorgesehenen Anwendungsszenarien und der implementierten CHASE-Variante verglichen. Funktionalitäten, die genauerer Beschreibung benötigen, z. B. weil Funktionen nur eingeschränkt verfügbar sind, werden durch $\star$ gekennzeichnet und anschließend diskutiert. Graal und PDQ sind CHASE-ähnliche Logiksysteme und daher in der folgenden Tabelle nicht aufgeführt. Ein Vergleich dieser mit einigen der hier vorgestellten Tools, kann in [BKM+17] nachvollzogen werden. Dort werden die Systeme nach ähnlichen Kriterien wie im Folgenden verglichen, jedoch nicht auf ihre Einsatzgebiete und ihre Möglichkeiten zur Terminierungserkennung.

	ChaseFun	**ChaseTEQ**	**Llunatic**	**ChaTEAU**
EGDs	$\star_1$	✓	✓	✓
TGDs	×	✓	✓	✓
s-t-TGDs	✓	✓	✓	✓
CHASE-Variante	Oblivious	Standard, Oblivious, Skolem	$\star_2$	Standard
Terminierungs-erkennung	×	✓	×	✓
Anwendung	Daten-Austausch	Data-Cleaning	Data-Cleaning, Daten-Austausch	$\star_3$

Tabelle 3.1: Vergleich der CHASE-Tools

- $\star_1$ ChaseFun unterstützt funktionale Abhängigkeiten auf dem Zielschema. Folglich werden EGDs nicht voll unterstützt, sondern lediglich ein Spezialfall dieser.
- $\star_2$ Llunatic verwendet einen modifizierten Standard-CHASE. Details zu den Unterschieden finden sich in [GMPS14].
- $\star_3$ ChaTEAU wird als Universalwerkzeug entwickelt und deshalb nicht an Anwendungsszenarien gekoppelt. ChaTEAU ist jedoch in der Lage, den CHASE auf Anfragen und Instanzen zu berechnen, und kann somit für Data-Cleaning und den Datenaustausch verwendet werden. In dieser Arbeit wird eine BACKCHASE-Phase für die Anwendung im Provenance-Management eingeführt (Implementierung in Kapitel 5). Weiterhin wird hier auch die Invertierung von TGDs implementiert. Dadurch wird auch Query-Rewriting möglich.

3.1.3 Provenance-Tools

Zeitgleich zum Projekt bezüglich der CHASE-Tools hat eine zweite Studentengruppe Provenance-Tools untersucht. Diese implementieren nicht zwingender Weise den CHASE, haben aber ein ähnliches Anwendungsgebiet. Die folgenden Ausführungen beruhen auf den Ergebnissen dieser Projektgruppe [SRH+20].

[5] Projektseite: github.com/ProofDrivenQuerying/pdq

Orchestra

Orchestra[6] ist ein Peer-to-Peer-System für den Datenaustausch. Um die Ursprünge ausgetauschter Daten nachvollziehen zu können, implementiert Orchestra Why-Provenance auf Tupelebene. Weiter bietet Orchestra dies hierfür einen einsehbaren Graph, welcher die Ableitung der How-Provenance erlaubt. Das System setzt hierbei auf einem Datenbanksystem auf und erlaubt verteilte Updates auf Peers mit unterschiedlichen Schemata durch eine Form des Query-Rewriting.

Perm

Perm[7] ist ein auf PostgreSQL aufsetzendes Tool für Query-Rewriting von SQL-Anfragen. Dabei kann die Provenance von Ergebnissen nachvollzogen werden, indem die Anfrage intern umgeschrieben wird. Dabei werden künstliche Attribute in die Anfrage eingefügt, welche die ursprünglichen Attributwerte der eingeflossenen Tupel mit ausgeben.

ProvSQL

ProvSQL[8] setzt wie Perm auf PostgreSQL auf und führt How-Provenance mittels Provenance-Polynomen ein. Hierfür werden Tabellen, die dafür zur Verfügung stehen sollen, um eine Spalte für Tupel-Identifier in Form von UUIDs erweitert. Durch diese UUIDs und interne Funktionen lassen sich die Provenance-Polynome direkt bei der Anfrage mit ausgeben.

Vergleich

Die oben vorgestellten Provenance-Tools werden hier entsprechend ihrer unterstützten Funktionalitäten verglichen. Gemeinsamkeiten und Unterschiede sind der nachfolgenden Tabelle zu entnehmen. Punkte, die gesondert betrachtet werden, sind mit einem $\star$ markiert und werden anschließend diskutiert.

	Orchestra	**Perm**	**ProvSQL**	**ChaTEAU**
Provenance	HOW	HOW	HOW	$\star_1$
Eingabeformat	SQL	SQL	SQL	XML
Ausgabeformat	Graph	Tabelle	Tabelle, Graph	XML
Selektion	×	✓	✓	$\star_2$
Aggregation	×	$\star_3$	×	×

Tabelle 3.2: Vergleich der Provenance-Tools

$\star_1$ Da ChaTEAU erst mit dieser Arbeit um eine BACKCHASE-Phase erweitert wird, unterstützt ChaTEAU noch keine Provenance. In dieser Arbeit soll ein Ansatz geschaffen werden, um diejenigen Tupel zu identifizieren, welche nicht aus der aktuellen Schemaversion rekonstruierbar sind, sodass irrelevante Daten nicht gespeichert werden müssen. Die Why-Provenance auf Tupelebene wird in dieser Arbeit durch die Teilimplementation in Kapitel 5 umgangen, indem die TIDs direkt in Anfrage und Ergebnisschema eingebettet werden. Es findet sich bei der Invertierung von TGDs ein Ansatz für die Why-Provenance auf Attributebene. Hier werden Attribute, die in der Why-Provenance einer Anfrage, aber nicht im Anfrageergebnis vorkommen, markiert. Diese Markierungen können nach einem Evolutionsschritt genutzt werden, um zu überprüfen, ob alle für die Anfrage relevanten Attribute erhalten bleiben. Diese Ansätze könnten später genutzt werden, um Why-Provenance zu unterstützen.

[6] Quellcode: code.google.com/archive/p/penn-orchestra/
[7] Projektseite: www.cs.iit.edu/ dbgroup/projects/perm.html
[8] Projektseite: github.com/PierreSenellart/provsql

- $\star_2$ Da ChaTEAU sich des CHASE bedient und sich der Kontext dieser Arbeit auf konjunktive Anfragen beschränkt, unterstützt ChaTEAU ausschließlich die Selektion von Attributen auf Konstanten und von Attributen auf Attribute (Joins).
- $\star_3$ Aggregatfunktionen sind von der Provenance in Perm nicht ausgeschlossen, führen aber zu Fehlern bzw. unerwünschten Effekten. Da pro Ergebnistupel die Ursprungswerte dessen angegeben werden, tauchen Werte, welche durch Gruppierung zusammengefasst werden sollten, mehrfach auf.

3.2 Stand der Forschung

In diesem Abschnitt werden die BACKCHASE-Varianten für die Anwendungsszenarien Answering Queries using Views, die Anfrageoptimierung und den Data-Provenance-Fall vorgestellt. Zudem werden einige Verbesserungen, die während der CHASE- oder BACKCHASE-Phase angewandt werden können, für die entsprechenden Anwendungsfälle präsentiert.

3.2.1 Anfrageoptimierung

Bei der Anfrageoptimierung wird bei gegebener Anfrage Q und gegebener Menge von Integritätsbedingungen und physischer Strukturen C eine Anfrage Q' gesucht, die unter C optimiert wurde. Dies kann sowohl durch eine Form der semantischen Optimierung von Anfragen motiviert sein, aber auch durch Schemaevolution begründet sein [DPT06]. Letzterer Anwendungsfall entsteht aus der Notwendigkeit, Anfragen gegen ein Schema S_{t+1} stellen zu können, auch wenn die angefragten Daten noch nicht in dieses Schema migriert wurden und sich somit noch im Schema S_t befinden.

CHASE-Phase

Wir chasen hier Q mit C zu einem Universalplan $U = CHASE_C(Q)$, welcher alle für die Anfragen relevanten Relationen und physischen Strukturen, wie z. B. Indexe, einführt. Der Universalplan vereinigt also alle möglichen alternativen Wege, Q unter den Integritätsbedingungen C zu beantworten. Damit Indexe im Anfrageplan berücksichtigt werden können, müssen wir für diese besondere TGDs einführen. Damit sie auch im Universalplan vorkommen, müssen Indexe eigene relationale Atome erhalten. Damit kein irrelevanter Index im CHASE-Ergebnis auftaucht, wird die Verwendung der entsprechenden Basisrelation vorausgesetzt. Daraus ergibt sich für einen beliebigen eindimensionalen Index die TGD

$$r(x, \vec{y}) \rightarrow i_R(x).$$

Damit diese ihre Basisrelationen dort ersetzen können, wo nur das Attribut benötigt wird, über welchem der Index angelegt wurde, muss eine weitere TGD

$$i_R(x) \rightarrow \exists \vec{Y} r(x, \vec{Y})$$

eingeführt werden. Die Nutzung des Index beim Zugriff auf die Basisrelation kann hier nicht abgebildet werden. Wie materialisierte Sichten bei der Anfrageoptimierung mit dem CHASE berücksichtigt werden können, wird im Abschnitt 3.2.2 diskutiert.

BACKCHASE-Phase

In der BACKCHASE-Phase werden aus allen Teilmengen U_i von U jene gesucht, für welche gilt, dass

$$U_i \equiv_C Q,$$
$$\nexists U_j : U_j \subset U_i \wedge U_j \equiv_C Q.$$

Für den Äquivalenztest unter C genügt es eine Abbildung zu finden, unter welcher q in $CHASE(U_i)$ enthalten ist [DPT06]. Hervorzuheben ist, dass damit alle minimalen Rewritings der Ursprungsanfrage gefunden werden, damit jedoch noch keine Auswahl der optimalen Anfrage getroffen wurde. Dies muss über ein Kostenmodell passieren, welches beispielsweise Anzahl der Tupel pro Relationen oder die Zahl der Verbundoperationen einbezieht. Der CHASE kann hier keine Auswahl basierend auf Zugriffskosten treffen. Besonders muss hier auf Indexe geachtet werden, da diese Einfluss auf die Kosten der Zugriffe ihrer entsprechenden Basisrelationen haben. Durch die BACKCHASE-Phase werden im Regelfall Teilmengen, welche sowohl Index als auch Basisrelation enthalten, als nicht-minimal gekennzeichnet. Dies ist sinnvoll, da Index und Basisrelation nicht durch eine Verbundoperation verknüpft werden. Das Kostenmodell sollte dies jedoch reflektieren und die Ausnutzung des Index bei Verwendung der Basisrelation einkalkulieren.

Optimierung

Eine vollständige Überprüfung des Suchraumes ist während der BACKCHASE-Phase nicht nötig. Da nur minimale Rewritings von Interesse sind, kann die Suche *bottom-up* durchgeführt werden. Beginnt man mit jenen Teilmengen, die nur eine atomare Formel enthalten, dann alle zweielementigen, usw., werden die minimalen Rewritings zuerst gefunden. Alle Obermengen dieser, können dann aus dem Suchraum entfernt werden, da sie nicht mehr minimal sein können (*Pruning*).

3.2.2 Answering Queries using Views

Answering Queries using Views (AQuV) ist ein Problem, welches bei gegebener Anfrage Q und gegebener Menge von Sichtdefinitionen V eine zu Q äquivalente Anfrage Q^V sucht, welche Sichten verwendet. Allgemeine Integritätsbedingungen, die auf der Datenbank gelten, können ebenfalls eingearbeitet werden. Im Folgenden werden diese der Einfachheit halber allerdings vernachlässigt. Es gibt grundlegend zwei verschiedene Ansätze für das AQuV-Problem. Diese folgen direkt daraus, was unter einer Anfrage, welche Sichten verwendet, verstanden wird.
Zum einen kann von Q^V gefordert werden, dass sie ausschließlich Sichten verwendet. Dies kann zum Beispiel durch den Datenschutz motiviert sein.

> **Beispiel:** AQuV – Datenschutz
> An einer Universität sollen die Mitarbeiter der Studienbüros, welche Noten eintragen und einsehen, keine persönlichen Daten der Studenten sehen können. Dazu wurden Sichten angelegt, aus welchen die persönlichen Daten der Studenten entfernt wurden.
> Eine Anfrage kann dann auf natürliche Art gegen die Basisrelationen gestellt werden, soll aber vorher intern zu einer äquivalenten Anfrage umgeschrieben werden, die auf den Sichten arbeitet.
> Ist für die Anfrage ein Attribut wichtig, welches sensible Daten enthält, schlägt das Rewriting der Anfrage fehl. Der Nutzer erfährt dann, dass die Befugnisse für diese Anfrage nicht vorliegen.

Auf der anderen Seite, kann AQuV auch so verstanden werden, dass möglichst viele Sichten durch die Anfrage Q^V verwendet werden sollen. Dabei soll aber nicht die Minimalität von Q^V vernachlässigt werden. Es dürfen also keine überflüssigen Sichten im Rewriting vorkommen. Dies kann zum Beispiel in Data Warehouses Einsatz finden.

> **Beispiel:** AQuV – Data Warehouse
> Da sich Daten in Data Warehouses selten ändern, lohnt es sich oft, bestimmte Ergebnisse als materialisierte Sichten zu speichern, um häufig verwendete Auswertungen effizienter zu gestalten. So kann das immer neue Ausführen der Anfrage durch einen Relationenscan der Ergebnissicht ersetzt werden.
> Im Fall komplexerer Auswertungen kann es daher wünschenswert sein, möglichst viele dieser materia-

lisierten Sichten für eine Anfrage zu verwenden. Hierbei ist aber die Verwendung von Basisrelationen nicht verboten, sollte das Rewriting sonst unmöglich sein.

Die folgenden theoretischen Ausführungen stützen sich auf [Ile14] und [DH13].

CHASE-Phase

Zum Finden solcher Rewritings lassen sich CHASE und BACKCHASE verwenden. Dazu werden die Sichtdefinitionen als Menge von TGDs V formuliert. Mittels des CHASE wird die ursprüngliche Anfrage Q um die Sichtdefinitionen erweitert und zu einem Universalplan U aufgebläht. Hierzu wird $CHASE_V(Q) = U$ berechnet. U enthält dann neben den Basisrelationen, welche in der Anfrage enthalten sind, auch alle Sichten aus V, die durch Relationen aus Q oder bereits aufgenommene Sichten gebildet werden können. Der Universalplan beinhaltet demnach alle für die Anfrage relevanten Relationen und Sichten.

BACKCHASE-Phase

Sollen ausschließlich Sichten für das Rewriting benutzt werden, so kann der Universalplan anschließend auf die darin enthaltenen Sichten eingeschränkt werden, da die Basisrelationen nicht verwendet werden dürfen. Dies ist der erste Schritt der BACKCHASE-Phase. Weiter müssen die Sichtdefinitionen in der BACKCHASE-Phase invertiert werden. Hierbei muss besonders darauf geachtet werden, dass beim Invertieren von TGDs Attribute, welche in der Basisrelation aber nicht in der Sicht enthalten sind, existenzquantifiziert sein müssen. Die Menge der invertierten TGDs V^{-1} drückt aus, welche Sichten (teilweise) welche Basisrelation beinhalten und somit für die Beantwortung der Anfrage genutzt werden können. Die Invertierung einer Sichtdefinition funktioniert wie folgt:

$$f : \forall \vec{x}, \vec{y} : r(\vec{x}, \vec{y}) \rightarrow v(\vec{x})$$
$$\Downarrow$$
$$f^{-1} : \forall \vec{x} : v(\vec{x}) \rightarrow \exists \vec{Y} : r(\vec{x}, \vec{Y})$$

Zuletzt wird für jede Teilmenge U_i des Universalplans überprüft, ob diese ein äquivalentes Rewriting zu Q ist. Dazu wird U_i mit V^{-1} gechaset. Ist das Ergebnis $CHASE_{V^{-1}}(U_i)$ Obermenge von Q und minimal (keine Basisrelation oder Sicht kommt doppelt im CHASE-Ergebnis vor), ist U_i ein Rewriting für die Anfrage Q, welches nur Sichten verwendet. Möchte man hingegen möglichst viele Sichten verwenden, muss die Suche nach Q^V in U angepasst werden. Hier ist das Finden eines maximal-enthaltenen Rewritings nützlich [Hal00]. Dies kann ähnlich wie beim zuvor beschriebenen Ansatz in der BACKCHASE-Phase ermittelt werden, da ein minimales und äquivalentes Rewriting per Definition bereits *maximal-enthalten* ist. Genügt keine Sichtkombination der Äquivalenz zu Q, so können maximal-enthaltene Rewritings um die Basisrelationen erweitert werden, welche für die Äquivalenz zu Q fehlen.

Beispiel: AQuV – CHASE und BACKCHASE
Gegeben seien die oben definierten Relationen für Studenten und deren Noten, sowie eine Sicht auf die Noten von Informatikstudenten. Weiter existiert eine TGD f, welche diese Sicht definiert. Eine Anfrage Q, die für alle Informatikstudenten ihre Matrikelnummer und Noten in den entsprechenden Modulen ausgibt, soll nur auf Sichten ausgeführt werden. Grund dafür ist, dass aus Datenschutz-

gründen der Lesezugriff auf die Basisrelationen eingeschränkt wurde. Gegeben sind folglich:

$$r_{\text{Student}}(ma, st, vn, nn),$$
$$r_{\text{Note}}(ma, mo, no),$$
$$v_{\text{NotenInf}}(ma, mo, no),$$
$$f : \forall ma, vn, nn, mo, no : r_{\text{Student}}(ma, \text{'INF'}, vn, nn) \wedge r_{\text{Note}}(ma, mo, no) \rightarrow v_{\text{NotenInf}}(ma, mo, no),$$
$$Q(ma, mo, no) \text{ :- } r_{\text{Student}}(ma, \text{'INF'}, Vn, Nn) \wedge r_{\text{Note}}(ma, mo, no).$$

In der CHASE-Phase wird $CHASE_V(Q)$ berechnet. Dabei ist V die Menge aller TGDs, welche Sichten definieren. In diesem Beispiel liegt nur eine einzige TGD vor. Da $CHASE_V(Q)$ auch Q selbst enthält, existiert ein aktiver Trigger für f.

$$CHASE_V(Q) = r_{\text{Student}}(ma, \text{'INF'}, Vn, Nn), r_{\text{Note}}(ma, mo, no)$$
$$\Downarrow f$$
$$CHASE_V(Q) = r_{\text{Student}}(ma, \text{'INF'}, Vn, Nn), r_{\text{Note}}(ma, mo, no), v_{\text{NotenInf}}(ma, mo, no)$$

Da keine weiteren Trigger existieren und durch den Kopf von f keine neuen erzeugt werden, ist die CHASE-Phase hiermit abgeschlossen und der Universalplan

$$U = r_{\text{Student}}(ma, \text{'INF'}, Vn, Nn), r_{\text{Note}}(ma, mo, no), v_{\text{NotenInf}}(ma, mo, no)$$

gefunden. Anschließend kann der Universalplan auf die Sichten eingeschränkt und die Sichtdefinitionen invertiert werden. Daraus ergibt sich für die BACKCHASE-Phase

$$U = v_{\text{NotenInf}}(ma, mo, no),$$
$$f^{-1} : \forall ma, mo, no : v_{\text{NotenInf}}(ma, mo, no) \rightarrow \exists Vn, Nn : r_{\text{Student}}(ma, \text{'INF'}, Vn, Nn)$$
$$\wedge r_{\text{Note}}(ma, mo, no).$$

Anschließend wird in U nach Teilmengen gesucht, die Q enthalten. In diesem Fall haben wir nur ein Element in U. Da aber nicht der gesamte Universalplan gechaset wird, nennen wir die Teilmenge, welche nur $v_{\text{NotenInf}}(ma, mo, no)$ enthält U_1. Diese Teilmenge von U erfüllt den Rumpf von f^{-1}, aber nicht den Kopf der Abhängigkeit. Folglich liegt hier ein aktiver Trigger vor, sodass sich

$$CHASE_{V^{-1}}(U_1) = v_{\text{NotenInf}}(ma, mo, no), r_{\text{Student}}(ma, \text{'INF'}, Vn, Nn), r_{\text{Note}}(ma, mo, no)$$

ergibt. Offensichtlich ist Q Teilmenge von $CHASE_{V^{-1}}(U_1)$ und $Q_V(ma, mo, no)$:- $v_{\text{NotenInf}}(ma, mo, no)$ somit ein minimales Rewriting für Q.

Optimierung

Das Suchen äquivalenter Teilmengen kann optimiert werden, indem man den Pruning-Ansatz verwendet, um die Teilmengen von U zu überprüfen. Dadurch werden keine nicht-minimalen Kandidaten für Rewritings überprüft.
Weiter wird in [Ile14] und [DH13] ein Ansatz vorgestellt, welcher das Berechnen von $CHASE_{V^{-1}}(U_i)$ für U_i verhindert, die kein äquivalentes Rewriting für Q liefern. Hierzu werden nicht alle Teilmengen von U mit V^{-1} gechaset sondern der auf Sichten eingeschränkte Universalplan. Um aber aus dem CHASE-Ergebnis $CHASE_{V^{-1}}(U) = U^V$ die für das Rewriting verwendbaren Sichten zu rekonstruieren, werden bei der Berechnung Provenance-Annotationen für jedes Atom im CHASE-Ergebnis angefügt. Diese Annotationen beschreiben die Zeugenbasis der einzelnen Atome, also den Rumpf der invertierten Abhängigkeit, die zur Aufnahme des Atoms in U^V führte. Ähnlich wie beim Standardansatz werden anschließend

Containment-Mappings gesucht, also Abbildungen h_i von Q in U^V, die zeigen, dass $h_i(Q) \subseteq U^V$ gilt. Für solche Abbildungen werden dann die Zeugenbasen der entsprechenden Atome in U^V zurückgegeben. Diese Sichten bilden dann für h_i die modifizierte Anfrage Q^V.

3.2.3 Provenance

Ein wichtiges Kriterium bei wissenschaftlichen Auswertungen ist die Reproduzierbarkeit dieser. Dies wird aber durch Änderungen am Schema für neue Auswertungen erschwert. Weiter nehmen Daten für große Analysen viel Platz ein, sodass nicht alle Daten aufgehoben werden sollen oder können. Teils ist dies nicht einmal nötig, weil ganze Datensätze oder bestimmte Attribute nicht an der Auswertung beteiligt waren. In [AH19] wurden die bislang einzigen Ansätze vorgestellt, um den CHASE für diesen Anwendungsfall einzusetzen. Dort wurde allerdings bisher nur eine grundlegende Idee dafür aufgezeigt, welche im Folgenden aufgegriffen und erweitert wird. Ziel ist es basierend auf einem Anfrageergebnis K bzw. einem Teil dessen K' und der entsprechenden Anfrage Q eine minimale Teilinstanz I' von I zu berechnen, welche persistent gespeichert werden muss, um die Auswertung reproduzierbar zu halten. I ist dabei die ursprüngliche Instanz, auf welcher die Anfrage Q zum Ergebnis K geführt hat.

CHASE-Phase

Der CHASE dient hier zur Berechnung des Anfrageergebnisses. Die Reproduzierbarkeit kann also in einem Prozess mit der Auswertung sichergestellt werden. Hierzu wird die Anfrage als s-t-TGD f formuliert, deren Rumpf auf dem Schema S_1 definiert ist, über welchem auch I existiert. Der Kopf dieser Abhängigkeit ist über dem Schema des Anfrageergebnisses S_2 definiert. Der CHASE-Parameter für diese Phase ist ein Schema-Mapping $M = (S_1, S_2, \Sigma)$, wobei Σ die Menge der geltenden Abhängigkeiten inklusive f ist. Da für die BACKCHASE-Phase die Auswertung invertiert werden muss und solche Auswertungen meist verlustbehaftet sind, können Provenance-Annotationen verwendet werden. Dazu werden für die Tupel aus I TIDs vergeben. Mit einem Why- oder How-Provenance-Ansatz kann die Invertierung verbessert werden. Dabei werden folgende Klassen von CHASE-Inversen unterschieden [AH18b] [FKPT11]:

- **exakte CHASE-Inverse:** Durch die Inverse ist die gesamte Originalinstanz wiederherstellbar ($I' = I$).
- **klassische CHASE-Inverse:** Die durch die Inverse gebildete Instanz I' ist unter dem CHASE äquivalent zur Originalinstanz. Das heißt, es existiert ein Homomorphismus, welcher I' auf I abbildet und umgekehrt ($I' \equiv I$).
- **tupelerhaltend-relaxierte CHASE-Inverse:** Eine Teilinstanz der Originaldatenbank mit gleicher Tupelzahl kann durch die Inverse rekonstruiert werden ($I' \preceq I$, $|I'| = |I|$).
- **relaxierte CHASE-Inverse:** Die Inverse kann einen Teil der Originalinstanz wiederherstellen ($I' \preceq I$).
- **ergebnisäquivalente CHASE-Inverse:** Für die rekonstruierte Instanz ist bezüglich des CHASE mit M gleich ($CHASE_M(I') = CHASE_M(I)$).

Hierbei bedeutet $I' \preceq I$, dass die Teilinstanz I' weniger Informationen enthält als I, da sie z. B. weniger Attribute oder mehr Nullwerte besitzt. Weiter wird in [AH18b] tabellarisch erfasst, welche Operationen der Relationenalgebra welche Inversentypen haben und wie der Einbezug der Provenance diese Inversentypen beeinflusst.

BACKCHASE-Phase

In der BACKCHASE-Phase muss die Anfrage in TGD-Form invertiert werden, um aus den Ergebnistupeln einen Teil der Originalinstanz zu berechnen. Mittel Zeugenbasen für die Ergebnistupel oder dem

Provenance-Polynom der Anfrage, soll eine möglichst genaue CHASE-Inverse gebildet werden. Offensichtlich können Informationen, welche nicht im Ergebnisschema vorliegen und auch nicht zweifelsfrei aus der Anfrage erkennbar sind, nicht wiederhergestellt werden. Allerdings können mittels der Provenance-Annotationen deutliche Verbesserungen erreicht werden [AH18b].

Beispiel: Minimalbeispiel CHASE-Inverse mit TIDs
Betrachten wir zur Veranschaulichung folgendes Minimalbeispiel mit den Relationen $r_1(A, B)$ und $r_2(B, C)$, welche unsere Instanz I bilden, sowie der Anfrage $Q(a, c)$:- $r_1(a, B), r_2(B, c)$. Diese definieren wir als s-t-TGD

$$q : r_1(a, b) \wedge r_2(b, c) \rightarrow v_Q(a, c).$$

Wir führen zunächst $CHASE_q(I)$ aus, um das Anfrageergebnis v_Q zu berechnen.

A	B
1	1
1	2
2	2

$\bowtie$

B	C
1	2
2	1

$\xRightarrow{CHASE_q(I)}$

A	C
1	2
1	1
2	1

Invertieren wir nun die Anfrage Q, ergibt sich die s-t-TGD

$$q^{-1} : v_Q(a, c) \rightarrow \exists B : v_{r_1}(a, B) \wedge v_{r_2}(B, c).$$

v_{r_1} und v_{r_2} repräsentieren hier die zu rekonstruierende Teilinstanz der Datenbank. Führen wir nun $CHASE_{q^{-1}}(v_Q)$ aus, um eine Teildatenbank zu rekonstruieren, erhalten wir folgende rekonstruierte Instanz I', bestehend aus $v_{r_1}(A, B)$ und $v_{r_2}(B, C)$, mit nummerierten Nullwerten η_i:

A	C
1	2
1	1
2	1

$\xRightarrow{CHASE_{q^{-1}}(I)}$,

A	B
1	η_1
1	η_2
2	η_3

B	C
η_1	2
η_2	1
η_3	1

Dass es sich hierbei um eine relaxierte CHASE-Inverse handelt, lässt sich zeigen, indem man η_1 auf 1, η_2 auf 2 und η_3 auf 2 abbildet. Offensichtlich handelt es sich aber um keine tupelerhaltend-relaxierte CHASE-Inverse, da I' mehr Tupel enthält.
Sind die Zeugenbasen der Tupel aus dem Anfrageergebnis bekannt, lässt sich dies verbessern. Betrachtet man diese, fällt auf, dass die Tupel $(2, 1)$ und $(1, 1)$ aus v_Q ein gemeinsames Tupel in ihrer Zeugenbasis haben ($(2, 1)$ aus r_2). Mit dieser Information fallen die rekonstruierten Tupel aus v_{r_2} $(\eta_2, 1)$ und $(\eta_3, 1)$ zusammen. Auch das η_3 in v_{r_1} wird durch ein η_2 ersetzt.
Die mithilfe der Provenance-Informationen rekonstruierten Relationen sind dann:

A	B
1	η_1
1	η_2
2	η_2

B	C
η_1	2
η_2	1

Dies entspricht einer tupelerhaltend-relaxierten CHASE-Inversen.

Mit dieser Rekonstruktion der Ursprungsdaten ist das eigentliche Ziel des Provenance-Managements aber noch nicht erreicht. Das Ziel ist, herauszufinden welche der Ergebnistupel für die Reproduzierbarkeit der Auswertung persistent gespeichert werden müssen, weil sie nicht mehr rekonstruierbar sind. Besonders relevant ist hierbei die mögliche Evolution des Schemas nach Ausführung der Anfrage [AH19]. Dabei können ebenfalls Informationen verloren gehen. Das eigentliche Resultat des Provenance-Managements

ist also eine Teilinstanz I_{Prov}, welche diejenigen Tupel aus I' im Schema S_n (Schema zur Zeit der Auswertung) sind, die nicht aus I im Schema S_{n+1} (aktuelles Datenbankschema nach Evolution) rekonstruiert werden können.
Wie genau der CHASE&BACKCHASE-Ansatz hierfür verwendet werden kann und wie die entsprechenden Provenance-Annotationen in diesen einfließen können, wird in Kapitel 4 diskutiert.

Kapitel 4

Konzept

In diesem Kapitel wird vorgestellt, wie es mittels des CHASE möglich ist, Daten aus Anfragen zu rekonstruieren. Weiter wird darauf eingegangen, wie dies in Kombination mit inversen Schemaabbildungen genutzt werden kann, um die Zahl der zu archivierenden Daten zu reduzieren.

4.1 Das Problem im Detail

Für wissenschaftliche Auswertungen ist die Reproduzierbarkeit der Ergebnisse ein wichtiges Qualitätsmerkmal und ausschlaggebend für die Glaubwürdigkeit von Statistiken und Forschungsergebnissen. Dies führt allerdings ohne weitere Maßnahmen dazu, dass eine große Menge von Daten persistent gespeichert werden muss. Dies ist vor allem problematisch, wenn sich die Struktur der Daten ändert. Dies ist nicht untypisch für Datenbanken, die Forschungsdaten speichern. Grund dafür können beispielsweise neue Standards oder Untersuchungen anderer Eigenschaften der erhobenen Daten sein. Diese Schemaevolutionen können dazu führen, dass alte Anfragen nicht mehr auf neuere Schemata passen und die Daten der Anfrage nicht mehr klar aus dem neuen Schema erkennbar sind. Folglich müssten sehr viele alte Daten dauerhaft aufgehoben werden, um eine Anfrage reproduzierbar zu halten.
Ziel ist hier die Entwicklung eines Ansatzes, welcher die Berechnung der Tupel erlaubt, welche nicht mehr aus dem neuen Schema rekonstruierbar sind. Voraussetzung dafür ist, dass alle Tupel der Datenbank eine TID besitzen und Evolutionsschritte sowie zu reproduzierende Anfragen gespeichert werden. Dadurch können nach einem Evolutionsschritt zur nächsten Schemaversion alle Auswertungen, für welche Reproduzierbarkeit gefordert wird, durchgeführt werden und alle Tupel, die dafür zwingend erhalten bleiben müssen, berechnet werden. Die übrigen Tupel sind demnach nicht mehr für bisherige Auswertung nötig und müssen nicht zwingend aufbewahrt werden.
Der CHASE findet hier zweifach Anwendung. Zum einen wird er in der CHASE-Phase genutzt, um die Anfrage bzw. die Evolution auszuführen, und somit das Anfrageergebnis bzw. den Datenbankzustand nach der Evolution zu berechnen. Zum anderen verwendet die BACKCHASE-Phase für diesen Ansatz ebenfalls den CHASE-Algorithmus. Dieser wird zur Berechnung von Inversen benutzt, um die für die Anfrage relevante Teilmenge der Daten zu berechnen, oder die aktuelle Datenbank anhand der gegebenen Evolutionsoperationen in eine alte Schemaversion zu transformieren.

4.2 Sicherung der Voraussetzungen

Damit der CHASE die Evolutionsschritte und Anfragen verarbeiten kann, müssen diese in Form von Abhängigkeiten eingepflegt werden. Die Schemaevolution wird als s-t-TGDs festgehalten. Das Quellschema ist der alte Datenbestand und das Zielschema die Datenbank nach der Evolution. Weiter müssen ggf. Funktionen für Teile der Schemaabbildung erhalten werden. Diese sind dann wichtig, wenn die Evolution

auch Werte verändert. Dies kann beispielsweise der Fall sein, wenn neue Standards oder Ähnliches eingeführt werden. Wichtig ist, dass diese Funktionen jeweils eine Inverse besitzen, welche ebenfalls erhalten bleiben muss.

Beispiel: TGDs mit Funktionen
Man stelle sich zur Veranschaulichung folgenden Sachverhalt vor:
Einige Lehrstühle des Instituts für Informatik haben sich dazu entschieden gemeinsam ein neues Institut zu gründen. Veranstaltungen des neuen Institut für Visual and Analytic Computing waren somit vor der Evolution im Katalog des Institut für Informatik. Weiter sollen nun auch die Semesterwochenstunden (SWS) einer Veranstaltung gespeichert werden. Geschieht die Kennzeichnung des Instituts nicht durch ein gesondertes Attribut, sondern ist beispielsweise Teil der Modulnummer, ist es unpraktikabel für jede Veranstaltung eine einzelne s-t-TGD anzugeben. Anstelle dessen könnte eine Funktion und eine einzelne TGD dies übernehmen. Wichtig ist dabei aber, dass zu jeder solchen Funktion gleichzeitig eine inverse Funktion existiert, die ebenfalls gespeichert wird. Gegeben sei die Funktion $INST(mo)$, welche die Modulnummer mo mit geändertem Institutspräfix zurückgibt, falls sich die Zugehörigkeit zum Institut mit der Evolution ändert. Weiter existiere die Funktion $INST^{-1}(mo)$ welche die Modulnummer mit ihrem Präfix vor der Evolution zurückgibt. Dann erfüllen folgende TGD E_1 und ihre inverse E_1^{-1} die oben beschriebenen Evolutionsoperationen

$$E_1 : \forall mo, dozent : Veranstaltung(mo, dozent) \rightarrow \exists Sws : Veranstaltung(INST(mo), dozent, Sws),$$
$$E_1^{-1} : \forall mo, dozen, sws : Veranstaltung(mo, dozent, sws) \rightarrow Veranstaltung(INST^{-1}(mo), dozent).$$

TIDs sind essenziell, um die Why-Provenance zu benutzen. Die Zeugenbasen der Ergebnistupel erlauben eine genauere Rekonstruktion der ursprünglichen Tupel, welche die Basis der Berechnung zwingend zu erhaltener Tupel darstellen. TIDs werden als zusätzliche Attribute in allen Relationen eingefügt und sind global im gesamten Schema eindeutig. So können anhand der TIDs nicht nur Tupel innerhalb einer Relation unterschieden werden, sondern sie sind gleichzeitig ihrer entsprechenden Tabelle zuordenbar. Eine Schemaänderung kann allerdings auch Einfluss auf die Tupel der sich entwickelnden Datenbank haben, z. B. wenn ein neues Attribut eingefügt wird, welches mehrwertig durch den Schlüssel der Relation bestimmt wird. Gleichzeitig brauchen aber auch alle Tupel der aktuellen Schemaversion ihre eigenen Tuple-Identifier. Das bedeutet wir benötigen ein invertierbares Mapping, welches die TIDs von S auf die TIDs von S' abbildet. Dafür muss bei jedem Evolutionsschritt, wenn die TIDs für Tupel im neuen Schema vergeben werden, das Mapping um alle Abbildungen von TIDs aus S auf TIDs in S' erweitert werden. Diese Abbildung kann als *Dictionary* gespeichert werden. Bei der Invertierung der Evolution, um alte Tupel zu berechnen, können die TIDs für S durch einen *lookup* im Dictionary gefunden werden. Dann können durch Cleaning-EGDs, welche die Gleichheit aller Tupel mit gleicher TID erzwingen, die ursprünglichen Tupel wieder hergestellt werden.

Beispiel: Minimalbeispiel TID-Dictionary
Gegeben sei die Relation $r_1(tid, x, y)$, welche unsere Instanz I im Schema S bildet. Weiter existiere eine s-t-TGD

$$E : \forall tid, x, y : r_1(tid, x, y) \rightarrow \exists TID_X, TID_Y : r_2(TID_X, x) \wedge r_3(TID_Y, y),$$

welche die Evolution in ein neues Schema S' beschreibt. Auch wenn die TIDs für r_2 und r_3 hier Nullwerte ergeben, nehmen wir an, dass sie fortlaufend durchnummeriert werden. Der Evolutionsschritt mit E ergibt folgendes:

TID	X	Y
S_1	1	2
S_2	1	1

$\xrightarrow{CHASE_E(I)}$

TID	X
X_1	1

TID	Y
Y_1	2
Y_2	1

Verwendet man hierfür den Standard-CHASE, stößt man auf ein Problem. Die Prüfung auf aktive Trigger führt dazu, dass die Beziehung zwischen S_2 und X_1 verloren geht. Hierfür muss der CHASE angepasst werden, sodass bei inaktiven Triggern trotzdem die entsprechenden Einträge im TID-Dictionary vorgenommen werden. Würde man das Dictionary als neue Relation einführen und E entsprechend erweitern, verlieren die relationalen Atome der rechten Seite ihre Unabhängigkeit, sodass ein zusätzlicher Nullwert für TID_X generiert werden würde. Das TID-Dictionary sieht mit dieser Anpassung wie folgt aus:

$\mathbf{TID}_S$	$\mathbf{TID}_{S'}$
S_1	X_1
S_1	Y_1
S_2	X_1
S_2	Y_2

Dazu existiere die Funktion $lookup(tid)$, welche für eine TID aus S' die TID aus S zurückgibt. Wird die s-t-TGD der Schemaevolution nun invertiert, um die alten Tupel zu berechnen, ergeben sich folgende zwei Abhängigkeiten, welche gemeinsam die Inverse zu E bilden:

$$E_1^{-1} : \forall tid, x : r_2(tid, x) \rightarrow \exists Y : r_1(lookup(tid), x, Y),$$
$$E_2^{-1} : \forall tid, y : r_3(tid, y) \rightarrow \exists X : r_1(lookup(tid), X, y).$$

Weiter wird die Cleaning-EGD

$$C : \forall tid, x_1, x_2, y_1, y_2 : r_1(tid, x_1, y_1) \wedge r_1(tid, x_2, y_2) \rightarrow x_1 = x_2 \wedge y_1 = y_2$$

benötigt, um die Gleichheit von Tupeln mit gleicher TID zu erzwingen. Wir fassen diese drei Abhängigkeiten in der Menge Σ für das Mapping (S', S, Σ) zusammen. Die Berechnung der Tupel im Schema S erfolgt dann durch den CHASE mit Σ:

TID	X
X_1	1

TID	Y
Y_1	2
Y_2	1

$\xrightarrow{CHASE_\Sigma(I)}$

TID	X	Y
S_1	1	2
S_2	1	1

Das Konzept solcher TGDs, deren Kopf Funktionen enthalten, wurden in [FKPT11] als sogenannte *second-order-TGDs* eingeführt. Diese werden im Folgenden genutzt, um solch eine lookup-Funktion zu realisieren. Weiter können sie genutzt werden, um mit den TIDs Zeugenbasen zu bilden und diese auch mit dem CHASE zu verwenden. Zeugenbasen können im allgemeinen Fall auch Alternativen enthalten, sodass hier diejenige bevorzugt werden soll, die zur Speicherung der wenigstens Tupel führt. In [FKPT11] wird neben den oben genannten second-order-TGDs auch die Möglichkeit der Komposition von Schemaevolutionen erörtert. So müssen nicht für jede Schemaversion alle Evolutionsoperationen gespeichert werden, sondern durch Kompositionen der Evolutionen ersetzt. So ließe sich von Version S'' direkt zu S zurückrechnen, ohne S' als Zwischenschritt zu betrachten. Dies ist in unserem Anwendungsfall jedoch nicht für mehrere Evolutionen praktikabel. Grund dafür ist, dass im Regelfall auf jeder Schemaversion wissenschaftliche Auswertungen liefen, die wiederum reproduzierbar sein müssen. Dementsprechend ist jede Schemaversion relevant und es muss möglich sein die aktuellen Daten in jede Schemaversion zurückzurechnen. Allerdings ist das Stellen einer alten Anfrage für S auf S' durch eine Komposition zu bewältigen [FKPT11]. Somit wird eine Anfrage Q gestellt gegen S' umgesetzt, indem die Komposition der Anfrage mit der invertierten Schemaevolution berechnet wird.

4.3 Beschreibung des Konzeptes

Die Berechnung von I_{Prov}, der Menge Daten, welche dauerhaft gespeichert werden müssen, erfolgt in mehreren Schritten. Prinzipiell ist die Reihenfolge der ersten zwei Phasen vertauschbar. Die im Folgenden vorgestellte Variante hat den Vorteil, dass nur Tupel aus $I(S')$, deren TID laut Dictionary für die Anfrage Q relevant ist, bei der Invertierung der Evolutionsoperationen betrachtet werden müssen. Führt man die Evolution zuerst aus, kann man hingegen mehrere Anfragen gleichzeitig betrachten, muss jedoch für jede Anfrage $I'(S)$ aufbewahren, bis für diese I_{Prov} berechnet wurde. Weiter müssen alle Tupel aus $I(S')$ in das Schema S zurückgerechnet werden. Letztere Variante ist aber generell vorteilhafter in einer realen Anwendung, weil sie somit vor jeder Schemaevolution ausgeführt werden kann und alle wissenschaftlichen Auswertungen abgearbeitet werden können. Danach kann $I(S)$ bis auf die I_{Prov}-Tupel der jeweiligen Anfragen gelöscht werden. Die einzelnen Phasen stützen sich hier auf die Ansätze aus [Aug17], [AH18a] und [AH19]. Diese werden dann auf mehrere Anfragen und Evolutionsschritte ausgeweitet und in die Berechnung von I_{Prov} eingepflegt.

4.3.1 Bildung der Inversen

In [AH18b] wurde vorgestellt, welche Form der Inverse durch welche Operatoren bezüglich der Originalinstanz erreicht wird. Die Operatoren konjunktive Anfragen sind jedoch eingeschränkt, da sich beispielsweise keine Aggregatfunktionen mit diesen abbilden lassen. Wir betrachten somit hier nur Selektion, Projektion, Verbund und Umbenennung. Für die Berechnung von I_{Prov} ist eine relaxierte CHASE-Inverse Voraussetzung, da eine Teilinstanz von I berechnet werden soll. Weiter betrachten wir nicht nur die Inverse bezüglich der Originalinstanz, sondern sind auch am Inversentyp in Bezug auf die an der Anfrage bzw. Evolution beteiligten Tupel interessiert. Diesen wird untersucht, da wir explizit nach den Tupeln suchen, welche für die Reproduzierbarkeit einer Anfrage wichtig sind. Je nach Anwendungsfall kann daher Interesse bestehen, nur die minimale Menge an Tupeln für die Reproduktion der Auswertung zu erhalten (Ergebnisäquivalenz nach Evolution) oder alle an der Anfrage beteiligten Tupel zu rekonstruieren. Für Ersteres reicht die relaxierte CHASE-Inverse aus, Letzteres verlangt hingegen eine tupelerhaltend-relaxierte Inverse. Tupel, die nie Teil einer Auswertung sind, werden somit nicht betrachtet. Im folgenden Stellen wir vor, wie solche Anfragen und Evolutionen in ChaTEAU und dem hier vorgestellten Konzept umgesetzt werden können. Dabei sei R eine Relation aus dem Schema S in der Instanz I, c eine Konstante, A_i Attribute der Relation R und *Ergebnis* die Ergebnisrelation der Anfrage bzw. der Evolution.

Selektion

Als s-t-TGD lässt sich die Selektion $A_j = c$ als

$$R(A_1, A_2, ..., c, ..., A_n) \rightarrow Ergebnis(A_1, A_2, ..., c, ..., A_n)$$

formulieren. Bei der Selektion auf Attribute lässt sich die Inverse einfach berechnen, da ohne weitere Projektion alle Attribute im Ergebnisvorkommen. Bezüglich der Originalinstanz I ist die Inverse folglich relaxiert, da nur die Tupel mit $A_j = c$ durch diese rekonstruiert werden können [AH18b]. Bezüglich der Tupel, die an der Anfrage beteiligt sind, ist hier sogar eine tupelerhaltend-relaxierte Inverse möglich, falls der Schlüssel der Relation nicht aus dem Anfrageergebnis herausprojiziert wird. Die Inverse lässt sich somit ohne Zusatzinformationen als s-t-TGD

$$Ergebnis(A_1, A_2, ..., c, ..., A_n) \rightarrow R(A_1, A_2, ..., c, ..., A_n)$$

formulieren. Die Selektion gefährdet jedoch die Reproduzierbarkeit einer Anfrage nach der Schmeaevolution. Wird durch die Evolution das Selektionsattribut herausprojiziert, werden Zusatzinformationen benötigt. Hier muss zu jedem Tupel das entsprechende Attribut rekonstruiert werden können, z. B. indem die betroffenen Tupel gesondert gespeichert werden, oder indem für jedes Tupel das besagte Selektionsattribut zusammen mit einem Schlüssel aufgehoben wird. Gleiches gilt, wenn die Evolution eine Selektion

enthält. Wird hier auf dem gleichen Attribut selektiert wie in der Anfrage, müssen die von der Anfrage betroffenen Tupel aufgehoben werden, falls die Selektionskonstante der Anfrage nicht mit der Konstante der Evolution übereinstimmt, da sie sonst bei der Schemaevolution aus dem Datenbestand herausfallen. Ist die Konstante jedoch gleich, werden keine Zusatzinformationen benötigt. Ist das Selektionskriterium der Evolution über einem anderen Attribut definiert als in der Anfrage, kann mit der Evolution berechnet werden, welche Tupel aufgehoben werden müssen (siehe 4.3.4).

Projektion

Die Projektion lässt sich als s-t-TGD durch

$$R(A_1, A_2, ...A_n) \rightarrow Ergebnis(A_2, ..., A_n)$$

repräsentieren. Da hier alle Tupel auf I das Muster des Rumpfes erfüllen, muss hier nicht zwischen Inversentypen bezüglich I und an der Anfrage beteiligten Tupeln unterschieden werden. Allerdings unterscheidet sich der Inversentyp hier nach der Art der Projektion. Ist der Schlüssel der Relation noch vollständig im Ergebnis enthalten, lässt sich eine tupelerhaltend-relaxierte CHASE-Inverse

$$Ergebnis(A_2, ..., A_n) \rightarrow \exists A_1 : R(A_1, A_2, ...A_n) \tag{4.1}$$

bilden, indem die herausprojizierten Attribute mit Nullwerten aufgefüllt werden [AH18b]. Ist der Schlüssel nicht mehr im Ergebnis enthalten, kann der Trigger einiger Tupel inaktiv sein, da bereits ein anderes Tupel die gleichen Werte in *Ergebnis* generiert hat. Folglich handelt es sich hierbei nur noch um eine relaxierte Inverse. Dies lässt sich jedoch durch Einbindung der Provenance beheben. Für Ergebnistupel, die durch Zusammenfallen von Tupeln entstanden sind, lassen sich ihre Ursprünge durch die ihre Zeugenbasis rekonstruieren, um eine tupelerhaltend-relaxierte Inverse zu erreichen [AH18b]. Bei der Evolution ist das Entfernen von Attributen ein herausprojizieren dieser in den s-t-TGDs, welche die Evolutionsoperationen umsetzen. Weiter tritt bei Evolutionen häufig auch ein zweiter Fall auf, der bei Anfragen nicht vorgesehen ist. Das Hinzufügen von Attributen lässt sich strukturell auch durch eine Projektion abbilden. Hierbei findet eine Projektion auf alle Attribute der Relation und ein Attribut X statt, welches nicht in R vorkommt. Eine entsprechende s-t-TGD

$$R(A_1, A_2, ...A_n) \rightarrow \exists X : Ergebnis(X, A_1, A_2, ..., A_n)$$

lässt sich jedoch einfach Invertieren, indem das Attribut X aus dem Rumpf der Inversen

$$Ergebnis(X, A_1, A_2, ..., A_n) \rightarrow R(A_1, A_2, ...A_n)$$

herausprojiziert wird. Da beim Hinzufügen eines Attributes allerdings nur neue Werte hinzukommen, ist diese Inverse stets exakt. Neben den in 4.3.1 genannten Fällen, gefährdet die Projektion die Reproduzierbarkeit von Anfragen nach einer Schemaevolution, wenn die Evolution ein Attribut entfernt, auf welches in einer Anfrage projiziert wird. Auch hier müssen die von den Anfragen betroffenen Tupel aufbewahrt werden, bzw. die Werte für das herausprojizierte Attribut rekonstruierbar bleiben.

Verbund

Der Verbund wird in s-t-TGDs über gleiche Variablen an den entsprechenden Positionen realisiert. Soll beispielsweise R mit einer zweiten Relation S mit den Attributen B_i über die Verbundbedingung $A_1 = B_1$ verknüpft werden, lautet die entsprechende s-t-TGD

$$R(A_1, A_2, ..., A_n) \wedge S(A_1, B_2, ..., B_n) \rightarrow Ergebnis(A_1, ..., A_n, B_2, ..., B_n).$$

Die Inverse

$$Ergebnis(A_1, ..., A_n, B_2, ..., B_n) \rightarrow R(A_1, A_2, ..., A_n) \wedge S(A_1, B_2, ..., B_n)$$

ist hier bezüglich I entweder exakt oder relaxiert. Die Inverse ist genau dann exakt, wenn alle Tupel aus R mindestens einen Verbundpartner in S besitzen und umgekehrt [AH18b]. Bezüglich der an dem Verbund beteiligten Tupel ist die Inverse jedoch stets exakt, sodass hier keine zusätzlichen Informationen benötigt werden.

Umbenennung

Die Umbenennung wird in s-t-TGDs über das Schema gelöst. Dieses legt Attributnamen fest, sodass für die Umbenennung eines Attributes der Attributname der Relation angepasst werden muss. Daher ist die Umbenennung in einer s-t-TGD

$$R(A_1, ..., A_n) \rightarrow Ergebnis(A_1, ..., A_n)$$

nicht direkt sichtbar, sondern nur unter Beachtung der jeweiligen Relationenschemas. Daher ist die Inverse aber jedoch exakt, da alle Variablen gleichermaßen in Kopf und Rumpf der Abhängigkeit vorkommen [AH18b].

4.3.2 Berechnung und Invertierung der Anfrage

Die erste Phase besteht darin, mittels CHASE und BACKCHASE die Tupel zu identifizieren, welche an der Auswertung beteiligt waren. Dabei berechnen wir eine Teilinstanz I' von I im Schema S. Hierzu berechnen wir zunächst den CHASE auf I mit der als TGD formulierten Anfrage Q [AH18a]. Gleichzeitig berechnen wir die Zeugenbasis jedes Tupels im Anfrageergebnis K. Anschließend wird die Anfrage Q invertiert und mit $CHASE_{Q^{-1}}(K)$ die Teilinstanz I' berechnet (siehe Abbildung 4.1). Diese ist Ausgangslage für die Berechnung von I_{Prov}.

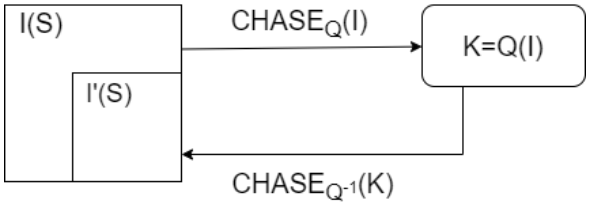


Abbildung 4.1: Phase 1 der Berechnung von I_{Prov} abgeleitet aus [AH18a]

4.3.3 Berechnung und Invertierung der Schemaevolution

Nun kann die Evolution mit der Menge der Evolutionsoperationen E betrachtet werden. Die Tupel in der nächsten Schemaversion S' können durch $CHASE_E(I)$ berechnet werden [AH18a]. Dabei erhalten Tupel in $I(S')$ neue TIDs für das aktuelle Schema und das TID-Dictionary für die Evolution von S zu S' wird angelegt. Das Dictionary wird bei der Invertierung genutzt, um die TIDs der Tupel für das Schema S zu rekonstruieren. Die Menge E^{-1}, welche als CHASE-Parameter für Rückrechnung der Evolution Anwendung findet, enthält dabei alle Abhängigkeiten, die durch die Invertierung der Abhängigkeiten aus E entstehen, sowie entsprechende Cleaning-EGDs C_i, wie sie bereits im obigen Beispiel zum TID-Dictionary gezeigt wurden. $CHASE_{E^{-1}}(I) = J$ ergibt dann eine Teilinstanz von $I(S)$ welche durch inverse Schemaoperationen berechnet werden kann (siehe Abbildung 4.2).

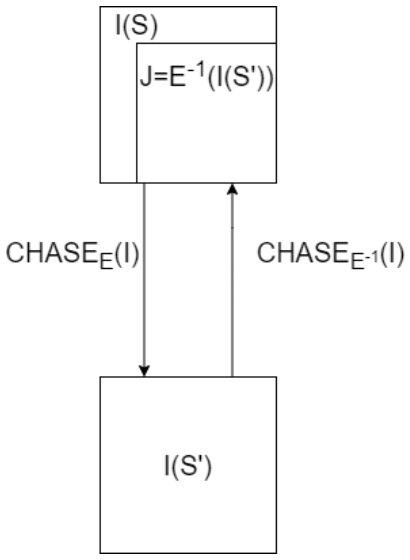


Abbildung 4.2: Phase 2 der Berechnung von I_{Prov}

4.3.4 Berechnung der zu speichernden Tupel

In [AH18a] wurde der Ansatz zur Verwendung des CHASE und BACKCHASE für den Provenance-Fall bereits vorgestellt. Wir erweitern diesen hier um die Berechnung von I_{Prov}. Hierbei ergibt sich I_{Prov} wie folgt aus den Tupeln, welche in die Anfrage Q eingeflossen sind, und denen, welche sich aus $I(S')$ durch Rückrechnung der Schemaevolution berechnen lassen:

$$I_{Prov} = I'(S) - J_M.$$

J_M ist hierbei die Menge der Tupel aus $I'(S)$, welche sich auf Tupel aus J mit dem Mapping M abbilden lassen. Dieser Schritt ist wichtig, damit sichergestellt wird, dass die Tupel aus J auch alle Attribute enthalten, welche für die Anfrage Q relevant sind und keine Nullwerte für diese Attribute auftauchen. Dabei können Konstanten nur mit der Identität und Nullwerte auf beliebige Werte abgebildet werden. Eine Außnahme bilden hier allerdings Attribute, die für Verbundoperationen in der Anfrage relevant sind, aber aus dem Ergebnis herausprojiziert wurden. Bei der Invertierung, werden deren Attributwerte mit Nullwerten belegt. Werden diese wiederum bei der Abbildung auf Nullwerte abgebildet, kann das ursprüngliche Verbundkriterium verloren gehen und statt dessen beispielsweise ein Kreuzprodukt entstehen. Diese Nullwerte müssen daher besonders gekennzeichnet werden und müssen ebenfalls auf Konstanten abgebildet werden.

4.3.5 Archivierung der Daten

I_{Prov} wird nun archiviert und die Tupel aus $I(S)$, die nicht für die Reproduzierbarkeit wissenschaftlicher Auswertungen aufgehoben werden müssen, können gelöscht werden, um Speicherplatz für neue Daten freizugeben. Die Eigenschaft der Speicherpflicht von Tupeln aus $I'(S)$, welche nicht in I_{Prov} vorkommen (D), vererbt sich damit aber an die Tupel aus $I(S')$, welche eben diese rekonstruieren. Werden jene Tupel bei einem späteren Evolutionsschritt nicht mehr für wissenschaftliche Auswertungen im Schema S' verwendet, und müssen dementsprechend beim nächsten Evolutionsschritt nicht aufgehoben werden, muss dies berücksichtigt werden. Offensichtlich muss zwischen dauerhafter Archivierung und der vererbten Pflicht zur Aufbewahrung unterschieden werden, da sich die vererbte Aufbewahrungspflicht mit zunehmenden Schemaevolutionen weiter fortpflanzen kann. Betrachten wir zur Veranschaulichung die folgende Abbildung 4.3.

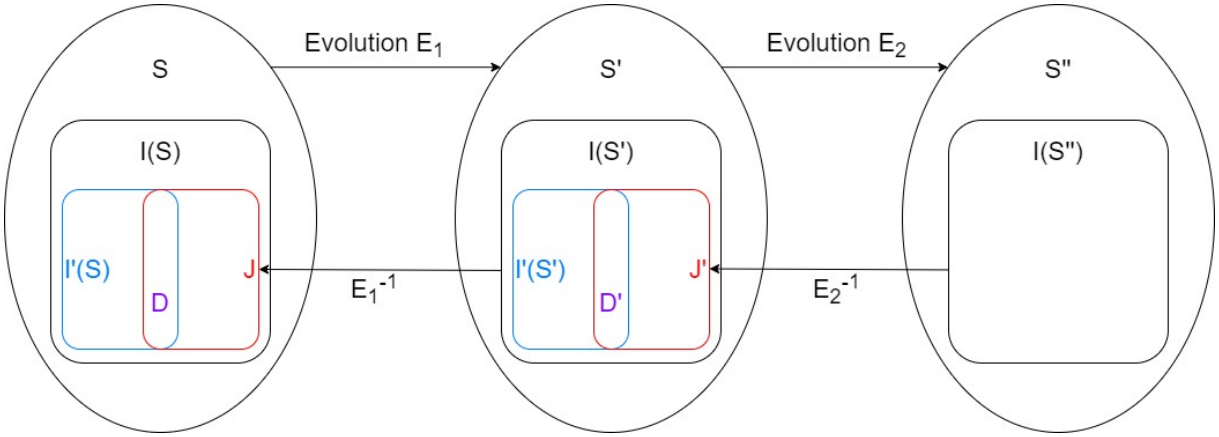


Abbildung 4.3: Archivierung der Daten

Bei der Evolution von S zu S' mit E_1, muss nur I_{Prov} ($I'(S)$ ohne J) gespeichert werden. Führen wir nun aber eine zweite Evolution von S' nach S'' durch, muss nicht nur auf das I_{Prov} dieser Schemaversion, sondern auch auf die Tupel aus $I(S')$ geachtet werden, welche D rekonstruieren ($E_1(D)$). Diese müssen nicht zwangsläufig Teil der Menge I_{Prov} für S' sein. Sie müssen jedoch entweder genau wie I_{Prov} gespeichert werden, oder ihre Speicherpflicht an $E_2(E_1(D))$ weitergeben. Es bietet sich an, die TIDs der entsprechenden Tupel in einer gesonderten Relationen zu halten. Diese wird dann beim Löschen von Elementen eingebunden, sodass Datensätze nicht gelöscht werden, wenn sie in dieser Relation vorliegen. Die TIDs von Tupeln, welche die Aufbewahrungspflicht vererbt bekommen haben, müssen dann bei einer Schemaevolution von S' zu S'' in dieser Tabelle durch die TIDs der Tupel, an welche sie diese Pflicht weitergeben, ersetzt werden. Voraussetzung dafür ist allerdings, dass die erbenden Tupel im Schema S'' wiederum die Attribute besitzen, welche für die ursprüngliche Anfrage Q relevant ist, sonst müssen auch die entsprechenden Tupel im Schema S' archiviert werden und keine Vererbung der Aufbewahrungspflicht ist nötig.

4.3.6 Beispiel

Im Folgenden betrachten wir unser Studentenbeispiel und führen an diesem eine Evolution und zwei Anfragen durch. Diese invertieren wir dann wie oben beschrieben, um die Tupel aus dem Schema S zu berechnen, welche nach der Evolution von Schemaversion S zu S' nicht gelöscht werden dürfen.

Beispiel: Ablauf einer Evolution
Gegeben sei eine Studentenrelation, eine Notenrelation und eine Relation, welche an der Universität angestellte Studenten (Hiwis) enthält. Die Relationen seien wie folgt strukturiert, wobei die Matrikelnummer immer Schlüssel der Relation ist. Der volle Name eines Studenten sei ein alternativer Schlüssel.

$$Student(matrikelnummer, name, vorname, studiengang, strasse, hausnummer, plz),$$
$$Note(matrikelnummer, modulnummer, note),$$
$$Hiwi(matrikelnummer, lehrstuhl).$$

Weiter seien folgenden zwei Anfragen bekannt. Q_1 gibt für jeden Informatikstudenten seine Matrikelnummer, Modulnummer und Note zurück. Diese bietet die Grundlage für die Noteneinsicht von Studenten. Q_2 gibt zu jedem Hiwi seinen vollen Namen, den Lehrstuhl und seine Anschrift aus. Dies kann beispielsweise benutzt werden, um die Briefe für Abrechnungen an den entsprechenden Hiwi

zu senden.
Offensichtlich müssen beide Anfragen auch über Schemaänderungen hinweg reproduzierbar bleiben, um Zeugnisse zu drucken, bzw. um Nachweise über beschäftigte Studenten zu liefern. Diese Anfragen lassen sich wie folgt als s-t-TGDs definieren, wobei die Variablennamen den abgekürzten Attributnamen entsprechen:

$$Q_1 : Student(ma, na, vo, \text{'Informatik'}, str, ha, plz) \wedge Note(ma, mo, no) \rightarrow Ergebnis_1(ma, mo, no),$$
$$Q_2 : Student(ma, na, vo, stu, str, ha, plz) \wedge Hiwi(ma, le) \rightarrow Ergebnis_2(na, vo, le, str, ha, plz).$$

Weiter soll das obige Schema in eine neue Version überführt werden. In dieser kommen die Namen der Studenten nicht mehr vor. Weiter wird die Hiwi-Relation geändert, um besser zu reflektieren, dass es noch weitere studentische Hilfskräfte gibt, welche nicht unbedingt als Hiwis bei Lehrstühlen angestellt sind.

$$Student'(matrikelnummer, studiengang, strasse, hausnummer, plz),$$
$$Note'(matrikelnummer, modulnummer, note),$$
$$StudHilfskraft(matrikelnummer, kostenstelle).$$

Da wir zwischen Quell- und Zielschema unterscheiden, müssen auch alle unveränderten Relationen durch s-t-TGDs abgebildet werden. Die Evolution lässt sich somit durch folgende s-t-TGDs beschreiben:

$$Student(ma, na, vo, stu, str, ha, plz) \rightarrow Student'(ma, stu, str, ha, plz),$$
$$Note(ma, ma, no) \rightarrow Note'(ma, mo, no),$$
$$Hiwi(ma, le) \rightarrow StudHilfskraft(ma, le).$$

Gegeben seien des Weiteren folgende Tupel. Diese werden in den folgenden Abbildungen 4.1, 4.2 und 4.3 tabellarisch entsprechend ihrer Relationen aufgeführt.

Matrikelnummer	**Name**	**Vorname**	**Studiengang**	**Strasse**	**Hausnummer**	**PLZ**
0042	Adam	Doug	Physik	Lange Strasse	89	18146
3200	Buches	Georg	Germanistik	Neue Strasse	4	18146
4711	Maier	Ronald	Informatik	Pappelallee	9	18146
9001	Kuhlmann	Goetz	Informatik	Schillerstrasse	96	18146

Tabelle 4.1: Student

Matrikelnummer	**Modulnummer**	**Note**
0042	00001	1.3
3200	10001	2.0
4711	20001	1.0
4711	20002	1.7
9001	20001	1.3
9001	20003	1.0

Tabelle 4.2: Note

Matrikelnummer	Lehrstuhl
0042	Theoretische Phsyik
9001	DBIS

Tabelle 4.3: Hiwi

Daraus ergeben sich durch den CHASE mit Q_1, Q_2 und den Evolutionsoperationen folgende Ergebnisse bzw. Relationen in einer neuen Schemaversion, welche in den Tabellen 4.4 bis 4.8 abgebildet werden. Der CHASE wird hierbei für jede Anfrage bzw. die s-t-TGDs für die Evolution einzeln ausgeführt. Wir vernachlässigen die Einbindung von TIDs hier, da jeweils eindeutige Attributkombinationen ({Matrikelnummer} bzw {Name, Vorname}) auf den rechten Seiten der Abhängigkeiten existieren, sodass durch TIDs an dieser Stelle keine Verbesserung erreicht werden kann.

Matrikelnummer	Modulnummer	Note
4711	20001	1.0
4711	20002	1.7
9001	20001	1.3
9001	20003	1.0

Tabelle 4.4: Ergebnis$_1$

Name	Vorname	Lehrstuhl	Strasse	Hausnummer	PLZ
Adam	Doug	Theoretische Physik	Lange Strasse	89	18146
Kuhlmann	Goetz	DBIS	Schillerstrasse	96	18146

Tabelle 4.5: Ergebnis$_2$

Matrikelnummer	Studiengang	Strasse	Hausnummer	PLZ
0042	Physik	Lange Strasse	89	18146
3200	Germanistik	Neue Strasse	4	18146
4711	Informatik	Pappelallee	9	18146
9001	Informatik	Schillerstrasse	96	18146

Tabelle 4.6: Student'

Matrikelnummer	Modulnummer	Note
0042	00001	1.3
3200	10001	2.0
4711	20001	1.0
4711	20002	1.7
9001	20001	1.3
9001	20003	1.0

Tabelle 4.7: Note'

Matrikelnummer	Kostenstelle
0042	Theoretische Phsyik
9001	DBIS

Tabelle 4.8: StudHilfskraft

Beginnen wir für die Berechnung von I_{Prov} mit der Invertierung der Evolutionsoperationen. Die Inverse bilden wir, indem wir Kopf und Rumpf der Abbildung vertauschen und ggf. die Variablen der neuen rechten Seite existenzquantifizieren, falls sie nicht auf der neuen linken Seite vorkommen. Somit erhalten wir die folgenden invertierten s-t-TGDs sowie die rekonstruierten Relationen (siehe Abbildungen 4.9 bis 4.11), welche durch den CHASE mit diesen invertierten s-t-TGDs berechnet werden (J).

$$Student'(ma, stu, str, ha, plz) \rightarrow Student(ma, Na, Vo, stu, str, ha, plz),$$
$$Note'(ma, mo, no) \rightarrow Note(ma, mo, no),$$
$$StudHilfskraft(ma, ko) \rightarrow Hiwi(ma, ko).$$

Matrikelnummer	Name	Vorname	Studiengang	Strasse	Hausnummer	PLZ
0042	η_1	η_2	Physik	Lange Strasse	89	18146
3200	η_3	η_4	Germanistik	Neue Strasse	4	18146
4711	η_5	η_6	Informatik	Pappelallee	9	18146
9001	η_7	η_8	Informatik	Schillerstrasse	96	18146

Tabelle 4.9: Student (rekonstruiert aus Student')

Matrikelnummer	Modulnummer	Note
0042	00001	1.3
3200	10001	2.0
4711	20001	1.0
4711	20002	1.7
9001	20001	1.3
9001	20003	1.0

Tabelle 4.10: Note (rekonstruiert aus Note')

Matrikelnummer	Lehrstuhl
0042	Theoretische Phsyik
9001	DBIS

Tabelle 4.11: Hiwi (rekonstruiert aus StudHilfskraft)

Anschließend berechnen wir für jede Anfrage eine Inverse. Zudem ermitteln wir mit dem CHASE, der invertierten Anfrage und dem Anfrageergebnis die rekonstruierbare Teilinstanz $I'(S)$ für jede Anfrage Q_i (siehe Abbildungen 4.12 bis 4.15). Besondere Acht muss hierbei auf Provenance-Nullwerte gelegt werden. Diese Nullwerte dürfen bei der Suche nach Abbildungen für die Berechnung von

I_{Prov} nicht auf Nullwerte abgebildet werden, da sie für die Berechnung der Anfrage wichtig waren. Da die einzigen Terme in einem relationalen Atom bei konjunktiven Anfragen, welche hierfür in Frage kommen, Konstanten oder Verbundattribute sind, ist das finden dieser einfach. Konstanten bleiben bei der Invertierung unverändert und können somit nur durch die Identität abgebildet werden. Variablen von Verbundattributen kommen auf der linken Seite der zu invertierenden Abhängigkeit mehrfach vor. Kommen diese Verbundattribute nicht im Ergebnis vor, müssen die bei der Invertierung entstehenden Nullwerte gesondert gekennzeichnet werden, z. B. durch eine besondere Benennung der Nullwerte. Der Übersichtlichkeit halber markieren wir solche Variablen und deren Nullwerte im Folgenden rot.

$Q_1^{-1} : Ergebnis_1(ma, mo, no) \rightarrow Student(ma, Na, Vo, \text{'Informatik'}, Str, Ha, Plz) \wedge Note(ma, mo, no)$,
$Q_2^{-1} : Ergebnis_2(na, vo, le, str, ha, plz) \rightarrow Student(Ma, na, vo, stu, str, ha, plz) \wedge Hiwi(Ma, le)$.

Matrikelnummer	Name	Vorname	Studiengang	Strasse	Hausnummer	PLZ
4711	η_1	η_2	Informatik	η_3	η_4	η_5
9001	η_6	η_7	Informatik	η_8	η_9	η_{10}

Tabelle 4.12: Student (rekonstruiert aus $Ergebnis_1$)

Matrikelnummer	Modulnummer	Note
4711	20001	1.0
4711	20002	1.7
9001	20001	1.3
9001	20003	1.0

Tabelle 4.13: Note (rekonstruiert aus $Ergebnis_1$)

Matrikelnummer	Name	Vorname	Studiengang	Strasse	Hausnummer	PLZ
η_1	Adam	Doug	η_2	Lange Strasse	89	18146
η_3	Kuhlmann	Goetz	η_4	Schillerstrasse	96	18146

Tabelle 4.14: Student (rekonstruiert aus $Ergebnis_2$)

Matrikelnummer	Lehrstuhl
η_1	Theoretische Phsyik
η_3	DBIS

Tabelle 4.15: Hiwi (rekonstruiert aus $Ergebnis_2$)

Schlussendlich können wir I_{Prov} berechnen, indem wir die durch die Anfrageergebnisse rekonstruierten Ergebnisse auf die aus der Evolution rekonstruierten Tupel abbilden. Dabei können Konstanten nur durch ihre Identität, Provenance-Nullwerte nur durch beliebige Konstanten und Nullwerte durch beliebige Werte abgebildet werden. Tupel die nicht abgebildet werden können, müssen aufgehoben

werden, damit die Anfragen rekonstruierbar bleiben.

$$(4711, \eta_1, \eta_2, \text{Informatik}, \eta_3, \eta_4, \eta_5) \text{ und}$$
$$(9001, \eta_6, \eta_7, \text{Informatik}, \eta_8, \eta_9, \eta_{10})$$

aus Tabelle 4.12 lassen sich durch

$$(4711, \eta_5, \eta_6, \text{Informatik}, \mathit{Pappelallee}, 9, 18146) \text{ und}$$
$$(9001, \eta_7, \eta_8, \text{Informatik}, \mathit{Schillderstrasse}, 96, 18146)$$

aus Relation 4.9 rekonstruieren. Ebenso lassen sich

$$(\eta_1, \mathit{TheoretischePhsyik}) \text{ und}$$
$$(\eta_3, \mathit{DBIS})$$

auf die Tupel in Relation 4.11 abbilden. Die Tupel aus Tabelle 4.14 lassen sich jedoch nicht durch Tupel aus Relation 4.9 rekonstruieren, da die Attribute 'Name' und 'Vorname' dort nicht mit Konstanten besetzt sind. Daher müssen

$$(0042, \mathit{Adam}, \mathit{Doug}, \mathit{Physik}, \mathit{LangeStrasse}, 89, 18146) \text{ und}$$
$$(9001, \mathit{Kuhlmann}, \mathit{Goetz}, \mathit{Informatik}, \mathit{Schillerstrasse}, 96, 18146)$$

aufgehoben werden und bilden somit für dieses Beispiel I_{Prov}.

Kapitel 5

Implementierung

In diesem Kapitel wird die Implementierung des Konzeptes in ChaTEAU vorgestellt. ChaTEAU wurde als Java-Projekt implementiert, sodass auch die folgenden Ausführungen im Kontext dieser Programmiersprache zu betrachten sind.

5.1 Allgemeines

Die hier vorgestellte Implementation bezieht sich auf einen Spezialfall möglicher Schemaevolutionen. Grund dafür ist, dass ChaTEAU derzeit keine Unterstützung für second-order-TGDs anbietet. Dies ist jedoch Voraussetzung für den im Konzept beschriebenen Ansatz. Des Weiteren wurde in [AMJ+20] eine Fallstudie anhand einer Forschungsdatenbank vorgestellt. Diese zeigt, dass dort ca. 80% aller Schemaänderungen das Hinzufügen eines neuen Attributes sind. Die zweithäufigste Form der Schemaänderung ist das Verschmelzen von Attributen (ca. 9%). Solche Verschmelzungen werden jedoch wiederum durch Funktionen umgesetzt. Die Werte der zu verschmelzenden Attribute werden Parameter einer Funktion, deren Rückgabewert den Wert des neuen Attributes bestimmt. Die ursprünglichen Attribute werden dann aus dem Schema entfernt. Dies umzusetzen, erfordert dementsprechend second-order-TGDs.

Beispiel: Verschmelzung durch second-order-TGD
Betrachten wir als Beispiel eine Relation, die Veranstaltungen an einer Universtität speichert. Die Relation charakterisiert Veranstaltungen durch einen Namen (na), eine Veranstaltungsnummer (ve), die Zahl der eingetragenen Dozenten (zd) und die Menge der Studenten, welche für diese Veranstaltung eingetragen sind (zs). In der nächsten Schemaversion soll die Zahl der Dozenten und Studenten zu einer Teilnehmerzahl (zt) zusammengefasst werden. Die strukturelle Änderung lässt sich durch das Entfernen der Attribute für die Dozenten- und Studentenzahl und das Hinzufügen eines Teilnehmerzahl-Attributes umsetzen. Jedoch gehen dabei existierende Werte verloren. Folgende second-order-TGD erlaubt die Verschmelzung ohne diesen Verlust durchzuführen.

$$Veranstaltung(na, ve, zd, zs) \rightarrow Veranstaltung'(na, ve, zt).$$

Deshalb betrachten wir im Folgenden ausschließlich Schemaevolutionen bei denen Attribute hinzugefügt oder entfernt werden. Die Implementation wird auf diesen Anwendungsfall angepasst. Da Attribute, welche mehrwertig von einem Schlüssel abhängen würden, in einem guten Datenbankentwurf in eine gesonderte Tabelle ausgelagert werden, bedeutet dies, dass die TIDs in diesem Fall unverändert bleiben. Somit entfällt die Notwendigkeit des TID-Dictionary für eine Schemaevolution.
Weiter ist es auf Grund der Implementation des CHASE-Algorithmus in ChaTEAU nur über Umwege möglich, durch s-t-TGDs neue Relationen zu definieren. Ein Ursache dessen ist, dass Attribute typisiert sind und immer einen Namen brauchen. Entsprechend müssten neue Relationen aus den angegebenen Integritätsbedingungen generiert werden. Dies wäre durch die reine Abhängigkeit von den Eingabevariablen aber sehr fehleranfällig. Deshalb wird für die folgenden Implementationen das Zielschema einer Evolution oder Anfrage immer mit dem *SchemaTag* T und das Eingabeschema mit S im Input versehen.

5.2 ChaTEAU

ChaTEAU wurde an der Universität Rostock in mehreren Phasen entwickelt. [BKM+17] lieferte hierbei die Grundlage für die Implementierung des CHASE-Algorithmus. Andere Abschlussarbeiten haben diverse Funktionalitäten und Konzepte implementiert, welche im Folgenden ausgenutzt werden. Somit ist in ChaTEAU bereits der Standard-CHASE implementiert und kann auf Instanzen und Anfragen ausgeführt werden. Dabei unterstützt ChaTEAU bereits EGDs, TGDs und s-t-TGDs. Für letztere enthalten Instanzen eine Abbildung von Relationen auf Tags, welche eine Unterscheidung zwischen Quellschema (S) und Zielschema (T) erlauben. Weiter lassen sich aus den Instanzen das Schema (Abbildung von Relationennamen auf ihre Attribute) sowie die Gesamtheit aller enthaltenen Attribute erfragen. Tupel werden in Instanzen als Menge relationaler Atome gespeichert. Dabei unterliegen sie keiner weiteren Ordnung oder besonderen Zugriffsstrukturen. Das relationale Atom bildet sich dann, indem der Name der Relation als Symbol für das Atom auftritt und die Terme des Atoms mit den Konstanten bzw. Nullwerten des Tupels besetzt werden. Der Informatikstudent aus unserem Beispiel 2.1 wird dementsprechend als $Student(4711, Informatik, Ronald, Maier)$ gespeichert. Die relationalen Atome enthalten abseits des Relationennamens keine weiteren Schemainformationen.
ChaTEAU erhält seine Eingaben aus XML-Dateien, deren Informationen beim Einlesen in Instanzen und Integritätsbedingungen transformiert werden. Die Eingabe von Instanzen besteht dabei aus den Schemainformationen inklusive Integritätsbedingungen und den relationalen Atomen, die bereits vorliegen. Ein Beispiel für so eine Eingabedatei bietet die Abbildung 5.1.

```
<input>
  <schema>
    <relations>
      <relation name="Studenten" tag="S">
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="nachname" type="string"/>
        <attribute name="vorname" type="string"/>
      </relation>
      ...
    </relations>
    <dependencies>
      <sttgd>
        <body>
          <atom name="Studenten">
            <variable name="#V_matrikelnummer_1" />
            <variable name="#V_nachname_1" />
            <variable name="#V_vorname_1" />
          </atom>
          ...
        </body>
        <head>
          ...
        </head>
      </sttgd>
    </dependencies>
  </schema>
  <instance>
    <relationalAtom name="Studenten">
      <tuple id="Studenten_1">
        <constant name="3" />
        <constant name="Mueller" />
        <constant name="Max" />
      </tuple>
      ...
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt 5.1: Ausschnitt - Beispiel Eingabedatei

5.3 Umsetzung

Der in Kapitel 4 vorgestellte Ansatz, läuft in drei Phasen ab. Die ersten beiden Teilschritte, welche unabhängig voneinander ausgeführt werden können, sind sich grundlegend sehr ähnlich. Sowohl für die Evolution als auch für die Anfragen wird die Ausgangsinstanz mit s-t-TGDs gechaset. In letzterem Fall wird jedoch nur die s-t-TGD der Anfrage selbst genutzt. In beiden Fällen muss die Menge der Abhängigkeiten anschließend invertiert werden. Somit lassen sich diese beiden Phasen aus dem Konzept in einer Klasse (siehe Quellcodeausschnitt 5.2) vereinigen.

```
public class Provenance {
  private Instance source;
  private LinkedHashSet<IntegrityConstraint> sigma;
  ...
}
```

Quellcodeausschnitt 5.2: Ausschnitt - Provenance.java

Für jede Anfrage und jeden Evolutionsschritt, muss das entsprechende Objekt der *Provenance*-Klasse persistent gespeichert werden. Dazu lassen sich die Schemainformationen und die Menge der Integritätsbedingungen in eine Datei schreiben. Damit diese auch wieder eingelesen werden kann, bieten sich hier Frameworks zur Persistierung von Java-Objekten an.
Um das Invertieren von Abhängigkeiten zu ermöglichen, wurde eine entsprechende Methode für die Klasse *TGD* implementiert. Da s-t-TGDs von dieser abgeleitet sind, steht auch ihnen diese Funktionalität zur Verfügung. In ChaTEAU sind allquantifizierte Variablen in Abhängigkeiten gekennzeichnet, indem sie mit der Zeichenkette *'#V'* beginnen. Existienzquantifizierte Variablen sind durch *'#E'* markiert und Nullwerte an *'#N'* erkennbar. Um eine Abhängigkeit zu invertieren, müssen zunächst die relationalen Atome der linken und rechten Seite vertauscht werden. Dabei müssen Attribute der rechten Seite, welche nun nicht mehr in der linken Seite vorkommen, statt eines Allquantors mit einem Existenzquantoren versehen werden. Ebenso müssen vorher existenzquantifizierte Variablen, welche jetzt auf der linken Seite der Abhängigkeit vorkommen, allquantifiziert werden. So wird beispielsweise durch eine Invertierung

$$\begin{gathered} Student(\#V\_Matrikelnummer\_1, \#V\_Studiengang\_1) \\ \rightarrow Hiwi(\#V\_Matrikelnummer\_1, \#E\_Lehrstuhl\_1) \\ zu \\ Hiwi(\#V\_Matrikelnummer\_1, \#V\_Lehrstuhl\_1) \\ \rightarrow Student(\#V\_Matrikelnummer\_1, \#E\_Studiengang\_1). \end{gathered}$$

Im folgenden Quellcodeausschnitt 5.3 wird die Methode gezeigt, welche die Invertierung einer TGD ermöglicht. Besonders wichtig ist hierbei, dass die zu invertierende Abhängigkeit nicht verändert wird. Dies hätte Auswirkungen auf alle Vorkommen der Abhängigkeit. Um dies zu verhindern, werden immer neue Terme und relationale Atome erzeugt. Die neu erzeugte TGD ist somit im Kontext ihrer Objektidentität unabhängig, sodass beispielsweise bei der Invertierung einer Evolutionsoperation keine Fehler auftreten. Da die rechte Seite einer Abhängigkeit im Fall einer EGD kein relationales Atom ist, muss an gegebenen Stellen eine Typumwandlung vorgenommen werden (siehe Abbildung 5.3 Z. 16). Diese ist jedoch im Kontext einer TGD stets sicher. In Kapitel 4 wurden besondere Nullwerte vorgestellt, welche das Anfrageergebnis beeinflussen, aber aus dem Ergebnis herausprojiziert werden. Diese *Provenance-Nullwerte* werden hier gesondert gekennzeichnet. Dafür wird bei der Invertierung einer TGD die linke Seite der Abhängigkeit nach Mehrfachvorkommen einer Variable überprüft, welche nicht auf der rechten Seite vorkommt. Solche allquantifizierten Variablen werden bei der Invertierung zu existenzquantifizierten Variablen mit der Kennzeichnung *Prov#*.

```
1  public Tgd invert() {
2    HashSet<RelationalAtom> newBody = new HashSet<RelationalAtom>();
3    for (RelationalAtom a: (HashSet<RelationalAtom>) this.getHead()) {
4      ArrayList<Term> terms = a.getTerms();
5      ArrayList<Term> newTerms = new ArrayList<Term>();
6      for(Term t: terms) {
7        Term newTerm = new Term(t);
8        if(newTerm.getTermType()==TermType.Null) {
9          t.setTermType(TermType.Variable);
10       }
11       newTerms.add(newTerm);
12     }
13     newBody.add(new RelationalAtom(a.getName(), newTerms));
14   }
15   HashSet<RelationalAtom> newHead = new HashSet<RelationalAtom>();
16   HashSet<RelationalAtom> oldHead =
     (HashSet<RelationalAtom>)this.getHead();
17   for (RelationalAtom a: this.getBody()) {
18     ArrayList<Term> terms = a.getTerms();
19     ArrayList<Term> newTerms = new ArrayList<Term>();
20     for(Term t: terms) {
21       Term newTerm = new Term(t);
22       for(RelationalAtom atom:  oldHead) {
23         if(!(atom.getTerms().contains(t))&&
           t.getTermType().equals(TermType.Variable)) {
24           int countOccurrences = 0;
25           for(RelationalAtom b: this.getBody()) {
26             if(b.getTerms().contains(t)) countOccurrences++;
27           }
28           if(countOccurrences >= 2) {
29             newTerm = new
               Term(t.getTermValueVariable().toString().replaceFirst("#V_",
               "#E_Prov#"));
30           }
31           else {
32             newTerm = new
               Term(t.getTermValueVariable().toString().replaceFirst("#V",
               "#E"));
33           }
34         }
35       }
36       newTerms.add(newTerm);
37     }
38     newHead.add(new RelationalAtom(a.getName(), newTerms));
39   }
40   return new Tgd(newBody, newHead);
41 }
```

Quellcodeausschnitt 5.3: Tgd.java - Methode zur Invertierung einer TGD

ChaTEAU ist jedoch durch vorangegangene Implementationen darauf ausgelegt, dass die Namen der Terme den Namen der Attribute entsprechen müssen, welche sie repräsentieren. Daher kann der Teil der Implementation aus Quellcodeausschnitt 5.3, welcher die Variablennamen von Provenance-Nullwerten markiert, derzeit nicht verwendet werden.
Bei der Invertierung einer Anfrage oder Schemaevolution, wird für jede Abhängigkeit überprüft, ob es sich dabei um eine TGD handelt. Es ist bei einer Schemaevolution nicht ungewöhnlich, dass EGDs auf dem Zielschema formuliert werden, z. B. um die Zieldatenbank zu bereinigen. Diese besitzen jedoch keine Inverse, da die alten Werte nach Anwendung der EGD verloren gehen. Somit können diese bei der Invertierung der Abhängigkeiten vernachlässigt werden. Die Bildung der Inverse lässt sich direkt über das entsprechende Provenance-Objekt durch die *invert*-Methode (siehe Quellcodeausschnitt 5.4) auslösen.

```
1  public Provenance invert(ChaseLog log) {
2    LinkedHashSet<IntegrityConstraint> sigmaInv = new
     LinkedHashSet<IntegrityConstraint>();
3    for(IntegrityConstraint c: this.getSigma()) {
4      if(c instanceof Tgd) {
5        sigmaInv.add(((Tgd) c).invert());
6      }
7    }
8    HashMap<String, SchemaTag> oldTags = this.getSource().getSchemaTags();
9    HashMap<String, SchemaTag> newTags = new HashMap<String, SchemaTag>();
10   for(String rel: oldTags.keySet()) {
11     if(oldTags.get(rel).equals(SchemaTag.T)) {
12       newTags.put(rel, SchemaTag.S);
13     }
14     else {
15       newTags.put(rel, SchemaTag.T);
16     }
17   }
18   Instance i = new
     Instance(this.getTargetInstance(log).getRelationalAtoms(),
     this.getSource().getAttributes(), this.getSource().getSchema(),
     OriginTag.i, newTags);
19   return new Provenance(i, sigmaInv);
20 }
```

Quellcodeausschnitt 5.4: Tgd.java - Methode zur Invertierung einer TGD

Der CHASE mit Integritätsbedingungen als CHASE-Parametern und einer Instanz als CHASE-Objekt, berechnet neue Fakten (TGDs) bzw. Änderungen von Werten (EGDs). Jedoch besaß ChaTEAU bisher keine Funktionalität, welche ausschließlich die neu berechneten Fakten des CHASE zurückgibt. Dies ist für die Formulierung von Anfragen als TGDs und die Berechnung von Schemaevolutionen wichtig, da nicht alle in der Quellinstanz vorliegenden Tupel im Ergebnis enthalten sein dürfen. Hierfür enthält die Provenance-Klasse die Methode *getTargetInstance* (siehe Quellcodeausschnitt 5.5). Diese setzt zudem bereits das Zielschema. Relationen, die als Teil des Zielschemas markiert sind, werden in die neue Instanz aufgenommen und dort als Teil des Quellschemas markiert. Relationen und deren Tupel, welche aus dem ursprünglichen Quellschema stammen, werden vernachlässigt.
Auch hier muss eine Unabhängigkeit zwischen den ursprünglichen Instanzen und der neu berechneten Instanz für die Anfrage oder neue Schemaversion herrschen. Diese erreichen wir durch Anlegen neuer Mengen und Abbildungen für das Schema. Da die Namen der Relationen nicht verändert werden, können wir diese direkt referenzieren. Selbst bei der Umbenennung eines Attributes ist dies unproblematisch,

weil dies durch unterschiedliche Namen in Ziel- und Quellschema der Eingabedatei repräsentiert wird. Die eingelesenen Zeichenketten bleiben somit trotzdem unverändert.

```
1  public Instance getTargetInstance(ChaseLog log) {
2    Instance i = Chase.chase(this.getSource(), this.getSigma(), log);
3    HashSet<String> targetRelations = new HashSet<String>();
4    HashMap<String, SchemaTag> relations = i.getSchemaTags();
5    for(String rel: relations.keySet()) {
6      if(relations.get(rel).equals(SchemaTag.T)) {
7        targetRelations.add(rel);
8      }
9    }
10   HashMap<String, ArrayList<String>> targetSchema = new HashMap<String,
     ArrayList<String>>();
11   for(String rel: targetRelations) {
12     targetSchema.put(rel, i.getSchema().get(rel));
13   }
14   HashMap<String, SchemaTag> targetSchemaTags = new HashMap<String,
     SchemaTag>();
15   for(String rel: targetRelations) {
16     targetSchemaTags.put(rel, SchemaTag.S);
17   }
18   HashSet<String> targetAttributes = new HashSet<String>();
19   for(String rel: targetRelations) {
20     targetAttributes.addAll(targetSchema.get(rel));
21   }
22   return new
     Instance(i.getRelationalAtomsBySchema(targetRelations),targetAttributes,
     targetSchema, OriginTag.i, targetSchemaTags);
23 }
```

Quellcodeausschnitt 5.5: Provenance.java - CHASE ohne Quelltupel

In dieser Teilimplementierung wird kein TID-Dictionary benötigt. Beim Hinzufügen oder Entfernen von Attributen bleiben die TIDs für Tupel von einer Schemaversion zur nächsten gleich. Daher legen wir hier fest, dass das TID-Attribut einer Relation, ähnlich wie Variablen und Nullwerte, besonders markiert wird. Per Konvention beginnt das TID-Attribut daher mir *TID#*, z. B. 'TID#Student'. Weiter müssen TIDs in der Eingabedatei eindeutig sein.
Schlussendlich vereinen wir die zwei Phasen für die Berechnung von I_{Prov}, zu sehen in Quellcodeausschnitt 5.6. Hierzu suchen wir Abbildungen aus den Inversen der Anfrageergebnisse in das Inverse der Evolution und geben die TIDs der zu speichernden Tupel zurück (siehe Quellcodeausschnitt 5.7). In beiden Fällen liegen diese als Instanzen vor. Da die relationalen Atome einer Instanz, welche die Tupel repräsentieren, nur Konstanten oder Nullwerte enthalten, ist die Suche nach Abbildungen unkompliziert. Für eine Anfrage überprüfen wir die *Terme* (Werte) jedes Tupels.

```
1  private static HashSet<Integer> findNondeducibleTuples(HashSet<Instance>
   inverseQueryResults, Instance inverseEvolutionInstance){
2    HashMap<String, String> tidMap = new HashMap<String, String>();
3    HashMap<String, ArrayList<String>> schema =
     inverseEvolutionInstance.getSchema();
4    for(String relName: schema.keySet()) {
5      ArrayList<String> relAttributes = schema.get(relName);
6      for(String relAttr: relAttributes) {
7        if(relAttr.contains("TID#")) {
8          tidMap.put(relName, relAttr);
9          break;
10       }
11     }
12   }
13   HashSet<Integer> tids = new HashSet<Integer>();
14   HashMap<String, HashSet<RelationalAtom>> inverseEvolutionAtomMap = new
     HashMap<String, HashSet<RelationalAtom>>();
15   for(String relName: inverseEvolutionInstance.getSchema().keySet()) {
16     inverseEvolutionAtomMap.put(relName,
       inverseEvolutionInstance.getRelationalAtomsBySchema(relName));
17   }
18   for(Instance i: inverseQueryResults) {
19     for(String relName: i.getSchema().keySet()) {
20       HashSet<RelationalAtom> atoms =
         i.getRelationalAtomsBySchema(relName);
21       HashSet<RelationalAtom> evolutionAtoms =
         inverseEvolutionAtomMap.get(relName);
22       String tidAttr = tidMap.get(relName);
23       for(RelationalAtom instanceAtom: atoms) {
24         ArrayList<Term> instanceTerms = instanceAtom.getTerms();
25         Boolean mappingFound = false;
26         for(RelationalAtom evolutionAtom: evolutionAtoms) {
27           ArrayList<Term> evolutionTerms = evolutionAtom.getTerms();
28           if(isMappable(instanceTerms, evolutionTerms)) {
29             mappingFound = true;
30             break;
31           }
32         }
33         if(!mappingFound) {
34           tids.add(instanceTerms.get(schema.get(relName).indexOf(tidAttr))
             .getTermValue());
35         }
36       }
37     }
38   }
39   return tids;
40 }
```

Quellcodeausschnitt 5.6: ProvenanceTest.java - Berechnung zuspeichernder Tupel

Handelt es sich bei dem Term um eine Konstante, muss in der Instanz der Schemainversen mindestens ein relationales Atom der selben Relation existieren, dessen Term an der gleichen Stelle den gleichen Wert besitzt. Ist dies für einen Term nicht der Fall, z. B. weil in der Instanz der Schemainversen an der Stelle einer Konstante nur Nullwerte existieren (Attribut bei der Evolution entfernt), ist das entsprechende relationale Atom nicht so rekonstruierbar, dass es für die Auswertung verwendet werden kann, und muss dementsprechend persistent gespeichert werden. Provenance-Nullwerte müssen ebenfalls auf Konstanten abgebildet werden, um die Verbundbedingung zu erhalten (siehe Abbilding 5.7: Zeile 10 ff.). Alle anderen Nullwerte können auf beliebige Werte abgebildet werden und benötigen daher keiner gesonderten Betrachtung.

```
1  private static Boolean isMappable(ArrayList<Term> instanceTerms,
   ArrayList<Term> evolutionTerms) {
2    for(int index = 0; index<instanceTerms.size(); index++) {
3      Term instanceTerm = instanceTerms.get(index);
4      Term evolutionTerm = evolutionTerms.get(index);
5      if(instanceTerm.getTermType().equals(TermType.Const)) {
6        if(!(instanceTerm.getConstType().equals(evolutionTerm.getConstType())
7        &&instanceTerm.getTermValue().equals(evolutionTerm.getTermValue())))
         {
8          return false;
9        }
10     }
11     else if(instanceTerm.getTermType().equals(TermType.Null)&&
       instanceTerm.getTermValueNull().getIndexName().contains("Prov#")){
12       if(evolutionTerm.getTermType().equals(TermType.Null)) {
13         return false;
14       }
15     }
16   }
17   return true;
18 }
```

Quellcodeausschnitt 5.7: ProvenanceTest.java - Überprüfung auf Abbildbarkeit

5.4 Beispiel

Zur Veranschaulichung greifen wir das Beispiel aus Abschnitt 4.3.6 auf. Aufgrund einiger Restriktionen, die durch vorherige Implementationen in ChaTEAU existieren, können die Anfragen aus diesem Beispiel nicht vollständig übernommen werden. ChaTEAU kann derzeit weder fehlerfrei Konstanten auf der linken Seite von Abhängigkeiten unterstützten, noch erlaubt es die beliebige Benennung von Variablen. Daher modifizieren wir die verwendeten Anfragen leicht. Die Relationen werden noch jeweils um ein TID-Attribut erweitert und die Relationen des Zielschemas für die Evolution umbenannt, um die Formulierung in XML zu realisieren. Die modifizierten Anfragen lauten

$$Q_1 : Student(tid_1, ma, na, vo, stu, str, ha, plz) \wedge Note(tid_2, ma, mo, no)$$
$$\rightarrow Ergebnis_1(ma, mo, no, tid_1, tid_2) \text{ und}$$
$$Q_2 : Student(tid_1, ma, na, vo, stu, str, ha, plz) \wedge Hiwi(tid_2, ma, le) \rightarrow$$
$$Ergebnis_2(ma, na, vo, le, str, ha, plz, tid_1, tid_2).$$

Diese Änderungen an der Anfrage verändern das Ergebnis (I_{Prov}) aus Abschnitt 4.3.6 nicht, da der Auslöser für diese Tupel dort war, dass eine Anfrage die Attribute 'Name' und 'Vorname' benötigte, diese jedoch bei der Evolution entfernt wurden. Dies wurde in diesem angepassten Beispiel beibehalten. ChaTEAU gibt die TIDs der Tupel

$$(0042, Adam, Doug, Physik, LangeStrasse, 89, 18146) \text{ und}$$
$$(9001, Kuhlmann, Goetz, Informatik, Schillerstrasse, 96, 18146)$$

zurück und liefert somit das erwartete Ergebnis. Das vollständige Beispiel, bestehend aus allen Eingabedateien sowie den berechneten Zwischenergebnissen und der Ausgabe von ChaTEAU, befindet sich im Anhang A.

Kapitel 6

Evaluation

In diesem Kapitel werden die Vor- und Nachteile der Methoden aus den Kapiteln 4 und 5 diskutiert. Dabei wird betrachtet, wie geeignet der hier vorgestellte Ansatz für jeweilige Anwendungsfälle ist und in wie fern die Implementation das Konzept umsetzen konnte und wie dabei aufgetretene Probleme bewältigt wurden. Einige der hier aufgeführten Punkte werden dann im Kapitel 7 aufgegriffen.

6.1 Evaluation des Konzepts

Der hier vorgestellte Ansatz erfüllt das Ziel der Minimierung zu speichernder Daten. Dank des TID-Dictionaries und der Erkennung für Anfragen relevanten Attributen, müssen wir nur die Tupel gesondert speichern, welche durch die Schemaevolution für mindestens eine Anfrage unzureichend abgebildet werden. Der Ansatz könnte somit in den normalen Ablauf einer Schemaänderung von Forschungsdatenbanken eingebettet werden. Die dafür aufzuhebenden Informationen der Schemaevolutionen (TGDs und ggf. EGDs) und der jeweiligen Schemaversionen (Relationennamen, Attribute, ...) sind im Vergleich zu vielen Replikationen ganzer Datenbestände nur ein Bruchteil an nötigem Speicherplatz.
Zudem wurde eine Möglichkeit gezeigt die Why-Provenance auf Attributebene mit dem CHASE zu kombinieren, indem nach Mehrfachaufkommen von Variablen gesucht wird, welche nicht Teil des Anfrageergebnisses sind. Diese erhalten bei der Invertierung der Anfrage einen speziellen Nullwert, welcher zeigt, dass dieser, im Gegensatz zu anderen Nullwerten, nicht irrelevant für die Auswertung ist.
Durch das TID-Dictionary benötigt der hier vorgestellte Ansatz im allgemeinen Fall einen CHASE-Algorithmus, welcher mit second-order-TGDs umgehen kann. Durch solche second-order-TGDs ließen sich aber auch komplexere Anfragen und Schemaevolutionen formulieren, was in der Praxis teilweise bereits getan wird. So machen das Verschmelzen oder Auftrennen von Attributen laut [AMJ+20] ca. 15% aller dort auftretenden Schemaänderungen aus. Diese können durch second-order-TGDs direkt umgesetzt werden, indem man entsprechende Funktionen hierfür einführt. Die Mehrheit aller Schemaänderungen besteht jedoch aus dem Hinzufügen oder Entfernen von Attributen. Diese Fälle können auch ohne second-order-TGDs umgesetzt werden.
Die Berechnungen, welche durch den hier vorgestellten Ansatz durchgeführt werden, bedeuten einen potenziell großen Mehraufwand für Schemaevolutionen in der Datenbank. Somit eignet sich die hier konzipierte Methode nicht für Datenbanken, welche sehr häufig Schemaänderungen unterliegen.
Zudem ist weiterhin offen, wie im allgemeinem Fall mit der vererbten Speicherpflicht umgegangen werden kann. Offensichtlich muss ein Tupel aus der Schemaversion S', welches ein Tupel in S abbildet, das für eine Anfrage auf dem selben Schema benötigt ist, seine Speicherpflicht bei der Schemaevolution nach S'' weitergeben. Dieser Punkt wird im nachfolgenden Kapitel 7 noch einmal genauer betrachtet.

6.2 Evaluation der Implementierung

Da der in ChaTEAU implementierte CHASE nicht mit second-order-TGDs umgehen kann, wurde in dieser Arbeit einer Teilimplementation umgesetzt. Hierbei beschränkten wir uns auf den häufigsten Fall der Schemaevolution, welcher ohne second-order-TGDs auskommt.
Die Implementation wurde in den aktuellen Aufbau von ChaTEAU eingebettet, indem zwei neue Klassen eingeführt und einige bereits existierende Klassen erweitert wurden. Die Implementation hat daher keine großen Modifikationen der bisherigen Implementierungen erfordert. Hier liegt aber auch ein bisher unbewältigtes Problem. Die ersten Implementierungen ChaTEAUs fanden allein unter dem Kontext des CHASE statt. Die hier vorgestellte BACKCHASE-Phase, welche wiederum eine Kombination aus der Anwendung des CHASE und Modifikation von Abhängigkeiten ist, z. B. durch Invertierung, war zu diesem Zeitpunkt nicht relevant, sodass die ursprünglichen Implementationen einige Probleme im Kontext dieser Arbeit hervorgerufen haben. So war z B. nicht vorgesehen, dass Terme ihren Term-Typ ändern können sollten, wie es bei der Invertierung nötig wäre. Daher war die beste Möglichkeit dies zu behandeln, die Variable aus ihrer Zeichenkette, wie sie in einer Input-Datei vorkommt, zu generieren. Weiter gibt es in ChaTEAU eine Vielzahl von Fallunterscheidungen, sowohl beim CHASE-Algorithmus als auch in anderen Programmteilen. Daher wäre das Einfügen neuer Term-Typen und abgeleiteter Klassen mit einer Zahl von Folgeproblemen verbunden gewesen. Deshalb wurden die Provenance-Nullwerte, welche für die Berechnungen von I_{Prov} wichtig sein können, durch eine besondere Kennzeichnung im Variablennamen markiert. Dabei fiel auf, dass ChaTEAU nur Namen für Variablen zulässt, die mit den korrespondierenden Attributnamen übereinstimmen. Somit kann die hier vorgestellte Implementierung keine Anfragen verarbeiten, deren Why-Provenance Attribute enthält, die nicht im Ergebnis-Schema vorkommen. Weiter ist beim Testen der Implementation aufgefallen, dass ChaTEAU Probleme mit Konstanten auf der linken Seite von Abhängigkeiten hat, wenn diese auf Instanzen angewandt werden. Da dies zu Fehlern beim CHASE führt, können auch solche Anfragen in diesem Kontext nicht verarbeitet werden.
Weiter wurde keine direkte Einbettung der Why-Provenance auf Tupelebene erreicht. Ziel für eine Implementierung der Why-Provenance wäre die Möglichkeit der Verarbeitung durch den CHASE-Algorithmus. Dieser kann jedoch ohne second-order-TGDs lediglich die relationalen Atome der Abhängigkeiten verarbeiten, sodass die TIDs für die Provenance dort mit eingebaut werden mussten. Daher wurden diese direkt als Teil der Anfrage betrachtet. Jedoch kann die Why-Provenance eines Ergebnistupels auch Alternativen für Zeugenbasen enthalten. Ein Umgang mit solchen mengenwertigen Attributen im Ergebnistupel ist jedoch aktuell nicht möglich, da der CHASE keine Möglichkeit zur Modifizierung von Tupeln bei nicht aktiven Triggern bietet.
In der Input-Datei gibt es für relationale Atome der Instanz bereits ein ID-Feld. Auf dieses konnte jedoch nicht innerhalb von ChaTEAU zugegriffen werden, sodass TIDs per Konvention als Bestandteil des Schemas festgelegt wurden. Fehlen TIDs, so erkennt die Methode zur Berechnung von I_{Prov} dies und bricht mit einem Hinweis auf das Fehlen dieser ab.
Schlussendlich ist die Form der Eingabe verbesserungswürdig. Aktuell kann ChaTEAU nur zwischen einem Quell- und einem Zielschema unterscheiden. Für den Anwendungsfall der Evolution, bei der auf die Rekonstruierbarkeit von Anfragen geachtet wird, ist dies nicht ausreichend. Ein Zielschema beschreibt die Struktur der Datenbank nach der Evolution, und ein Zielschema beschreibt strukturell je ein Anfrageergebnis. Daher wird in dieser Implementation für jede Anfrage und die Evolution jeweils eine Eingabedatei benötigt. Es muss deshalb nicht das komplette Quellschema für jede Anfrage aufgeführt werden, da nur die Relationen, welche abgefragt werden, benötigt sind. Jedoch ist das Auslesen aus einer gemeinsamen Eingabedatei praxisnäher und verhindert Fehler durch widersprüchliche Einträge in den Input-Dateien.

Kapitel 7

Zusammenfassung und Ausblick

In diesem letzten Kapitel wird die Arbeit strukturell und inhaltlich zusammengefasst. Weiter wird ein Ausblick darauf gegeben, welche Fragen bzw. Problemstellungen in dieser Arbeit nicht betrachtet wurden und somit noch offen sind.

7.1 Zusammenfassung

In Kapitel 1 wurde die Erweiterung ChaTEAUs um eine BACKCHASE-Phase motiviert und beschrieben, warum der Provenance-Fall eine besondere Bedeutung für das Forschungsdatenmanagement hat. Im darauffolgenden Kapitel 2 wurden zunächst die Grundlagen für diese Arbeit eingeführt. Hierbei wurden etablierte Konzepte in eine einheitliche Notation gebracht und für den Kontext dieser Arbeit definiert. Anschließend wurde in Kapitel 3 der Stand der Forschung und Technik vorgestellt. Hier wurden ausgewählte CHASE- und Provenance-Tools sowie die hier betrachten Anwendungsfälle des BACKCHASE vorgestellt und verglichen. Anschließend wurde in Kapitel 4 ein Ansatz ausgearbeitet, um die Reproduzierbarkeit von Anfragen wissenschaftlicher Auswertungen unter Schemaevolutionen effizienter zu gewährleisten. Damit nicht immer wieder Replikate des Datenbestandes gespeichert werden müssen, wurde hier der Ansatz aus [AH19] aufgegriffen. Dabei wurde gezeigt, wie mit Hilfe von second-order-TGDs eine Abbildung für Tuple-Identifier zwischen Schemaversionen gebildet werden kann. Diese kann durch den CHASE mit second-order-TGDs benutzt werden, um Tupel in alten Schemaversionen zu berechnen, auch wenn für jede Schemaversion neue TIDs vergeben werden. Für dieses Konzept wurde im darauffolgenden Kapitel 5 eine Teilimplementation vorgestellt. Diese beschränkte sich auf den in der Praxis am häufigsten auftretenden Spezialfall. Dieser ließ sich auch ohne Verwendung von second-order-TGDs behandeln. Das Konzept und die Implementierung wurden dann in Kapitel 6 bezüglich ihrer Umsetzung ausgewertet. Dabei wurde sowohl diskutiert, unter welchen Bedingungen das hier vorgestellte Konzept sinnvoll Einsatz finden kann, als auch hervorgehoben, an welchen Stellen noch Anpassungen vorgenommen werden müssen.

7.2 Ausblick

Im folgenden fassen wir die offenen Probleme und Fragen zusammen, welche im Laufe dieser Arbeit aufkamen. Zusätzlich beschreiben wir Ideen, welche als Lösungsansätze denkbar sind, aber im Rahmen dieser Arbeit nicht ausgearbeitet werden konnten.
Aus dem Konzept ging hervor, dass sich die Speicherpflicht eines Tupels durch die Schemaversionen fortpflanzt. Ist ein Tupel t im Schema S durch invertierte Schemaevolution aus dem Tupel t' in der Schemaversion S' rekonstruierbar, muss t nicht gespeichert werden. Jedoch muss t' bei einer darauffolgenden Schemaevolution aufgehoben werden, wenn bei der Rekonstruktion von t' aus Tupeln der nächsten Schemaversion Informationen verloren gehen, welche für die Rekonstruktion von t für eine entsprechende

Anfrage benötigt werden. Denkbar wäre nach einer Evolution, aber bevor Tupel gelöscht werden, die invertierte Evolution bis zur Schemaversion des Tupels durchzuführen, welches für die Reproduzierbarkeit der Anfrage benötigt wird. Dies hätte aber zur Folge, dass der Overhead zur Minimierung der Menge von gespeicherten Tupel weiter wächst. Dies müsste nämlich wiederum für jede Anfrage durchgeführt werden. Besser wäre eine Variante, in der nur Anfragen der aktuellen Schemaversion bei einer Evolution berücksichtigt werden müssen. Vorstellbar wäre auch die Verwendung von Annotationen für Attribute. Bei der Berechnung von I_{Prov} werden Provenance-Nullwerte besonders betrachtet, da diese für die Auswertung relevant sind. Es wäre daher denkbar, für jedes Tupel Provenance-Annotationen an die Attribute zu fügen, sodass bei einer Evolution, welche solche Attribute verändert oder entfernt, das Tupel in der aktuellen Schemaversion erkannt und aufgehoben wird.
Weiter ist noch unbekannt, wie eine allgemeine Why-Provenance in den CHASE eingearbeitet werden kann. Teilmengen mengenwertiger Attribute können vom Standard-CHASE nicht verarbeitet werden. Second-order-TGDs oder ein Zusatzschritt während des CHASE sind hier mögliche Ansätze.
Die Implementierung dieser Arbeit wurde durch die ersten Implementierungen ChaTEAUs eingeschränkt. ChaTEAU war auf den zu implementierenden Anwendungsfall nicht ausgelegt, da dieser bei den ersten Entwicklungsschritten keine Rolle spielte. Somit ist es beispielsweise nicht möglich, Attribute in Abhängigkeiten beliebig zu benennen, was die Berücksichtigung von Provenance-Nullwerten in der hiesigen Implementation unmöglich macht. Auch für die Bezeichnung von Rollen bestimmter Attribute in manchen Abhängigkeiten wäre dies wichtig. Weiter kann ChaTEAU derzeit nicht richtig mit Konstanten auf der linken Seite von Abhängigkeiten umgehen und behandelt diese wie allquantifizierte Variablen. Auch einige der hier implementierten Funktionalitäten, wie z. B. die Provenance-Nullwerte, eignen sich für eigene Klassen oder Typen. Diese in die bisherige Implementation ChaTEAUs einzuarbeiten, hätte aber an diversen Stellen Konsequenzen für die bestehende Implementation, z B. durch Fallunterscheidungen. Diese vorher bestehenden Probleme konnten im Rahmen dieser Arbeit nicht behoben werden.

Literaturverzeichnis

[ABG+20] Auge, Tanja, Willi Brekenfelder, Andreas Görres, Andreas Heuer, Ruven Kronenberg, Maximilian Lamster, Erik Manthey, Jasper Roloff und Jakob Zimmer: *Testing CHASE-based Tools for Data Exchange and Query Rewriting.* Technischer Bericht, 2020. In Bearbeitung - Stand 16.08.2020, 19:09.

[ABU79] Aho, Alfred V., Catriel Beeri und Jeffrey D. Ullman: *The Theory of Joins in Relational Databases.* ACM Trans. Database Syst., 4(3):297–314, 1979.

[AH18a] Auge, Tanja und Andreas Heuer: *Combining Provenance Management and Schema Evolution.* In: *IPAW*, Band 11017 der Reihe *Lecture Notes in Computer Science*, Seiten 222–225. Springer, 2018.

[AH18b] Auge, Tanja und Andreas Heuer: *The Theory behind Minimizing Research Data: Result equivalent CHASE-inverse Mappings.* In: *LWDA*, Band 2191 der Reihe *CEUR Workshop Proceedings*, Seiten 1–12. CEUR-WS.org, 2018.

[AH19] Auge, Tanja und Andreas Heuer: *ProSA - Using the CHASE for Provenance Management.* In: *ADBIS*, Band 11695 der Reihe *Lecture Notes in Computer Science*, Seiten 357–372. Springer, 2019.

[AMJ+20] Auge, Tanja, Erik Manthey, Susanne Jürgensmann, Susanne Feistel und Andreas Heuer: *Schema Evolution and Reproducibility of Long-term Hydrographic Data Sets at the IOW.* 2020.

[Aug17] Auge, Tanja: *Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen.* Masterarbeit, Universität Rostock, DBIS, 2017.

[BKM+17] Benedikt, Michael, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro und Efthymia Tsamoura: *Benchmarking the Chase.* In: *PODS*, Seiten 37–52. ACM, 2017.

[BKT01] Buneman, Peter, Sanjeev Khanna und Wang Chiew Tan: *Why and Where: A Characterization of Data Provenance.* In: *ICDT*, Band 1973 der Reihe *Lecture Notes in Computer Science*, Seiten 316–330. Springer, 2001.

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

[CJ09] Chapman, Adriane und H. V. Jagadish: *Why not?* In: *SIGMOD Conference*, Seiten 523–534. ACM, 2009.

[CM77] Chandra, Ashok K. und Philip M. Merlin: *Optimal Implementation of Conjunctive Queries in Relational Data Bases.* In: *STOC*, Seiten 77–90. ACM, 1977.

[DH13] Deutsch, Alin und Richard Hull: *Provenance-Directed Chase&Backchase.* In: *In Search of Elegance in the Theory and Practice of Computation*, Band 8000 der Reihe *Lecture Notes in Computer Science*, Seiten 227–236. Springer, 2013.

[DPT06] Deutsch, Alin, Lucian Popa und Val Tannen: *Query reformulation with constraints.* SIGMOD Rec., 35(1):65–73, 2006.

[FKPT11] Fagin, Ronald, Phokion G. Kolaitis, Lucian Popa und Wang Chiew Tan: *Schema Mapping Evolution Through Composition and Inversion.* In: *Schema Matching and Mapping*, Data-Centric Systems and Applications, Seiten 191–222. Springer, 2011.

[Gö20] Görres, Andreas: *Erweiterung des CHASE-Werkzeugs ChaTEAU um ein Terminierungskriterium.* Masterarbeit, Universität Rostock, DBIS, 2020.

[GKT07] Green, Todd J., Gregory Karvounarakis und Val Tannen: *Provenance semirings.* In: *PODS*, Seiten 31–40. ACM, 2007.

[GMPS13] Geerts, Floris, Giansalvatore Mecca, Paolo Papotti und Donatello Santoro: *The LLUNATIC Data-Cleaning Framework.* Proc. VLDB Endow., 6(9):625–636, 2013.

[GMPS14] Geerts, Floris, Giansalvatore Mecca, Paolo Papotti und Donatello Santoro: *Mapping and cleaning.* In: *ICDE*, Seiten 232–243. IEEE Computer Society, 2014.

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

[Hal00] Halevy, Alon Y.: *Theory of Answering Queries Using Views.* SIGMOD Rec., 29(4):40–47, 2000.

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

[Ile14] Ileana, Ioana: *Query rewriting using views : a theoretical and practical perspective. (Réécriture de requêtes avec des vues : une perspective théorique et pratique).* Doktorarbeit, Télécom ParisTech, 2014.

[Jur18] Jurklies, Martin: *CHASE und BACKCHASE: Entwicklung eines Universal-Werkzeugs für eine Basistechnik der Datenbankforschung.* Masterarbeit, Universität Rostock, DBIS, 2018.

[MMS79] Maier, David, Alberto O. Mendelzon und Yehoshua Sagiv: *Testing Implications of Data Dependencies.* ACM Trans. Database Syst., 4(4):455–469, 1979.

[Ren19] Renn, Fabian: *Erweiterung des CHASE-Werkzeugs ChaTEAU um Anfragetransformationen.* Bachelorarbeit, Universität Rostock, DBIS, 2019.

[SRH+20] Strelnikov, Artur, Chris Röhrs, Leon Herrmann, Max Kaseler und Rocco Flach: *Komplexe Software Systeme 2020 - Provenance-Tools.* Technischer Bericht, 2020. In Bearbeitung - Stand 16.08.2020, 19:07.

[Zim20] Zimmer, Jakob: *Vereinheitlichung des CHASE auf Instanzen und Anfragen am Beispiel ChaTEAU.* Bachelorarbeit, Universität Rostock, DBIS, 2020.

Anhang A

Beispiel ChaTEAU

Im Folgenden sind die Dateien für das ChaTEAU-Beispiel aus Abschnitt 5.4 aufgelistet.

A.1 Eingabedateien

```
<input>
  <schema>
    <relations>
      <relation name="Student" tag="S">
        <attribute name="TID#Student" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="name" type="string"/>
        <attribute name="vorname" type="string"/>
        <attribute name="studiengang" type="string"/>
        <attribute name="strasse" type="string"/>
        <attribute name="hausnummer" type="int"/>
        <attribute name="plz" type="int"/>
      </relation>
      <relation name="Note" tag="S">
        <attribute name="TID#Note" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="modulnummer" type="int"/>
        <attribute name="note" type="double"/>
      </relation>
      <relation name="Hiwi" tag="S">
        <attribute name="TID#Hiwi" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="lehrstuhl" type="string"/>
      </relation>
      <relation name="Student2" tag="T">
        <attribute name="TID#Student2" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="studiengang" type="string"/>
        <attribute name="strasse" type="string"/>
        <attribute name="hausnummer" type="int"/>
        <attribute name="plz" type="int"/>
```

```
    </relation>
    <relation name="Note2" tag="T">
      <attribute name="TID#Note2" type="int"/>
      <attribute name="matrikelnummer" type="int"/>
      <attribute name="modulnummer" type="int"/>
      <attribute name="note" type="double"/>
    </relation>
    <relation name="StudHilfskraft" tag="T">
      <attribute name="TID#StudHilfskraft" type="int"/>
      <attribute name="matrikelnummer" type="int"/>
      <attribute name="kostenstelle" type="string"/>
    </relation>
  </relations>
  <dependencies>
    <sttgd>
      <body>
        <atom name="Student">
          <variable name="#V_TID_1"/>
          <variable name="#V_matrikelnummer_1" />
          <variable name="#V_name_1" />
          <variable name="#V_vorname_1" />
          <variable name="#V_studiengang_1" />
          <variable name="#V_strasse_1" />
          <variable name="#V_hausnummer_1" />
          <variable name="#V_plz_1" />
        </atom>
      </body>
      <head>
        <atom name="Student2">
          <variable name="#V_TID_1"/>
          <variable name="#V_matrikelnummer_1" />
          <variable name="#V_studiengang_1" />
          <variable name="#V_strasse_1" />
          <variable name="#V_hausnummer_1" />
          <variable name="#V_plz_1" />
        </atom>
      </head>
    </sttgd>
    <sttgd>
      <body>
        <atom name="Note">
          <variable name="#V_TID_1"/>
          <variable name="#V_matrikelnummer_1" />
          <variable name="#V_modulnummer_1" />
          <variable name="#V_note_1" />
        </atom>
      </body>
      <head>
        <atom name="Note2">
          <variable name="#V_TID_1"/>
          <variable name="#V_matrikelnummer_1" />
```

```
            <variable name="#V_modulnummer_1" />
            <variable name="#V_note_1" />
          </atom>
        </head>
      </sttgd>
      <sttgd>
        <body>
          <atom name="Hiwi">
            <variable name="#V_TID_1"/>
            <variable name="#V_matrikelnummer_1" />
            <variable name="#V_lehrstuhl_1" />
          </atom>
        </body>
        <head>
          <atom name="StudHilfskraft">
            <variable name="#V_TID_1"/>
            <variable name="#V_matrikelnummer_1" />
            <variable name="#V_lehrstuhl_1" />
          </atom>
        </head>
      </sttgd>
    </dependencies>
  </schema>
  <instance>
    <relationalAtom name="Student">
      <tuple id="Student_1">
        <constant name="01" />
        <constant name="0042" />
        <constant name="Adam" />
        <constant name="Doug" />
        <constant name="Physik" />
        <constant name="Lange Strasse" />
        <constant name="89" />
        <constant name="18146" />
      </tuple>
      <tuple id="Student_2">
        <constant name="02" />
        <constant name="3200" />
        <constant name="Buches" />
        <constant name="Georg" />
        <constant name="Germanistik" />
        <constant name="Neue Strasse" />
        <constant name="4" />
        <constant name="18146" />
      </tuple>
      <tuple id="Student_3">
        <constant name="03" />
        <constant name="4711" />
        <constant name="Maier" />
        <constant name="Ronald" />
        <constant name="Informatik" />
```

```
      <constant name="Pappelallee" />
      <constant name="9" />
      <constant name="18146" />
    </tuple>
    <tuple id="Student_4">
      <constant name="04" />
      <constant name="9001" />
      <constant name="Kuhlmann" />
      <constant name="Goetz" />
      <constant name="Informatik" />
      <constant name="Schillerstrasse" />
      <constant name="96" />
      <constant name="18146" />
    </tuple>
  </relationalAtom>
  <relationalAtom name="Note">
    <tuple id="Note_1">
      <constant name="11" />
      <constant name="0042" />
      <constant name="00001" />
      <constant name="1.3" />
    </tuple>
    <tuple id="Note_2">
      <constant name="12" />
      <constant name="3200" />
      <constant name="10001" />
      <constant name="2.0" />
    </tuple>
    <tuple id="Note_3">
      <constant name="13" />
      <constant name="4711" />
      <constant name="20001" />
      <constant name="1.0" />
    </tuple>
    <tuple id="Note_4">
      <constant name="14" />
      <constant name="4711" />
      <constant name="20002" />
      <constant name="1.7" />
    </tuple>
    <tuple id="Note_5">
      <constant name="15" />
      <constant name="9001" />
      <constant name="20001" />
      <constant name="1.3" />
    </tuple>
    <tuple id="Note_6">
      <constant name="16" />
      <constant name="9001" />
      <constant name="20003" />
      <constant name="1.0" />
```

```
      </tuple>
    </relationalAtom>
    <relationalAtom name="Hiwi">
      <tuple id="Hiwi_1">
        <constant name="21" />
        <constant name="0042" />
        <constant name="Theoretische Physik" />
      </tuple>
      <tuple id="Hiwi_2">
        <constant name="22" />
        <constant name="9001" />
        <constant name="DBIS" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.1: Beispiel ChaTEAU - Evolution

```
<input>
  <schema>
    <relations>
      <relation name="Student" tag="S">
        <attribute name="TID#Student" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="name" type="string"/>
        <attribute name="vorname" type="string"/>
        <attribute name="studiengang" type="string"/>
        <attribute name="strasse" type="string"/>
        <attribute name="hausnummer" type="int"/>
        <attribute name="plz" type="int"/>
      </relation>
      <relation name="Note" tag="S">
        <attribute name="TID#Note" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="modulnummer" type="int"/>
        <attribute name="note" type="double"/>
      </relation>
      <relation name="Hiwi" tag="S">
        <attribute name="TID#Hiwi" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="lehrstuhl" type="string"/>
      </relation>
      <relation name="Ergebnis1" tag="T">
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="modulnummer" type="string"/>
        <attribute name="note" type="string"/>
        <attribute name="TID#Studenten" type="int"/>
        <attribute name="TID#Note" type="int"/>
      </relation>
    </relations>
```

```
    <dependencies>
      <sttgd>
        <body>
          <atom name="Student">
            <variable name="#V_TID_1"/>
            <variable name="#V_matrikelnummer_1" />
            <variable name="#V_name_1" />
            <variable name="#V_vorname_1" />
            <variable name="#V_studiengang_1" />
            <variable name="#V_strasse_1" />
            <variable name="#V_hausnummer_1" />
            <variable name="#V_plz_1" />
          </atom>
          <atom name="Note">
            <variable name="#V_TID_2"/>
            <variable name="#V_matrikelnummer_1" />
            <variable name="#V_modulnummer_1" />
            <variable name="#V_note_1" />
          </atom>
        </body>
        <head>
          <atom name="Ergebnis1">
            <variable name="#V_matrikelnummer_1" />
            <variable name="#V_modulnummer_1" />
            <variable name="#V_note_1" />
            <variable name="#V_TID_1"/>
            <variable name="#V_TID_2"/>

          </atom>
        </head>
      </sttgd>
    </dependencies>
  </schema>
  <instance>
    <relationalAtom name="Student">
      <tuple id="Student_1">
        <constant name="01" />
        <constant name="0042" />
        <constant name="Adam" />
        <constant name="Doug" />
        <constant name="Physik" />
        <constant name="Lange Strasse" />
        <constant name="89" />
        <constant name="18146" />
      </tuple>
      <tuple id="Student_2">
        <constant name="02" />
        <constant name="3200" />
        <constant name="Buches" />
        <constant name="Georg" />
        <constant name="Germanistik" />
```

```
        <constant name="Neue Strasse" />
        <constant name="4" />
        <constant name="18146" />
      </tuple>
      <tuple id="Student_3">
        <constant name="03" />
        <constant name="4711" />
        <constant name="Maier" />
        <constant name="Ronald" />
        <constant name="Informatik" />
        <constant name="Pappelallee" />
        <constant name="9" />
        <constant name="18146" />
      </tuple>
      <tuple id="Student_4">
        <constant name="04" />
        <constant name="9001" />
        <constant name="Kuhlmann" />
        <constant name="Goetz" />
        <constant name="Informatik" />
        <constant name="Schillerstrasse" />
        <constant name="96" />
        <constant name="18146" />
      </tuple>
    </relationalAtom>
    <relationalAtom name="Note">
      <tuple id="Note_1">
        <constant name="11" />
        <constant name="0042" />
        <constant name="00001" />
        <constant name="1.3" />
      </tuple>
      <tuple id="Note_2">
        <constant name="12" />
        <constant name="3200" />
        <constant name="10001" />
        <constant name="2.0" />
      </tuple>
      <tuple id="Note_3">
        <constant name="13" />
        <constant name="4711" />
        <constant name="20001" />
        <constant name="1.0" />
      </tuple>
      <tuple id="Note_4">
        <constant name="14" />
        <constant name="4711" />
        <constant name="20002" />
        <constant name="1.7" />
      </tuple>
      <tuple id="Note_5">
```

```
        <constant name="15" />
        <constant name="9001" />
        <constant name="20001" />
        <constant name="1.3" />
      </tuple>
      <tuple id="Note_6">
        <constant name="16" />
        <constant name="9001" />
        <constant name="20003" />
        <constant name="1.0" />
      </tuple>
    </relationalAtom>
    <relationalAtom name="Hiwi">
      <tuple id="Hiwi_1">
        <constant name="21" />
        <constant name="0042" />
        <constant name="Theoretische Physik" />
      </tuple>
      <tuple id="Hiwi_2">
        <constant name="22" />
        <constant name="9001" />
        <constant name="DBIS" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.2: Beispiel ChaTEAU - Q_1

```
<input>
  <schema>
    <relations>
      <relation name="Student" tag="S">
        <attribute name="TID#Student" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="name" type="string"/>
        <attribute name="vorname" type="string"/>
        <attribute name="studiengang" type="string"/>
        <attribute name="strasse" type="string"/>
        <attribute name="hausnummer" type="int"/>
        <attribute name="plz" type="int"/>
      </relation>
      <relation name="Note" tag="S">
        <attribute name="TID#Note" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
        <attribute name="modulnummer" type="int"/>
        <attribute name="note" type="double"/>
      </relation>
      <relation name="Hiwi" tag="S">
        <attribute name="TID#Hiwi" type="int"/>
        <attribute name="matrikelnummer" type="int"/>
```

```
      <attribute name="lehrstuhl" type="string"/>
    </relation>
    <relation name="Ergebnis2" tag="T">
      <attribute name="matrikelnummer" type="int"/>
      <attribute name="name" type="string"/>
      <attribute name="vorname" type="string"/>
      <attribute name="lehrstuhl" type="string"/>
      <attribute name="strasse" type="string"/>
      <attribute name="hausnummer" type="int"/>
      <attribute name="plz" type="int"/>
      <attribute name="TID#Student" type="int"/>
      <attribute name="TID#Hiwi" type="int"/>
    </relation>
  </relations>
  <dependencies>
    <sttgd>
      <body>
        <atom name="Student">
          <variable name="#V_TID_1"/>
          <variable name="#V_matrikelnummer_1" />
          <variable name="#V_name_1" />
          <variable name="#V_vorname_1" />
          <variable name="#V_studiengang_1" />
          <variable name="#V_strasse_1" />
          <variable name="#V_hausnummer_1" />
          <variable name="#V_plz_1" />
        </atom>
        <atom name="Hiwi">
          <variable name="#V_TID_2"/>
          <variable name="#V_matrikelnummer_1" />
          <variable name="#V_lehrstuhl_1" />
        </atom>
      </body>
      <head>
        <atom name="Ergebnis2">
          <variable name="#V_matrikelnummer_1" />
          <variable name="#V_name_1" />
          <variable name="#V_vorname_1" />
          <variable name="#V_lehrstuhl_1" />
          <variable name="#V_strasse_1" />
          <variable name="#V_hausnummer_1" />
          <variable name="#V_plz_1" />
          <variable name="#V_TID_1"/>
          <variable name="#V_TID_2"/>

        </atom>
      </head>
    </sttgd>
  </dependencies>
</schema>
<instance>
```

```
<relationalAtom name="Student">
  <tuple id="Student_1">
    <constant name="01" />
    <constant name="0042" />
    <constant name="Adam" />
    <constant name="Doug" />
    <constant name="Physik" />
    <constant name="Lange Strasse" />
    <constant name="89" />
    <constant name="18146" />
  </tuple>
  <tuple id="Student_2">
    <constant name="02" />
    <constant name="3200" />
    <constant name="Buches" />
    <constant name="Georg" />
    <constant name="Germanistik" />
    <constant name="Neue Strasse" />
    <constant name="4" />
    <constant name="18146" />
  </tuple>
  <tuple id="Student_3">
    <constant name="03" />
    <constant name="4711" />
    <constant name="Maier" />
    <constant name="Ronald" />
    <constant name="Informatik" />
    <constant name="Pappelallee" />
    <constant name="9" />
    <constant name="18146" />
  </tuple>
  <tuple id="Student_4">
    <constant name="04" />
    <constant name="9001" />
    <constant name="Kuhlmann" />
    <constant name="Goetz" />
    <constant name="Informatik" />
    <constant name="Schillerstrasse" />
    <constant name="96" />
    <constant name="18146" />
  </tuple>
</relationalAtom>
<relationalAtom name="Note">
  <tuple id="Note_1">
    <constant name="11" />
    <constant name="0042" />
    <constant name="00001" />
    <constant name="1.3" />
  </tuple>
  <tuple id="Note_2">
    <constant name="12" />
```

```
        <constant name="3200" />
        <constant name="10001" />
        <constant name="2.0" />
      </tuple>
      <tuple id="Note_3">
        <constant name="13" />
        <constant name="4711" />
        <constant name="20001" />
        <constant name="1.0" />
      </tuple>
      <tuple id="Note_4">
        <constant name="14" />
        <constant name="4711" />
        <constant name="20002" />
        <constant name="1.7" />
      </tuple>
      <tuple id="Note_5">
        <constant name="15" />
        <constant name="9001" />
        <constant name="20001" />
        <constant name="1.3" />
      </tuple>
      <tuple id="Note_6">
        <constant name="16" />
        <constant name="9001" />
        <constant name="20003" />
        <constant name="1.0" />
      </tuple>
    </relationalAtom>
    <relationalAtom name="Hiwi">
      <tuple id="Hiwi_1">
        <constant name="21" />
        <constant name="0042" />
        <constant name="Theoretische Physik" />
      </tuple>
      <tuple id="Hiwi_2">
        <constant name="22" />
        <constant name="9001" />
        <constant name="DBIS" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.3: Beispiel ChaTEAU - Q_2

A.2 Dateien der Zwischenergebnisse

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
```

```
<schema>
  <relations>
    <relation name="StudHilfskraft">
      <attribute name="TID#StudHilfskraft" />
      <attribute name="matrikelnummer" />
      <attribute name="kostenstelle" />
    </relation>
    <relation name="Student2">
      <attribute name="TID#Student2" />
      <attribute name="matrikelnummer" />
      <attribute name="studiengang" />
      <attribute name="strasse" />
      <attribute name="hausnummer" />
      <attribute name="plz" />
    </relation>
    <relation name="Note2">
      <attribute name="TID#Note2" />
      <attribute name="matrikelnummer" />
      <attribute name="modulnummer" />
      <attribute name="note" />
    </relation>
  </relations>
  <dependencies />
</schema>
<instance>
  <relationalAtom name="StudHilfskraft">
    <tuple>
      <constant name="21" />
      <constant name="42" />
      <constant name="Theoretische Physik" />
    </tuple>
    <tuple>
      <constant name="22" />
      <constant name="9001" />
      <constant name="DBIS" />
    </tuple>
  </relationalAtom>
  <relationalAtom name="Student2">
    <tuple>
      <constant name="1" />
      <constant name="42" />
      <constant name="Physik" />
      <constant name="Lange Strasse" />
      <constant name="89" />
      <constant name="18146" />
    </tuple>
    <tuple>
      <constant name="2" />
      <constant name="3200" />
      <constant name="Germanistik" />
      <constant name="Neue Strasse" />
```

```
        <constant name="4" />
        <constant name="18146" />
      </tuple>
      <tuple>
        <constant name="3" />
        <constant name="4711" />
        <constant name="Informatik" />
        <constant name="Pappelallee" />
        <constant name="9" />
        <constant name="18146" />
      </tuple>
      <tuple>
        <constant name="4" />
        <constant name="9001" />
        <constant name="Informatik" />
        <constant name="Schillerstrasse" />
        <constant name="96" />
        <constant name="18146" />
      </tuple>
    </relationalAtom>
    <relationalAtom name="Note2">
      <tuple>
        <constant name="13" />
        <constant name="4711" />
        <constant name="20001" />
        <constant name="1.0" />
      </tuple>
      <tuple>
        <constant name="16" />
        <constant name="9001" />
        <constant name="20003" />
        <constant name="1.0" />
      </tuple>
      <tuple>
        <constant name="12" />
        <constant name="3200" />
        <constant name="10001" />
        <constant name="2.0" />
      </tuple>
      <tuple>
        <constant name="11" />
        <constant name="42" />
        <constant name="1" />
        <constant name="1.3" />
      </tuple>
      <tuple>
        <constant name="15" />
        <constant name="9001" />
        <constant name="20001" />
        <constant name="1.3" />
      </tuple>
```

```
      <tuple>
        <constant name="14" />
        <constant name="4711" />
        <constant name="20002" />
        <constant name="1.7" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.4: Beispiel ChaTEAU - Ergebnis der Evolution

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
  <schema>
    <relations>
      <relation name="Hiwi">
        <attribute name="TID#Hiwi" />
        <attribute name="matrikelnummer" />
        <attribute name="lehrstuhl" />
      </relation>
      <relation name="Student">
        <attribute name="TID#Student" />
        <attribute name="matrikelnummer" />
        <attribute name="name" />
        <attribute name="vorname" />
        <attribute name="studiengang" />
        <attribute name="strasse" />
        <attribute name="hausnummer" />
        <attribute name="plz" />
      </relation>
      <relation name="Note">
        <attribute name="TID#Note" />
        <attribute name="matrikelnummer" />
        <attribute name="modulnummer" />
        <attribute name="note" />
      </relation>
    </relations>
    <dependencies />
  </schema>
  <instance>
    <relationalAtom name="Hiwi">
      <tuple>
        <constant name="21" />
        <constant name="42" />
        <constant name="Theoretische Physik" />
      </tuple>
      <tuple>
        <constant name="22" />
        <constant name="9001" />
        <constant name="DBIS" />
```

```
    </tuple>
  </relationalAtom>
  <relationalAtom name="Student">
    <tuple>
      <constant name="3" />
      <constant name="4711" />
      <constant name="#N_name_4" />
      <constant name="#N_vorname_4" />
      <constant name="Informatik" />
      <constant name="Pappelallee" />
      <constant name="9" />
      <constant name="18146" />
    </tuple>
    <tuple>
      <constant name="1" />
      <constant name="42" />
      <constant name="#N_name_3" />
      <constant name="#N_vorname_3" />
      <constant name="Physik" />
      <constant name="Lange Strasse" />
      <constant name="89" />
      <constant name="18146" />
    </tuple>
    <tuple>
      <constant name="4" />
      <constant name="9001" />
      <constant name="#N_name_1" />
      <constant name="#N_vorname_1" />
      <constant name="Informatik" />
      <constant name="Schillerstrasse" />
      <constant name="96" />
      <constant name="18146" />
    </tuple>
    <tuple>
      <constant name="2" />
      <constant name="3200" />
      <constant name="#N_name_2" />
      <constant name="#N_vorname_2" />
      <constant name="Germanistik" />
      <constant name="Neue Strasse" />
      <constant name="4" />
      <constant name="18146" />
    </tuple>
  </relationalAtom>
  <relationalAtom name="Note">
    <tuple>
      <constant name="11" />
      <constant name="42" />
      <constant name="1" />
      <constant name="1.3" />
    </tuple>
```

```
      <tuple>
        <constant name="14" />
        <constant name="4711" />
        <constant name="20002" />
        <constant name="1.7" />
      </tuple>
      <tuple>
        <constant name="16" />
        <constant name="9001" />
        <constant name="20003" />
        <constant name="1.0" />
      </tuple>
      <tuple>
        <constant name="15" />
        <constant name="9001" />
        <constant name="20001" />
        <constant name="1.3" />
      </tuple>
      <tuple>
        <constant name="12" />
        <constant name="3200" />
        <constant name="10001" />
        <constant name="2.0" />
      </tuple>
      <tuple>
        <constant name="13" />
        <constant name="4711" />
        <constant name="20001" />
        <constant name="1.0" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.5: Beispiel ChaTEAU - Inverse der Evolution

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
  <schema>
    <relations>
      <relation name="Ergebnis1">
        <attribute name="matrikelnummer" />
        <attribute name="modulnummer" />
        <attribute name="note" />
        <attribute name="TID#Studenten" />
        <attribute name="TID#Note" />
      </relation>
    </relations>
    <dependencies />
  </schema>
  <instance>
```

```
    <relationalAtom name="Ergebnis1">
      <tuple>
        <constant name="3200" />
        <constant name="10001" />
        <constant name="2.0" />
        <constant name="2" />
        <constant name="12" />
      </tuple>
      <tuple>
        <constant name="4711" />
        <constant name="20002" />
        <constant name="1.7" />
        <constant name="3" />
        <constant name="14" />
      </tuple>
      <tuple>
        <constant name="4711" />
        <constant name="20001" />
        <constant name="1.0" />
        <constant name="3" />
        <constant name="13" />
      </tuple>
      <tuple>
        <constant name="9001" />
        <constant name="20001" />
        <constant name="1.3" />
        <constant name="4" />
        <constant name="15" />
      </tuple>
      <tuple>
        <constant name="9001" />
        <constant name="20003" />
        <constant name="1.0" />
        <constant name="4" />
        <constant name="16" />
      </tuple>
      <tuple>
        <constant name="42" />
        <constant name="1" />
        <constant name="1.3" />
        <constant name="1" />
        <constant name="11" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.6: Beispiel ChaTEAU - Ergebnis der Anfrage Q_1

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
```

```
<schema>
  <relations>
    <relation name="Hiwi">
      <attribute name="TID#Hiwi" />
      <attribute name="matrikelnummer" />
      <attribute name="lehrstuhl" />
    </relation>
    <relation name="Student">
      <attribute name="TID#Student" />
      <attribute name="matrikelnummer" />
      <attribute name="name" />
      <attribute name="vorname" />
      <attribute name="studiengang" />
      <attribute name="strasse" />
      <attribute name="hausnummer" />
      <attribute name="plz" />
    </relation>
    <relation name="Note">
      <attribute name="TID#Note" />
      <attribute name="matrikelnummer" />
      <attribute name="modulnummer" />
      <attribute name="note" />
    </relation>
  </relations>
  <dependencies />
</schema>
<instance>
  <relationalAtom name="Student">
    <tuple>
      <constant name="1" />
      <constant name="42" />
      <constant name="#N_name_2" />
      <constant name="#N_vorname_2" />
      <constant name="#N_studiengang_2" />
      <constant name="#N_strasse_2" />
      <constant name="#N_hausnummer_2" />
      <constant name="#N_plz_2" />
    </tuple>
    <tuple>
      <constant name="2" />
      <constant name="3200" />
      <constant name="#N_name_3" />
      <constant name="#N_vorname_3" />
      <constant name="#N_studiengang_3" />
      <constant name="#N_strasse_3" />
      <constant name="#N_hausnummer_3" />
      <constant name="#N_plz_3" />
    </tuple>
    <tuple>
      <constant name="3" />
      <constant name="4711" />
```

```
        <constant name="#N_name_4" />
        <constant name="#N_vorname_4" />
        <constant name="#N_studiengang_4" />
        <constant name="#N_strasse_4" />
        <constant name="#N_hausnummer_4" />
        <constant name="#N_plz_4" />
      </tuple>
      <tuple>
        <constant name="4" />
        <constant name="9001" />
        <constant name="#N_name_1" />
        <constant name="#N_vorname_1" />
        <constant name="#N_studiengang_1" />
        <constant name="#N_strasse_1" />
        <constant name="#N_hausnummer_1" />
        <constant name="#N_plz_1" />
      </tuple>
      <tuple>
        <constant name="3" />
        <constant name="4711" />
        <constant name="#N_name_6" />
        <constant name="#N_vorname_6" />
        <constant name="#N_studiengang_6" />
        <constant name="#N_strasse_6" />
        <constant name="#N_hausnummer_6" />
        <constant name="#N_plz_6" />
      </tuple>
      <tuple>
        <constant name="4" />
        <constant name="9001" />
        <constant name="#N_name_5" />
        <constant name="#N_vorname_5" />
        <constant name="#N_studiengang_5" />
        <constant name="#N_strasse_5" />
        <constant name="#N_hausnummer_5" />
        <constant name="#N_plz_5" />
      </tuple>
    </relationalAtom>
    <relationalAtom name="Note">
      <tuple>
        <constant name="11" />
        <constant name="42" />
        <constant name="1" />
        <constant name="1.3" />
      </tuple>
      <tuple>
        <constant name="14" />
        <constant name="4711" />
        <constant name="20002" />
        <constant name="1.7" />
      </tuple>
```

```
      <tuple>
        <constant name="16" />
        <constant name="9001" />
        <constant name="20003" />
        <constant name="1.0" />
      </tuple>
      <tuple>
        <constant name="15" />
        <constant name="9001" />
        <constant name="20001" />
        <constant name="1.3" />
      </tuple>
      <tuple>
        <constant name="12" />
        <constant name="3200" />
        <constant name="10001" />
        <constant name="2.0" />
      </tuple>
      <tuple>
        <constant name="13" />
        <constant name="4711" />
        <constant name="20001" />
        <constant name="1.0" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.7: Beispiel ChaTEAU - Inverse der Anfrage Q_1

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
  <schema>
    <relations>
      <relation name="Ergebnis2">
        <attribute name="matrikelnummer" />
        <attribute name="name" />
        <attribute name="vorname" />
        <attribute name="lehrstuhl" />
        <attribute name="strasse" />
        <attribute name="hausnummer" />
        <attribute name="plz" />
        <attribute name="TID#Student" />
        <attribute name="TID#Hiwi" />
      </relation>
    </relations>
    <dependencies />
  </schema>
  <instance>
    <relationalAtom name="Ergebnis2">
      <tuple>
```

```
        <constant name="42" />
        <constant name="Adam" />
        <constant name="Doug" />
        <constant name="Theoretische Physik" />
        <constant name="Lange Strasse" />
        <constant name="89" />
        <constant name="18146" />
        <constant name="1" />
        <constant name="21" />
      </tuple>
      <tuple>
        <constant name="9001" />
        <constant name="Kuhlmann" />
        <constant name="Goetz" />
        <constant name="DBIS" />
        <constant name="Schillerstrasse" />
        <constant name="96" />
        <constant name="18146" />
        <constant name="4" />
        <constant name="22" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.8: Beispiel ChaTEAU - Ergebnis der Anfrage Q_2

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
  <schema>
    <relations>
      <relation name="Hiwi">
        <attribute name="TID#Hiwi" />
        <attribute name="matrikelnummer" />
        <attribute name="lehrstuhl" />
      </relation>
      <relation name="Student">
        <attribute name="TID#Student" />
        <attribute name="matrikelnummer" />
        <attribute name="name" />
        <attribute name="vorname" />
        <attribute name="studiengang" />
        <attribute name="strasse" />
        <attribute name="hausnummer" />
        <attribute name="plz" />
      </relation>
      <relation name="Note">
        <attribute name="TID#Note" />
        <attribute name="matrikelnummer" />
        <attribute name="modulnummer" />
        <attribute name="note" />
```

```
    </relation>
  </relations>
  <dependencies />
</schema>
<instance>
  <relationalAtom name="Student">
    <tuple>
      <constant name="1" />
      <constant name="42" />
      <constant name="#N_name_2" />
      <constant name="#N_vorname_2" />
      <constant name="#N_studiengang_2" />
      <constant name="#N_strasse_2" />
      <constant name="#N_hausnummer_2" />
      <constant name="#N_plz_2" />
    </tuple>
    <tuple>
      <constant name="2" />
      <constant name="3200" />
      <constant name="#N_name_3" />
      <constant name="#N_vorname_3" />
      <constant name="#N_studiengang_3" />
      <constant name="#N_strasse_3" />
      <constant name="#N_hausnummer_3" />
      <constant name="#N_plz_3" />
    </tuple>
    <tuple>
      <constant name="3" />
      <constant name="4711" />
      <constant name="#N_name_4" />
      <constant name="#N_vorname_4" />
      <constant name="#N_studiengang_4" />
      <constant name="#N_strasse_4" />
      <constant name="#N_hausnummer_4" />
      <constant name="#N_plz_4" />
    </tuple>
    <tuple>
      <constant name="4" />
      <constant name="9001" />
      <constant name="#N_name_1" />
      <constant name="#N_vorname_1" />
      <constant name="#N_studiengang_1" />
      <constant name="#N_strasse_1" />
      <constant name="#N_hausnummer_1" />
      <constant name="#N_plz_1" />
    </tuple>
    <tuple>
      <constant name="3" />
      <constant name="4711" />
      <constant name="#N_name_6" />
      <constant name="#N_vorname_6" />
```

```
        <constant name="#N_studiengang_6" />
        <constant name="#N_strasse_6" />
        <constant name="#N_hausnummer_6" />
        <constant name="#N_plz_6" />
      </tuple>
      <tuple>
        <constant name="4" />
        <constant name="9001" />
        <constant name="#N_name_5" />
        <constant name="#N_vorname_5" />
        <constant name="#N_studiengang_5" />
        <constant name="#N_strasse_5" />
        <constant name="#N_hausnummer_5" />
        <constant name="#N_plz_5" />
      </tuple>
    </relationalAtom>
    <relationalAtom name="Note">
      <tuple>
        <constant name="11" />
        <constant name="42" />
        <constant name="1" />
        <constant name="1.3" />
      </tuple>
      <tuple>
        <constant name="14" />
        <constant name="4711" />
        <constant name="20002" />
        <constant name="1.7" />
      </tuple>
      <tuple>
        <constant name="16" />
        <constant name="9001" />
        <constant name="20003" />
        <constant name="1.0" />
      </tuple>
      <tuple>
        <constant name="15" />
        <constant name="9001" />
        <constant name="20001" />
        <constant name="1.3" />
      </tuple>
      <tuple>
        <constant name="12" />
        <constant name="3200" />
        <constant name="10001" />
        <constant name="2.0" />
      </tuple>
      <tuple>
        <constant name="13" />
        <constant name="4711" />
        <constant name="20001" />
```

```
        <constant name="1.0" />
      </tuple>
    </relationalAtom>
  </instance>
</input>
```

Quellcodeausschnitt A.9: Beispiel ChaTEAU - Inverse der Anfrage Q_2

A.3 Ausgabe ChaTEAU

```
-Constraints:
S-T TGD:
Student(#V_TID_1, #V_matrikelnummer_1, #V_name_1, #V_vorname_1,
#V_studiengang_1, #V_strasse_1, #V_hausnummer_1, #V_plz_1)
->
Student2(#V_TID_1, #V_matrikelnummer_1, #V_studiengang_1, #V_strasse_1,
#V_hausnummer_1, #V_plz_1)

S-T TGD:
Note(#V_TID_1, #V_matrikelnummer_1, #V_modulnummer_1, #V_note_1)
->
Note2(#V_TID_1, #V_matrikelnummer_1, #V_modulnummer_1, #V_note_1)

S-T TGD:
Hiwi(#V_TID_1, #V_matrikelnummer_1, #V_lehrstuhl_1)
->
StudHilfskraft(#V_TID_1, #V_matrikelnummer_1, #V_lehrstuhl_1)

-Constraints:
S-T TGD:
Student(#V_TID_1, #V_matrikelnummer_1, #V_name_1, #V_vorname_1,
#V_studiengang_1, #V_strasse_1, #V_hausnummer_1, #V_plz_1),
Note(#V_TID_2, #V_matrikelnummer_1, #V_modulnummer_1, #V_note_1)
->
Ergebnis1(#V_matrikelnummer_1, #V_modulnummer_1, #V_note_1, #V_TID_1,
#V_TID_2)

-Constraints:
S-T TGD:
Hiwi(#V_TID_2, #V_matrikelnummer_1, #V_lehrstuhl_1),
Student(#V_TID_1, #V_matrikelnummer_1, #V_name_1, #V_vorname_1,
#V_studiengang_1, #V_strasse_1, #V_hausnummer_1, #V_plz_1)
->
Ergebnis2(#V_matrikelnummer_1, #V_name_1, #V_vorname_1, #V_lehrstuhl_1,
#V_strasse_1, #V_hausnummer_1, #V_plz_1, #V_TID_1, #V_TID_2)

no mapping found between Student(1, 42, #N_name_2, #N_vorname_2,
#N_studiengang_2, #N_strasse_2, #N_hausnummer_2, #N_plz_2) and Student(3,
4711, #N_name_4, #N_vorname_4, Informatik, Pappelallee, 9, 18146)
```

```
32 mapping found between Student(1, 42, #N_name_2, #N_vorname_2,
   #N_studiengang_2, #N_strasse_2, #N_hausnummer_2, #N_plz_2) and Student(1,
   42, #N_name_3, #N_vorname_3, Physik, Lange Strasse, 89, 18146)
33 no mapping found between Student(2, 3200, #N_name_3, #N_vorname_3,
   #N_studiengang_3, #N_strasse_3, #N_hausnummer_3, #N_plz_3) and Student(3,
   4711, #N_name_4, #N_vorname_4, Informatik, Pappelallee, 9, 18146)
34 no mapping found between Student(2, 3200, #N_name_3, #N_vorname_3,
   #N_studiengang_3, #N_strasse_3, #N_hausnummer_3, #N_plz_3) and Student(1,
   42, #N_name_3, #N_vorname_3, Physik, Lange Strasse, 89, 18146)
35 no mapping found between Student(2, 3200, #N_name_3, #N_vorname_3,
   #N_studiengang_3, #N_strasse_3, #N_hausnummer_3, #N_plz_3) and Student(4,
   9001, #N_name_1, #N_vorname_1, Informatik, Schillerstrasse, 96, 18146)
36 mapping found between Student(2, 3200, #N_name_3, #N_vorname_3,
   #N_studiengang_3, #N_strasse_3, #N_hausnummer_3, #N_plz_3) and Student(2,
   3200, #N_name_2, #N_vorname_2, Germanistik, Neue Strasse, 4, 18146)
37 mapping found between Student(3, 4711, #N_name_4, #N_vorname_4,
   #N_studiengang_4, #N_strasse_4, #N_hausnummer_4, #N_plz_4) and Student(3,
   4711, #N_name_4, #N_vorname_4, Informatik, Pappelallee, 9, 18146)
38 no mapping found between Student(4, 9001, #N_name_1, #N_vorname_1,
   #N_studiengang_1, #N_strasse_1, #N_hausnummer_1, #N_plz_1) and Student(3,
   4711, #N_name_4, #N_vorname_4, Informatik, Pappelallee, 9, 18146)
39 no mapping found between Student(4, 9001, #N_name_1, #N_vorname_1,
   #N_studiengang_1, #N_strasse_1, #N_hausnummer_1, #N_plz_1) and Student(1,
   42, #N_name_3, #N_vorname_3, Physik, Lange Strasse, 89, 18146)
40 mapping found between Student(4, 9001, #N_name_1, #N_vorname_1,
   #N_studiengang_1, #N_strasse_1, #N_hausnummer_1, #N_plz_1) and Student(4,
   9001, #N_name_1, #N_vorname_1, Informatik, Schillerstrasse, 96, 18146)
41 mapping found between Student(3, 4711, #N_name_6, #N_vorname_6,
   #N_studiengang_6, #N_strasse_6, #N_hausnummer_6, #N_plz_6) and Student(3,
   4711, #N_name_4, #N_vorname_4, Informatik, Pappelallee, 9, 18146)
42 no mapping found between Student(4, 9001, #N_name_5, #N_vorname_5,
   #N_studiengang_5, #N_strasse_5, #N_hausnummer_5, #N_plz_5) and Student(3,
   4711, #N_name_4, #N_vorname_4, Informatik, Pappelallee, 9, 18146)
43 no mapping found between Student(4, 9001, #N_name_5, #N_vorname_5,
   #N_studiengang_5, #N_strasse_5, #N_hausnummer_5, #N_plz_5) and Student(1,
   42, #N_name_3, #N_vorname_3, Physik, Lange Strasse, 89, 18146)
44 mapping found between Student(4, 9001, #N_name_5, #N_vorname_5,
   #N_studiengang_5, #N_strasse_5, #N_hausnummer_5, #N_plz_5) and Student(4,
   9001, #N_name_1, #N_vorname_1, Informatik, Schillerstrasse, 96, 18146)
45 mapping found between Note(11, 42, 1, 1.3) and Note(11, 42, 1, 1.3)
46 no mapping found between Note(14, 4711, 20002, 1.7) and Note(11, 42, 1,
   1.3)
47 mapping found between Note(14, 4711, 20002, 1.7) and Note(14, 4711, 20002,
   1.7)
48 no mapping found between Note(16, 9001, 20003, 1.0) and Note(11, 42, 1,
   1.3)
49 no mapping found between Note(16, 9001, 20003, 1.0) and Note(14, 4711,
   20002, 1.7)
50 mapping found between Note(16, 9001, 20003, 1.0) and Note(16, 9001, 20003,
   1.0)
```

```
51 no mapping found between Note(15, 9001, 20001, 1.3) and Note(11, 42, 1,
   1.3)
52 no mapping found between Note(15, 9001, 20001, 1.3) and Note(14, 4711,
   20002, 1.7)
53 no mapping found between Note(15, 9001, 20001, 1.3) and Note(16, 9001,
   20003, 1.0)
54 mapping found between Note(15, 9001, 20001, 1.3) and Note(15, 9001, 20001,
   1.3)
55 no mapping found between Note(12, 3200, 10001, 2.0) and Note(11, 42, 1,
   1.3)
56 no mapping found between Note(12, 3200, 10001, 2.0) and Note(14, 4711,
   20002, 1.7)
57 no mapping found between Note(12, 3200, 10001, 2.0) and Note(16, 9001,
   20003, 1.0)
58 no mapping found between Note(12, 3200, 10001, 2.0) and Note(15, 9001,
   20001, 1.3)
59 mapping found between Note(12, 3200, 10001, 2.0) and Note(12, 3200, 10001,
   2.0)
60 no mapping found between Note(13, 4711, 20001, 1.0) and Note(11, 42, 1,
   1.3)
61 no mapping found between Note(13, 4711, 20001, 1.0) and Note(14, 4711,
   20002, 1.7)
62 no mapping found between Note(13, 4711, 20001, 1.0) and Note(16, 9001,
   20003, 1.0)
63 no mapping found between Note(13, 4711, 20001, 1.0) and Note(15, 9001,
   20001, 1.3)
64 no mapping found between Note(13, 4711, 20001, 1.0) and Note(12, 3200,
   10001, 2.0)
65 mapping found between Note(13, 4711, 20001, 1.0) and Note(13, 4711, 20001,
   1.0)
66 mapping found between Hiwi(21, 42, Theoretische Physik) and Hiwi(21, 42,
   Theoretische Physik)
67 no mapping found between Hiwi(22, 9001, DBIS) and Hiwi(21, 42,
   Theoretische Physik)
68 mapping found between Hiwi(22, 9001, DBIS) and Hiwi(22, 9001, DBIS)
69 no mapping found between Student(1, 42, Adam, Doug, #N_studiengang_2,
   Lange Strasse, 89, 18146) and Student(3, 4711, #N_name_4, #N_vorname_4,
   Informatik, Pappelallee, 9, 18146)
70 no mapping found between Student(1, 42, Adam, Doug, #N_studiengang_2,
   Lange Strasse, 89, 18146) and Student(1, 42, #N_name_3, #N_vorname_3,
   Physik, Lange Strasse, 89, 18146)
71 no mapping found between Student(1, 42, Adam, Doug, #N_studiengang_2,
   Lange Strasse, 89, 18146) and Student(4, 9001, #N_name_1, #N_vorname_1,
   Informatik, Schillerstrasse, 96, 18146)
72 no mapping found between Student(1, 42, Adam, Doug, #N_studiengang_2,
   Lange Strasse, 89, 18146) and Student(2, 3200, #N_name_2, #N_vorname_2,
   Germanistik, Neue Strasse, 4, 18146)
73 no mapping found between Student(4, 9001, Kuhlmann, Goetz,
   #N_studiengang_1, Schillerstrasse, 96, 18146) and Student(3, 4711,
   #N_name_4, #N_vorname_4, Informatik, Pappelallee, 9, 18146)
```

```
74 no mapping found between Student(4, 9001, Kuhlmann, Goetz,
   #N_studiengang_1, Schillerstrasse, 96, 18146) and Student(1, 42,
   #N_name_3, #N_vorname_3, Physik, Lange Strasse, 89, 18146)
75 no mapping found between Student(4, 9001, Kuhlmann, Goetz,
   #N_studiengang_1, Schillerstrasse, 96, 18146) and Student(4, 9001,
   #N_name_1, #N_vorname_1, Informatik, Schillerstrasse, 96, 18146)
76 no mapping found between Student(4, 9001, Kuhlmann, Goetz,
   #N_studiengang_1, Schillerstrasse, 96, 18146) and Student(2, 3200,
   #N_name_2, #N_vorname_2, Germanistik, Neue Strasse, 4, 18146)
77 [1, 4]
```

Anhang B

Aufbau des Datenträgers

Auf dem beiliegenden Datenträger sind die mit der Arbeit zusammenhängenden Daten hinterlegt. Dies umfasst unter anderem Testdatensätze, Abbildungen und der vollständige Code der Implementierung. Der Aufbau des Datenträgers wird im Folgenden kurz erklärt. Die Ordnernamen dienen hier als Überschriften.

Arbeit

- Digitale Fassung der Arbeit (Masterarbeit_Florian_Rose.pdf)
- Tex-Dateien der Arbeit
- verwendete Abbildungen (*Media*-Ordner)
- Beispieldateien aus Anhang A (*Example*-Ordner)

Code

- Projektordner von ChaTEAU zum Standpunkt der Abgabe

Literatur

Dieser Ordner enthält die im Literaturverzeichnis aufgeführten Quellen, welche digital verfügbar sind, und Webseiten, die in der Arbeit zitiert wurden (*Online*-Ordner). Der Dateiname entspricht dem Kürzel im Literaturverzeichnis. **Die PDF-Dateien der Literatur dienen der Nachvollziehbarkeit. Sie dürfen nicht öffentlich bereitgestellt werden.**

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 2020-09-23