Der BACKCHASE zur Unterstützung von Data Provenance und Schema-Evolution

Florian Rose
Lehrstuhl für Datenbank- und Informationssysteme
Institut für Informatik
Universität Rostock
Florian.Rose@uni-rostock.de

Andreas Heuer
Lehrstuhl für Datenbank- und Informationssysteme
Institut für Informatik
Universität Rostock
Andreas.Heuer@uni-rostock.de

KURZFASSUNG

Der CHASE-Algorithmus ist ein Multifunktionswerkzeug der Datenbankforschung. Ursprünglich für die Optimierung des Datenbankentwurfs genutzt, wird er heutzutage auch für den Datenaustausch und Data-Cleaning verwendet und kann ebenfalls genutzt werden, um Anfragen zu optimieren. Für einige Anwendungen reicht eine CHASE-Phase alleine jedoch nicht aus. Teilweise wird eine Nachbearbeitung, eine sogenannte BACKCHASE-Phase, benötigt. Dies ist beispielsweise der Fall, wenn Auswertungen zum Zweck der Rückverfolgbarkeit (Provenance) invertiert werden sollen.
In dieser Arbeit fassen wir zusammen, wie der Ansatz des CHASE und BACKCHASE verwendet werden kann, um die Why-Provenance von Anfragen unter Schema-Evolution zu gewährleisten.

Stichwörter

CHASE, BACKCHASE, Data Provenance, Schema-Evolution

1. MOTIVATION

Die Rückverfolgbarkeit von Daten bei wissenschaftlichen Auswertungen bietet die Grundlage der Nachvollziehbarkeit von veröffentlichten Statistiken, ein wichtiges Qualitätsmerkmal der wissenschaftlichen Praxis. Für das Forschungsdatenmanagement ist die Provenance der Auswertungsdaten daher von großem Interesse. Allerdings unterliegen die Strukturen solcher Datenbanken ebenfalls Schema-Evolutionen [4]. Dies kann die Reproduzierbarkeit von Auswertungen, die unter der letzten Schemaversion durchgeführt wurden, beeinträchtigen, z. B. falls ein Attribut entfernt wird, welches Teil der Ergebnisrelation für diese Auswertung war. Offensichtlich ist es ohne dieses Attribut nicht möglich, das vorangegangene Anfrageergebnis mit den zugrundeliegenden Daten zu reproduzieren. Folglich muss vor der Überführung in die nächste Schemaversion sichergestellt werden, dass die an der Anfrage beteiligten Tupel in der aktuellen Schemaversion "eingefroren" werden. Statt hierfür jedoch eine Kopie des gesamten Datenbestandes anzulegen, ist es wünschenswert, nur genau diejenigen Tupel in der aktuellen Schemaversion aufzuheben, welche an einer Anfrage beteiligt waren, deren Reproduzierbarkeit vorausgesetzt wird. Weiter soll dies auch nur passieren, falls die Schema-Evolution dafür sorgt, dass das entsprechende Tupel in der nächsten Version hierfür nicht ausreichend abgebildet wird.
Der CHASE-Algorithmus (siehe Abschnitt 2.1) bietet sich hier an, da er direkt zur Bildung der Anfrageergebnisse und neuen Schemaversion genutzt werden kann [1], sodass andere CHASE-Schritte zur Berechnung der aufzuhebenden Tupel direkt dort ansetzen können. Darüber hinaus sind Evolutionsschritte und Anfragen für den CHASE vom gleichen Parametertyp, sodass algorithmisch nicht zwischen ihnen unterschieden werden muss.

Wir werden im folgenden Abschnitt 2 den Stand der Forschung in den Bereichen CHASE, BACKCHASE, Data Provenance und Schema-Evolution zusammenfassen. Danach werden wir in Abschnitt 3 die Nutzung des BACKCHASE für die Data Provenance, speziell die Why-Provenance, und der Schema-Evolutionen vorstellen. In Abschnitt 4 werden wir die den vorgestellten Ansatz hinsichtlich der Terminierung und der Erweiterung des Standard-CHASE untersuchen, bevor wir in Abschnitt 5 abschließend offene Fragestellungen skizzieren.

2. STAND DER FORSCHUNG

2.1 CHASE & BACKCHASE

Der CHASE-Algorithmus ging aus der Datenbankforschung als Werkzeug zum Schlussfolgern mit Integritätsbedingungen hervor. Dabei wird eine Menge von Abhängigkeiten Γ, der CHASE-Parameter, auf ein CHASE-Objekt d angewandt und erweitert dies durch Forward-Chaining so, dass d dem CHASE-Parameter Γ genügt [5]. Grundlegend werden zwei Arten von Abhängigkeiten, die *equality-generating dependencies* (EGDs) und *tuple-generating dependencies* (TGDs), unterschieden [13]. Eine EGD stellt die Gleichheit bestimmter Werte sicher und ist eine Formel der Form

$$\forall \vec{x} : F(\vec{x}) \rightarrow x_1 = x_2.$$

Hierbei ist $\vec{x}$ ein Variablenvektor mit $x_1, ..., x_n$ als Komponenten. F ist eine Konjunktion von Prädikaten der Form $R_i(\vec{x})$, wobei R_i Relationenschemata des Datenbankschemas sind. TGDs hingegen rufen die Generierung neuer Tupel hervor und sind Formeln der Form

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

32^{nd} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), September 01-03, 2021, Munich, Germany.

Hierbei muss in F_1 jede Komponente aus $\vec{x}$ mindestens einmal vorkommen. Eine besondere Form von TGDs sind *source-to-target-TGDs* (s-t-TGDs). Bei ihnen ist der Rumpf (linke Seite der Abhängigkeit) über einem Quellschema S und der Kopf (rechte Seite) über einem Zielschema S' definiert. Durch die Beschränkung auf Konjunktionen von Prädikaten, betrachten wir im Folgenden nur Anfragen und Evolutionsoperationen, die ausschließlich als Konjunktionen darstellbar sind. Disjunktionen lassen sich spätestens in der BACKCHASE-Phase nach der Invertierung nicht ohne Zusatzinformationen (Zeugenbasis einzelner Tupel) verarbeiten. Bei der Anwendung des CHASE (siehe Algorithmus 1) wird dann im CHASE-Objekt nach Mustern des Rumpfes jeder Abhängigkeit gesucht. Ein solcher *Trigger* ist *aktiv*, falls der Kopf seiner entsprechenden Abhängigkeit noch nicht im CHASE-Objekt erfüllt ist. Da sich diverse Bedingungen als

Algorithmus 1: Standard-CHASE auf CHASE-Objekt d

```
if τ ist aktiv then
    if τ hat Form ∀x⃗ : F1(x⃗) → ∃y⃗ : F2(x⃗, y⃗) then
        Nimm Kopf von τ in d auf
    else if τ hat Form ∀x⃗ : F(x⃗) → x1 = x2 then
        if x1, x2 sind Konstanten und x1 ≠ x2 then
            CHASE schlägt fehl
        else if x2 ist Konstante then
            x1 ← x2
        else
            x2 ← x1
```

EGDs bzw. TGDs formulieren lassen und die Berechnung deren Implikation auf Instanzen oder Anfragen verschiedene Ziele erfüllen kann, ist der CHASE vielseitig einsetzbar. So lassen sich unter anderem (1) Datenmigrationen als s-t-TGDs für den CHASE auf einer Datenbankinstanz formulieren [10], (2) Data-Cleaning-Probleme durch Anwendung des CHASE auf einer Datenbankinstanz mit bestimmten EGDs zur Bereinigung lösen [12], (3) Anfragen durch Sichten beantworten, indem man den CHASE mit den Sichtdefinitionen auf der Anfrage als CHASE-Objekt ausführt [9], sowie (4) Anfragen semantisch optimieren, indem man Integritätsbedingungen als CHASE-Parameter in die Anfrage einarbeitet [15].

Für die Anwendungsfälle (3) und (4) reicht eine einzelne CHASE-Phase jedoch nicht aus. Die CHASE-Phase berechnet hier ein Zwischenergebnis, welches für die BACKCHASE-Phase, in diesem Fall eine zweite Anwendung des CHASE mit modifizierten Abhängigkeiten, verwendet wird [9]. Die BACKCHASE-Phase muss aber nicht zwingend eine zweite CHASE-Phase sein [16]. Je nach Anwendungsfall können auch hier verschiedene Techniken Ansatz finden.

In Abbildung 1 zeigen wir schematisch den Einsatz des BACKCHASE für die Anwendungsfälle (3) und (4): Im Anwendungsfall (3) wird eine Anfrage Q an Basisrelationen einer Datenbanken um Zugriffe auf verwendbare Sichten angereichert, die CHASE-Parameter enthalten dann die Sichtdefinitionen. Die BACKCHASE-Phase minimiert nun diese erweiterte Anfrage und schränkt sie (falls möglich) ausschließlich auf Zugriffe über die Sichtrelationen ein (Anfrage Q'). In der BACKCHASE-Phase wird der CHASE mit inversen Sichtdefinitionen angewendet [9]. Im Anwendungsfall 4 werden Integritätsbedingungen als CHASE-Parameter in

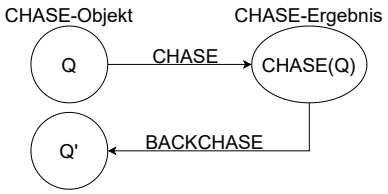

Abbildung 1: CHASE und BACKCHASE

die Anfrage Q eingearbeitet. Die BACKCHASE-Phase minimiert dann diese Anfrage, indem sie überflüssige Operationen (etwa Verbunde) eliminiert. In diesem Anwendungsfall ist die BACKCHASE-Phase ein durch homomorphe Abbildungen der Anfrage gesteuerter, äquivalenzerhaltender Reduktionsprozess [15]. Die BACKCHASE-Phase wurde für die Anwendungsfälle (1) und (2) bisher nicht benötigt. Wir werden sie in Abschnitt 3 aber einführen. Hier wollen wir das Anfrageergebnis einer Anfrage, verstanden als Datenbanktransformation wie in (1), mit Hilfe der Why-Provenance auf die minimale Zeugenbasis, also die relevanten Originaltupel aus der Datenbank, zurückführen. Der Schritt der Why-Provenance wird durch eine erweiterte inverse Anfrage dargestellt, die wir als BACKCHASE-Phase verstehen. Analog werden wir eine Schema-Evolution als eine Datenbanktransformation wie in (1) verstehen. Um die neue Datenbank nach der Evolution in die alte Darstellung vor der Evolution rückrechnen zu können, müssen wir auch hier die Evolution in einer BACKCHASE-Phase invertieren.

2.2 Data Provenance

Die Provenance beschäftigt sich mit der Frage nach dem Ursprung von Daten. Insbesondere ist dies für die Nachvollziehbarkeit von wissenschaftlichen Auswertungen interessant. Hauptsächlich unterscheidet man drei Abstraktionsstufen der Provenance, *Where-*, *Why-* und *How-*Provenance. Diese beschreiben in unterschiedlichem Detailgrad, woher die Ergebnisdaten stammen. Dies reicht von der Benennung der Datenquellen (Where), also den Relationen und Datenbankschemata [7], über die Angabe der beteiligten Tupel (Why), einer sogenannten Zeugenbasis [6], bis zu Provenance-Polynomen (How) [14], welche aussagen, wie die beteiligten Tupel kombiniert wurden.

Der CHASE kann genutzt werden, um ein Anfrageergebnis zu berechnen. Hierzu formuliert man die Anfrage als s-t-TGD und wendet sie mittels des CHASE auf die entsprechende Datenbankinstanz an. Aus dem Anfrageergebnis lassen sich die beteiligten Tupel errechnen, indem man die Inverse der Auswertung bildet. Hierzu führt man den CHASE auf der Ergebnisrelation mit der invertierten s-t-TGD der Anfrage aus. Je nach Art der Anfrage können sich unterschiedliche Arten von Inversen ergeben, welche mittels zusätzlicher Provenance-Annotationen noch verbessert werden können [2, 11]. Ziel ist es, eine möglichst genaue CHASE-Inverse zu erhalten, auch wenn Anfragen in der Regel verlustbehaftet sind und sich somit nicht jede Information rekonstruieren lässt.

2.3 Schema-Evolution

Die Evolution eines Datenbankschemas ist der Übergang von einer Schemaversion in eine neue. Diese sind durch Änderungen im Anwendungsgebiet begründet, z. B. neue rele-

vante Informationen, die zuvor nicht erhoben wurden, oder neue Anforderungen an die Anwendung [17]. Deshalb reichen die möglichen Operationen während einer Evolution vom simplen Hinzufügen eines neuen Attributs bis hin zu komplexen Berechnungen neuer Werte. Auch Evolutionsoperationen lassen sich als s-t-TGDs und EGDs darstellen. Hierbei ist das Quellschema die aktuelle Schemaversion und das Zielschema die Folgeversion. Sowohl Anfragen als auch Schema-Evolution liegen die Basisoperationen Selektion (Attribut gleich Attribut und Attribut gleich Konstante), Projektion, Verbund und die Umbenennung von Relationen und Attributen zugrunde. In beiden Fällen interessieren uns die CHASE-Inversen [11]. Diese drücken den Grad des Informationsverlustes aus, wenn wir versuchen, die Originalinstanz aus dem CHASE-Ergebnis durch Invertierung der Abhängigkeiten zu rekonstruieren. Dies ist für unseren Anwendungsfall von großer Relevanz, da die Berechnung der an der Anfrage beteiligten Tupel und ihrer Attribute Grundlage des in Abschnitt 3 vorgestellten Ansatzes ist. Dafür rekonstruieren wir die Teilinstanz unserer Originaldatenbank, müssen dabei aber darauf achten, dass keines der für die Anfrage bzw. Evolution relevanten Tupel bei der Rekonstruktion verloren geht. Daher benötigen wir bezüglich der an einer Anfrage bzw. Evolution beteiligten Tupel eine tupelerhaltend-relaxierte Inverse, d. h., dass die durch den BACKCHASE gewonnene Datenbankinstanz auf die Originalinstanz abbildbar ist und die gleiche Tupelzahl besitzt [2].

3. NUTZUNG DES BACKCHASE

Zunächst zeigen wir in Unterabschnitt 3.1, wie die oben genannten Basisoperatoren und komplexere Operationen für Anfragen und Evolutionen als Abhängigkeiten (s-t-TGDs), und somit als Parameter für den CHASE-Algorithmus, formuliert werden können. Wir verwenden hierbei eine Kurzschreibweise, bei der nur existenzquantifizierte Variablen explizit als solche angegeben werden. Die Namen der Variablen entsprechen den Namen der Attribute.

Danach werden wir in Unterabschnitt 3.2 dann die Basis für den BACKCHASE-Prozess legen, indem wir die Provenance- und Evolutionsschritte über mehrere Stufen hin invertieren und komponieren. Hierbei müssen wir insbesondere berücksichtigen, dass wir bei der Komposition verschiedener Schritte wertvolle Zwischeninformationen nicht verlieren. Die s-t-TGDs werden hier als Transformationssprache nicht mehr ausreichen, wir müssen auf Konstrukte der Prädikatenlogik zweiter Ordnung (so-TGDs, second-order TGDs) ausweichen [11].

3.1 Anfragen und Schema-Evolution als s-t-TGDs

Im Nachfolgenden zeigen wir am Beispiel der Projektion, wie sich Operatoren für Anfragen und Schema-Evolutionen als s-t-TGDs darstellen lassen. Weitere Details und Operatoren finden sich in [19].

Die Projektion lässt sich als s-t-TGD formulieren, indem nur die Attribute, auf welche projiziert werden soll, im Kopf der Abhängigkeit vorkommen, z. B.

$$R(A_1, A_2, A_3, ..., A_n) \rightarrow R'(A_1, A_2).$$

Da der Standard-CHASE jedoch nur für aktive Trigger den Kopf der Abhängigkeit in das Ergebnis der Anfrage bzw. Evolution aufnimmt, wird hier eine Duplikateliminierung vorgenommen, die nur eine relaxierte CHASE-Inverse bedeutet [2]. In Bezug auf die an der Anfrage beteiligten Tupel genügt dies jedoch einer tupel-erhaltend relaxierten Inverse, da Tupel, für die kein aktiver Trigger existiert, keinen Einfluss auf das Anfrageergebnis haben [18]. Die Inverse der obigen s-t-TGD lautet dann

$$R'(A_1, A_2) \rightarrow \exists A_3, ...A_n : R(A_1, A_2, A_3, ..., A_n)$$

Im Gegensatz zu Anfragen kann bei der Evolution jedoch noch ein Spezialfall auftreten. Da bei einer Schema-Evolution auch neue Attribute hinzukommen können, kann dort auch auf Attribute projiziert werden, die nicht im ursprünglichen Schema vorkommen. Für die Invertierung sind solche neu hinzugefügten Attribute jedoch unproblematisch, da sie keinen Einfluss auf die aus der Ursprungsrelation stammenden Informationen haben.

Die in [19] beschriebenen, grundlegenden Operatoren werden in Anfragen und Evolutionsoperationen häufig verknüpft und bieten somit die Grundlage der meisten bekannten Operationen zur Schema-Modifikation, die in [8] vorgestellt wurden. Dabei gilt, dass der weniger genaue Inversentyp dominiert [2, 11]. Hierbei muss jedoch in einem Fall besonders auf die Projektion geachtet werden. Für die Auswertung relevante Attribute, z. B. Verbundbedingungen, die nicht im Kopf der TGD vorkommen, müssen besonders markiert werden. Darauf gehen wir später in diesem Abschnitt genauer ein. Es können aber auch komplexere Auswertungen vorkommen, die z. B. Aggregate berechnen oder völlig neue Werte generieren. In diesen Fällen ist der Standard-CHASE nicht ausreichend, um die CHASE- oder BACKCHASE-Phase abzuschließen. Solche Auswertungen, lassen sich jedoch mit Hilfe von second-order-Abhängigkeiten umsetzen [11]. Hier können Funktionen definiert werden, welche die Generierung neuer Konstanten erlauben. Sind diese selbst nicht invertierbar, werden an den entsprechenden Stellen in der BACKCHASE-Phase Nullwerte generiert.

3.2 Komposition von Provenance- und Evolutionsschritten

Im Folgenden stellen wir einen Ansatz vor, welcher die Berechnung derjenigen Tupel erlaubt, welche nach einer Evolution nicht mehr ausreichend in der nächsten Schemaversion abgebildet werden, um bestimmte Anfragen zu reproduzieren. Hierzu setzen wir voraus, dass alle Tupel eine eindeutig ID (TID) besitzen und die Evolutionsschritte und Anfragen, die reproduzierbar bleiben sollen, gespeichert werden. Beim Übergang wird die Menge der Tupel berechnet, die nach der Evolution nicht mehr für die Beantwortung der ausgewählten Anfragen ausreichen. Diese Menge nennen wir im Folgenden I_{Prov} und muss nach der Schema-Evolution zusätzlich aufbewahrt werden.

Dieser Prozess läuft in drei Teilschritten ab. Zunächst berechnen wir mittels des CHASE und BACKCHASE für jede Anfrage, jeweils durch eine s-t-TGD abgebildet, ihr Ergebnis und die Menge der Tupel, die tatsächlich an der Auswertung beteiligt waren. Diese Mengen nennen wir $I'(S)$. Anschließend berechnen wir analog die Datenbank in der nächsten Schemaversion ($I(S')$), basierend auf den Evolutionsoperationen, die ebenfalls als s-t-TGDs dargestellt werden, sowie die Menge der Tupel J, die nach der Evolution durch Invertierung abbildbar sind. Im letzten Teilschritt suchen wir Abbildungen von Tupeln aus $I'(S)$ in J. Existiert eine sol-

che Abbildung für ein Tupel, ist es in der nachfolgenden Schemaversion ausreichend abgebildet, um für die Anfragen verwendet zu werden. Ist dies nicht der Fall, muss es zum Zweck der Reproduzierbarkeit aufbewahrt werden.

Provenance der Anfragen. In [1] wurde vorgestellt, wie eine Anfrage Q' auf der nächsten Schemaversion S' nach der Evolution E aussehen muss, um das Ergebnis der ursprünglichen Anfrage Q auf der alten Schemaversion S zu produzieren. Hierzu muss Q' der Ausführung von Q auf dem alten Datenbestand entsprechen. Es gilt also

$$Q'(S') = Q(E^{-1}(S')).$$

Da $CHASE_{E^{-1}}(S')$ jedoch nicht zwangsläufig alle Tupel ausreichend wiederherstellt, um die Anfrage Q zu beantworten, müssen dem Ergebnis noch die eingefrorenen Tupel I_{Prov} hinzugefügt werden. So ergibt sich für den allgemeinen Fall

$$Q'(S') = Q(E^{-1}(S') \cup I_{Prov}).$$

Die erste Phase der Berechnung von I_{Prov} identifiziert die beteiligten Tupel für alle ausgewählten Anfragen Q_i.

$$I'(S) = \bigcup_{i=1}^{n} CHASE_{Q_i^{-1}}(CHASE_{Q_i}(I))$$

Da bei Anfragen oder Evolutionsschritten Attribute, die für die Berechnung dieser relevant sind, herausprojiziert werden können, muss abgesichert werden, dass diese Attribute später erkannt werden können. Da Konstanten bei der Invertierung einer s-t-TGD weiter erhalten bleiben, ist die Selektion $A = c$ unproblematisch. Verbundattribute bzw. Attributvergleiche $A_i = A_j$, bei denen diese Attribute nicht im Kopf der TGD auftauchen, beeinflussen die Auswertung, werden aber nach dem BACKCHASE mit Nullwerten in den entsprechenden Tupeln besetzt. Um diese von Attributen abzuheben, die nicht für die Auswertung relevant sind, markieren wir Variablen, die im Kopf der invertierten s-t-TGD mehrfach vorkommen aber existenzquantifiziert sind, als besondere Variablen A_i^{Prov}. Diese generieren beim CHASE spezielle Nullwerte (*Provenance-Nullwerte*) η_j^{Prov}.

Provenance der Schema-Evolution. Anschließend führen wir die gleiche Berechnung für die Evolution E durch, um herauszufinden, welche Tupel sich nach der Evolution rekonstruieren lassen.

$$J = CHASE_{E^{-1}}(CHASE_E(I)) = CHASE_{E^{-1}}(I(S'))$$

Hierbei müssen wir Provenance-Nullwerten keine Beachtung schenken, da wir uns nur dafür interessieren, welche Informationen des alten Schemas im neuen erhalten bleiben. Da hier die Evolution berechnet werden soll, bedeutet dies allerdings auch die Berechnung neuer TIDs, falls Tupel nach der Evolution zusammenfallen oder aufgesplittet werden. Im allgemeinen Fall reicht der Standard-CHASE hier nicht aus. Betrachten wir zur Veranschaulichung folgende Beispiele. Das Attribut Z der Relation R auf der Datenbankinstanz $I(S)$ soll nach der Evolution E ebenfalls durch das Attribut Y abgedeckt werden. In diesem Fall werden Tupel aus R aufgesplittet, sodass potenziell neue TIDs nötig werden. Berechnet man nun $CHASE_E(I)$, ergibt sich die folgende Relation R auf der Datenbankinstanz $I(S')$.

$$E : R(tid, x, y, z) \rightarrow \exists Tid_2, Tid_3 : R(Tid_2, x, y) \wedge R(Tid_3, x, z)$$

Diese Nullwerte müssen nun durch entsprechende Konstanten für neue TIDs ersetzt werden. Im Falle des Aufsplittens

TID	X	Y	Z
1	1	1	11
2	1	2	12

(a) Relation R auf $I(S)$

TID	X	Y
η_1	1	1
η_2	1	11
η_3	1	2
η_4	1	12

(b) Relation R auf $I(S')$

Tabelle 1: Relation R vor und nach Anwendung des CHASE

TID	X	Y
1	1	1
2	1	2

(a) Relation R_2

TID	X
η_1	1

(b) Relation R_x

TID	Y
η_2	1
η_3	2

(c) Relation R_y

Tabelle 2: Relationen vor und nach Anwendung des CHASE

von Tupeln kann dies direkt über second-order-TGDs passieren, die statt Nullwerten über eine Funktion neue, eindeutige IDs berechnen, z. B. UUIDs. Fallen jedoch auch Tupel zusammen, z. B. weil ein Attribut durch die Evolution entfernt wird, funktioniert dies nicht. Hier könnten redundante Tupel mit unterschiedlichen IDs entstehen. Betrachten wir hierzu folgendes Beispiel der Relation R_2 und der Evolution E_2.

$$E_2 : R_2(tid, x, y) \rightarrow \exists Tid_2, Tid_3 : R_x(Tid_2, x) \wedge R_y(Tid_3, y)$$

Berechnet man nun $CHASE_E(I)$, ergeben sich die in Tabelle 2 dargestellten Relationen R_x und R_y in der nächsten Schemaversion. Hier können nicht direkt neue TIDs durch die s-t-TGD der Evolution vergeben werden, da sonst ein zusätzliches Tupel in R_x generiert wird, welches jedoch den gleichen X-Wert trägt. Solche Redundanzen sind nicht erwünscht, sodass in einem Zwischenschritt die Nullwerte durch entsprechende Konstanten ersetzt werden müssen. Des weiteren muss diese Vergabe neuer Konstanten invertierbar sein, damit bei der Invertierung der Evolution auch alte TIDs errechnet werden können. Daher bietet es sich an, ein Verzeichnis über die Provenance der TIDs zu führen, aus dem abgelesen werden kann, aus welchen Tupeln (TIDs) der vorherigen Schemaversion die Tupel der neuen Schemaversion gebildet wurden. Dies kann über eine Hilfstabelle $TID(tid_S, tid_{S'})$ realisiert werden, um welche die Abhängigkeiten für die Evolution erweitert werden müssen. Erweitern wir E_2 um die Nutzung dieser Hilfsrelation, ergibt sich folgende s-t-TGD

$$E_3 : R_2(tid, x, y) \rightarrow \exists Tid_2, Tid_3 : R_x(Tid_2, x) \wedge R_y(Tid_3, y) \wedge TID(tid, Tid_2) \wedge TID(tid, Tid_3).$$

Bevor die Invertierung des Evolutionsschrittes stattfindet, muss nun noch die Cleaning-EGD angewandt werden. Hierfür verwenden wir second-order-EGDs der Form

$$TID(tid, tid_2) \rightarrow tid_2 = f(tid_2).$$

Dabei ist f eine Funktion, die bei Eingabe eines Nullwertes eine neue TID (z. B. eine UUID), bei Eingabe einer Konstante jedoch diese selbst zurückgibt. Dies muss für alle TIDs durchgeführt werden. Besonders muss dabei darauf geachtet werden, dass die nummerierten Nullwerte global durch die Konstanten überschrieben werden.

Berechnung der zu speichernden Tupel. Im finalen Schritt berechnen wir nun I_{Prov}. Hierzu suchen wir die Menge derjenigen Tupel aus $I'(S)$, die sich auf Tupel in J abbilden

lassen. Diese Menge nennen wir J_M. Ein normaler Nullwert in einem Tupel in $I'(S)$ bedeutet, dass die Auswertung, aus deren BACKCHASE-Phase dieses Tupel stammt, jenes Attribut nicht benötigt. Diese Nullwerte können daher beliebig auf Konstanten oder Nullwerte in J abgebildet werden. Anders verhält sich dies jedoch bei den Provenance-Nullwerten. Da sie an der Auswertung beteiligt sind, dürfen diese nur auf Konstanten abgebildet werden. Sonst gehen z. B. Verbundkriterien und damit auch die Reproduzierbarkeit der Anfrage verloren. I_{Prov} ergibt sich dann aus denjenigen Tupeln, die an der Anfrage beteiligt waren, aber nicht in J_M sind. Diese Tupel können dann in extra Tabellen "eingefroren" werden, sodass sie für die entsprechenden Anfragen genutzt werden können. Weiter muss dazu bekannt sein, welche Anfragen auf welcher Schemaversion gestellt wurden, damit man weiß, auf welche Schemaversion zurückgerechnet werden muss. Zudem müssen auch alle Evolutionsschritte, die Schema S zu S' transformieren, aufbewahrt werden, damit das Zurückrechnen auf alte Versionen gelingt. Es ist auch möglich, die Evolutionsschritte zu komponieren und statt mehrerer Evolutionen nur die Kompositionen dieser zu speichern [11]. Dies verhindert jedoch das Berechnen der zwischenzeitigen Schemaversionen. Im allgemeinen Fall kann auf jeder Schemaversion S_i mindestens eine Anfrage gestellt werden, für welche wir die Reproduzierbarkeit voraussetzen wollen. Speichert man nur die s-t-TGD E_{komp}, welche die Evolution von S_{i-1} über S_i nach S_{i+1} beschreibt, so ist es nicht mehr möglich S_i selbst zu berechnen. Damit kann die Reproduzierbarkeit der Anfrage auf ihrer ursprünglichen Schemaversion nicht mehr gewährleistet werden. Kommen jedoch Fälle vor, bei denen zwischen zwei Schema-Evolution keine derartigen Auswertungen stattfanden, kann die Komposition der Evolutionsoperationen eingesetzt werden, um Zwischenschritte bei zukünftigen Berechnungen einzusparen.

Beispiel. Betrachten wir zur Veranschaulichung folgendes Anwendungsbeispiel. Gegeben sei die folgende Relation in Tabelle 3, sowie die Anfrage

$$Person(vo, na, 18, an) \rightarrow Q(na, an).$$

Aus Datenschutzgründen soll das Alter in der nächsten Schemaversion in Altersbereiche eingeteilt werden. Dies geschehe durch folgende Evolutionsoperation

$$Person(vo, na, al, an) \rightarrow PersonAnon(vo, na, f(al), an).$$

Der Altersbereich wird durch eine Funktion f bestimmt, welche das Alter in den entsprechenden Bereich $(0-5, 6-10, ...)$ einteilt. Wir vernachlässigen hier die TIDs, da keine der

Vorname	Name	Alter	Anschrift
Max	Mustermann	30	Parkstraße 3
Erika	Musterfrau	18	Schlossallee 7
Paul	Müller	18	Hafenstraße 12

Tabelle 3: Person ($I(S)$)

s-t-TGDs zu einem Zusammenfallen oder Aufsplitten von Tupel führt und diese somit nie verändert werden. Zunächst berechnen wir durch den CHASE das Anfrageergebnis Q, welches uns Namen und Anschriften der Personen liefert, die 18 Jahre alt sind (Tabelle 4). Anschließend führen wir den CHASE mit der invertierten Anfrage

$$Q(na, an) \rightarrow \exists Vo : Person(Vo, na, 18, an)$$

Name	Anschrift
Musterfrau	Schlossallee 7
Müller	Hafenstraße 12

Tabelle 4: Anfrageergebnis Q

auf dem Anfrageergebnis aus und erhalten somit die Menge derjenigen Tupel, die an der Anfrage beteiligt waren (abgebildet in Tabelle 5). Danach berechnen wir $I(S')$. Hierfür

Vorname	Nachname	Alter	Anschrift
η_1	Musterfrau	18	Schlossallee 7
η_2	Müller	18	Hafenstraße 12

Tabelle 5: Durch BACKCHASE generierte Personentabelle

führen wir zunächst den CHASE auf $I(S)$ mit der Evolution aus. Anschließend benutzen wir analog zur Anfrage den BACKCHASE auf $I(S')$ mit der inversen Evolution, um zu berechnen, welche Tupel und Informationen nach der Evolution rekonstruierbar sind. Da f nicht invertierbar ist, lautet die Inverse der Evolutionsoperation

$$PersonAnon(vo, na, ab, an) \rightarrow \exists Al : Person(vo, na, Al, an).$$

Der BACKCHASE liefert uns damit folgende Menge der rekonstruierbaren Tupel J, nachvollziehbar in Tabelle 6. Nun

Vorname	Name	Alter	Anschrift
Max	Mustermann	η_1	Parkstraße 3
Erika	Musterfrau	η_2	Schlossallee 7
Paul	Müller	η_3	Hafenstraße 12

Tabelle 6: Person (J)

suchen wir Abbildungen aus $I'(S)$ in J. Diese Menge J_M ist hier leer, da Konstanten (Alter) der Tupel aus $I'(S)$ nicht auf die Nullwerte in J abgebildet werden können. Daher bilden in diesem Fall alle an der Anfrage beteiligten Tupel I_{Prov} und müssen aufbewahrt werden. Hier sind das (Erika, Musterfrau, 18, Schlossallee 7) und (Paul, Müller, 18, Hafenstraße 12). Das bedeutet, dass nach der Berechnung der nachfolgenden Schemaversion dafür gesorgt werden muss, dass diese Tupel im alten Schema beibehalten werden. Alle anderen werden ausreichend für die Anfrage abgebildet.

4. EVALUATION

Der hier vorgestellte Ansatz verwendet den CHASE-Algorithmus zur Sicherstellung der Reproduzierbarkeit von Anfragen unter Schema-Evolution. Bekanntermaßen terminiert der CHASE-Algorithmus im allgemeinem Fall nicht zwingend, wird aber an zwei stellen der Berechnung eingesetzt. Im hier gezeigten Anwendungsfall ist die Terminierung allerdings garantiert. Grund dafür ist, dass in diesen CHASE-Phasen Ziel- und Quellschemata strikt getrennt sind, sodass auch neu generierte Werte nicht während der selben CHASE-Anwendung gelesen werden und somit niemals Zyklen entstehen können.

Die Berechnung der nachfolgenden Schemaversion schließt die Berechnung neuer TIDs ein. Da diese stets eindeutig sein müssen, und nachfolgend bei der Rückrechnung in alte Schemaversionen rekonstruiert werden müssen, schlagen wir vor

dies über eine s-o-Abhängigkeit zu lösen, die neue eindeutige Werte (z. B. in Form von UUIDs) liefert. Diese wiederum werden in einer Hilfstabelle abgelegt, um die Invertierung von Abhängigkeiten zu unterstützen. Dies stellt eine Erweiterung des Standard-CHASE da, welche es erlaubt zur Laufzeit neue Konstanten zu generieren. In unserem Fall ist dies vergleichbar mit einer EGD, deren Kopf neben einer Variable auch eine Konstante enthält. Diese wird durch eine gesonderte Methode im Programm berechnet und überprüft die Eingabe auf ihren Typ bezüglich des CHASE. Handelt es sich um eine Konstante, soll diese selbst zurückgegeben werden (TID bereits generiert). Bei einem Nullwert wird eine neue TID zurückgegeben, welche fortan in allen Tupeln den entsprechenden Nullwert ersetzt.

5. AUSBLICK

In diesem Beitrag haben wir einen Ansatz vorgestellt, um mit Hilfe des CHASE Daten zu identifizieren, die durch eine Schema-Evolution nicht ausreichend abgebildet werden, um bestimmte Anfragen zu reproduzieren. Die hier gezeigte Vorgehensweise ließe sich in den Evolutionsprozess einbinden, um nur diejenigen Daten aufzubewahren, welche für die Reproduzierbarkeit ausgewählter Anfragen relevant sind. Hierbei haben wir jedoch nur die Schema- und keine Datenevolution betrachtet, sodass der hier vorgestellte Ansatz Änderungen der Daten außerhalb von Schema-Evolution nicht berücksichtigt.

Eine offene Frage ist weiter, wie mit mehreren Schema-Evolution umgegangen werden kann. Tupel der Schemaversion S_{i+1}, die für eine ausgewählte Anfrage Q des Schemas S_i relevant sind, müssen auch beim Übergang in die Schemaversion S_{i+2} ausreichend abgebildet werden, sodass Q immer noch reproduzierbar ist. Der hier vorgestellte Algorithmus bietet dafür keine Überprüfung. Die Komposition der Evolutionsoperationen von Schema S_i zu S_{i+2} kann genutzt werden, um dies zu erreichen. Allerdings ist die Berechnung der Kompositionen über alle Schemaversionen und das Überprüfen für jede gewählte Anfrage vorheriger Schemaversionen ein großer Aufwand, welcher bei jeder Evolution betrieben werden muss. Wünschenswert ist eine Variante, welcher das aktuelle Schema genügt, um diese Überprüfung der nächsten Evolution durchzuführen, z. B. durch Annotation entsprechender Tupel und Attribute.

Eine weitergehende Fragestellung ist, wie wir den hier für einen Anwendungsfall vorgestellten BACKCHASE-Prozess für verschiedene Anwendungsfälle (in Abschnitt 2 Anwendungsfälle 1 bis 4) vereinheitlichen können, da wir alle Anwendungen mit einem einheitlichen CHASE&BACKCHASE-Tool (ChaTEAU, [3]) unterstützen wollen.

6. LITERATUR

[1] T. Auge and A. Heuer. Combining Provenance Management and Schema Evolution. In *IPAW*, volume 11017 of *Lecture Notes in Computer Science*, pages 222–225. Springer, 2018.

[2] T. Auge and A. Heuer. The Theory behind Minimizing Research Data: Result equivalent CHASE-inverse Mappings. In *LWDA*, volume 2191 of *CEUR Workshop Proceedings*, pages 1–12. CEUR-WS.org, 2018.

[3] T. Auge and A. Heuer. ProSA - using the CHASE for provenance management. In *ADBIS*, volume 11695 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2019.

[4] T. Auge, E. Manthey, S. Jürgensmann, S. Feistel, and A. Heuer. Schema evolution and reproducibility of long-term hydrographic data sets at the IOW. In *LWDA*, volume 2738 of *CEUR Workshop Proceedings*, pages 258–269. CEUR-WS.org, 2020.

[5] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro, and E. Tsamoura. Benchmarking the Chase. In *PODS*, pages 37–52. ACM, 2017.

[6] P. Buneman, S. Khanna, and W. C. Tan. Why and Where: A Characterization of Data Provenance. In *ICDT*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2001.

[7] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in Databases: Why, How, and Where. *Found. Trends Databases*, 1(4):379–474, 2009.

[8] C. Curino, H. J. Moon, and C. Zaniolo. Graceful database schema evolution: the PRISM workbench. *Proc. VLDB Endow.*, 1(1):761–772, 2008.

[9] A. Deutsch and R. Hull. Provenance-Directed Chase&Backchase. In *In Search of Elegance in the Theory and Practice of Computation*, volume 8000 of *Lecture Notes in Computer Science*, pages 227–236. Springer, 2013.

[10] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[11] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Schema Mapping Evolution Through Composition and Inversion. In *Schema Matching and Mapping*, Data-Centric Systems and Applications, pages 191–222. Springer, 2011.

[12] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Mapping and cleaning. In *ICDE*, pages 232–243. IEEE Computer Society, 2014.

[13] S. Greco, C. Molinaro, and F. Spezzano. *Incomplete Data and Data Dependencies in Relational Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[14] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40. ACM, 2007.

[15] A. Heuer, G. Saake, and K. Sattler. *Datenbanken - Implementierungstechniken, 4. Auflage.* MITP, 2019.

[16] M. Meier. The backchase revisited. *VLDB J.*, 23(3):495–516, 2014.

[17] M. L. Möller, S. Scherzinger, M. Klettke, and U. Störl. Why It Is Time for Yet Another Schema Evolution Benchmark - Visionary Paper. In *CAiSE Forum*, volume 386 of *Lecture Notes in Business Information Processing*, pages 113–125. Springer, 2020.

[18] F. Rose. Erweiterung des CHASE-Werkzeugs ChaTEAU um eine BACKCHASE-Phase. Masterarbeit, Universität Rostock, DBIS, 2020.

[19] F. Rose and A. Heuer. Der BACKCHASE zur Unterstützung von Data Provenance und Schema-Evolution. Technical Report CS 02-21, Institut für Informatik, Universität Rostock, 2021. Langfassung dieses Artikels.