Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

Provenance Management unter Verwendung von Schemaabbildungen mit Annotationen

Dissertation
von
Tanja Auge
zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

https://doi.org/10.18453/rosdok_id00004403

Gutachter:

1. Gutachter:
Prof. em. Dr. rer. nat. habil. Andraes Heuer
Universität Rostock, Deutschland

2. Gutachter:
Prof. Dr. Torsten Grust
Universität Tübingen, Deutschland

3. Gutachter:
Prof. Dr. rer. nat. Bertram Ludäscher
University of Illinois at Urbana-Champaign, School of Information Science, USA

Datum der Einreichung: November 2022

Datum der Verteidigung: Juli 2023

Zusammenfassung

Die Verfolgung, Organisation und Archivierung von Daten, welche im Rahmen wissenschaftlicher Projekte, Experimente oder Beobachtungen gesammelt werden, ist Aufgabe des Forschungsdatenmanagements. Je nach Anwendungsfall unterscheiden sich dabei die Maßnahmen und Verfahren, welche hierfür zum Einsatz kommen. In allen Fällen soll jedoch der Weg von der Datenerhebung bis zur Veröffentlichung replizierbar, rekonstruierbar und plausibel gehalten werden. Das kontinuierliche Wachstum der Daten, häufige Schemaänderungen sowie die Vielfältigkeit der Daten und ihrer Auswertungsmethoden machen die Speicherung aller möglichen Datenbankzustände dabei zu einer sehr aufwendigen und langwierigen Aufgabe.

Mit Hilfe von Provenance lässt sich jedoch feststellen, welcher Teil der originalen Daten — meinst primäre oder sekundäre Forschungsdaten — langfristig gespeichert werden muss, um die Reproduzierbarkeit, Replizierbarkeit oder Plausibilität einer konkreten Auswertung zu gewährleisten. Auch sollen im Falle temporaler Datenbanken Änderungen am Schema berücksichtigt werden können, um alte Datenbestände nicht komplett archivieren zu müssen, sondern aus bekannten Beständen zurück berechnet werden können. Neben der Auswertungsanfrage sowie dem Anfrageergebnis benötigen wir somit zusätzliche Annotationen, um ein Anfrageergebnis auf eines der oben genannten Kriterien zu überprüfen. Außerdem dürfen die gespeicherten Daten nicht mit bestehenden Datenschutzrichtlinien kollidieren.

Durch die Kombination vom Chase — einer Familie von Algorithmen zur Transformation von Datenbanken — mit Data Provenance und zusätzlichen Annotationen können für eine gegebene Auswertungsanfrage eine anonymisierte (minimale) Teil-Datenbank einer originalen (Forschungs-)Datenbank berechnet werden. Um die Reproduzierbarkeit, Replizierbarkeit oder Plausibilität eines Anfrageergebnisses zu gewährleisten, müssen die auf der originalen Datenbank durchgeführten Auswertungen auch auf der rekonstuierten (minimalen) Teil-Datenbank durchführbar sein. Hierfür nutzen wir eine Version des Chase&Backchase, einer Erweiterung des Chase. Ziel des Promotionsprojekts *ProSA* (**Pro**venance Management using **S**chema Mappings with **A**nnotations) ist somit die Anwendung und Verallgemeinerung von Techniken des Provenance Managements im Bereich des Forschungsdatenmanagements unter Verwendung des mit zusätzlichen Provenance-Informationen erweiterten Chase&Backchase.

Abstract

Tracking, organizing, and archiving data collected in the course of scientific projects, experiments, or observations is the task of research data management. The measures and procedures used for this vary depending on the application. In all cases, however, the path from data collection to publication should be kept replicable, reconstructable and plausible. The continuous growth of the data, frequent schema changes, and the diversity of the data itself and its evaluation methods make the storage of all possible database states a very time-consuming and tedious task.

However, using provenance, it is possible to determine which part of the original data — primary or secondary research data, at least — need be stored in the long term in order to ensure the repeatability, replicability or plausibility of a concrete evaluation. Also, in the case of temporal databases, changes to the schema should be able to be taken into account in order not to have to archive old data sets completely. Thus, in addition to the evaluation query and its result, we need additional annotations to check a query result against one of the above criteria. Furthermore, the stored data must not collide with existing data protection policies.

By combining Chase — a family of algorithms for transforming databases — with data provenance as well as additional annotations, an anonymized (minimal) partial database of an original (research) database can be computed for a given evaluation query. To ensure the repeatability, replicability, or plausibility of a query result, the evaluations performed on the original database must also be feasible on the reconstructed minimal partial database. For this purpose, we use a version of Chase&Backchase, an extension of Chase. Thus, the goal of the PhD project *ProSA* (**Pro**venance Management using **S**chema Mappings with **A**nnotations) is to apply and generalize provenance management techniques in the field of research data management using Chase&Backchase enhanced with additional provenance.

Inhaltsverzeichnis

I. Einleitung

1. Motivation **9**
- 1.1. Allgemeine Problemstellung . . . 14
- 1.2. Notationen und weitere Vorbemerkungen . . . 16
- 1.3. Aufbau der Arbeit . . . 16

2. Problembeschreibung und Ziele **19**
- 2.1. Problembeschreibung . . . 19
- 2.2. Ziele der Dissertation . . . 21
- 2.3. Anwendungsgebiet Forschungsdatenmanagement . . . 21
- 2.4. Problemstellung im Kontext des Forschungsdatenmanagements . . . 25

II. Grundlagen und State of the Art

3. Techniken, Voraussetzungen und Related Work **29**
- 3.1. Logik erster und zweiter Stufe . . . 29
- 3.2. Relationale Datenbanken . . . 31
- 3.3. Chase-Algorithmus . . . 35
- 3.4. Provenance . . . 49
- 3.5. Schemaevolution . . . 55
- 3.6. Privacy . . . 55

4. Analyse verschiedener CHASE- und Provenance-Systeme **63**
- 4.1. Analyse verschiedener Chase-Systeme . . . 63
- 4.2. Analyse verschiedener Provenance-Systeme . . . 71

III. Ergebnisse der Dissertation

5. Formalisierung der Problemstellung **83**
- 5.1. Modifizierte technische Problemstellung . . . 83
- 5.2. Zusammenfassung der Ergebnisse . . . 86
- 5.3. Abgrenzung zu anderen Ansätzen . . . 90

6. Berechnung der (minimalen) Teil-Datenbank **93**
- 6.1. Ergebnisäquivalente und tp-relaxte Chase-Inverse . . . 93
- 6.2. Invertierung von (s-t) tgds . . . 97
- 6.3. Auswertungsanfragen und ihre Chase-Inversen . . . 100
- 6.4. Chase&Backchase-Algorithmus zur Bestimmung der (minimalen) Teil-Datenbank . . . 107
- 6.5. Zwischenfazit . . . 109

7. Provenance und Evolution **111**
- 7.1. Berechnung der (minimalen) Teil-Datenbank bei sich ändernden Datenbanken . . . 111
- 7.2. Klassifikation relevanter Evolutionsoperartoren . . . 114
- 7.3. Evolutionsperatoren und ihre Chase-Inversen . . . 117
- 7.4. ***what***-Provenance . . . 127
- 7.5. Zwischenfazit . . . 128

8. Provenance und Privacy **131**
- 8.1. Interview-Studie . . . 131
- 8.2. Anonymisierung der ***where***, ***why***- und ***how***-Provenance . . . 135
- 8.3. Relevante Anonymisierungsmethoden . . . 138
- 8.4. Zwischenfazit . . . 141

9. ChaTEAU **143**
9.1. Evolutionsperatoren und ihre Chase-Inversen . . . 143
9.2. Ablauf und Demonstration von ChaTEAU . . . 146
9.3. Anwendungen von ChaTEAU . . . 152
9.4. Zwischenfazit . . . 153

10. ProSA **155**
10.1. Konzeptueller Ablauf von ProSA . . . 156
10.2. Transformation einer SQL-Anfrage in eine Menge von (s-t) tgds . . . 160
10.3. Invertierung der Auswertungsanfrage . . . 163
10.4. Einbindung von Aggregatfunktionen . . . 165
10.5. Einbindung von Provenance (als Erweiterung von ChaTEAU) . . . 171
10.6. Bestimmung des Chase-Inversentyps . . . 172
10.7. Einbindung von Privacy (als Erweiterung von ChaTEAU) . . . 173
10.8. Chase&Backchase in ProSA . . . 180
10.9. Implementierung des ProSA-Systems . . . 181

IV. Schlussbemerkungen

11. Zusammenfassung und Ausblick **189**
11.1. Zusammenfassung . . . 189
11.2. Diskussion der Ergebnisse und weiterführende Fragestellungen . . . 190

12. Danksagungen **193**

V. Listen und Verzeichnisse

Abkürzungsverzeichnis **i**

Symbolverzeichnis **iii**

Tabellenverzeichnis **vii**

Abbildungsverzeichnis **ix**

Anfrageverzeichnis **xi**

Projektverzeichnis **xiii**

Literatur **xv**

VI. Anhänge

A. Praxis-Beispiel **xxv**

B. Benchmark-Anfragen **xxvii**
B.1. Provenance-Anfragen . . . xxvii
B.2. Chase-Anfragen . . . xxix

C. Klassifikation der Chase-Inversen (Beweise) **xxxi**

D. Ein- und Ausgabe-Dateien für ChaTEAU und ProSA **xlv**
D.1. ChaTEAU-Dateien . . . xlvi
D.2. ProSA-Dateien . . . li

E. Fragenkatalog der Interview-Studie **lv**

F. Veröffentlichungen und betreute Arbeiten **lvii**
F.1. Vorarbeiten und Veröffentlichungen . . . lvii
F.2. Betreute studentische Abschlussarbeiten und Projekte (themenbezogener Auszug) . . . lviii

Teil I.

Einleitung

1. Motivation

Nachhaltigkeit — ein Begriff, der aus unserer heutigen Gesellschaft kaum noch wegzudenken ist. Viele Menschen assoziieren mit diesem Schlagwort zunächst die Vermeidung von Plastik und CO_2, Greta Thunberg, Fridays for Future oder die Anti-Atomkraft-Bewegung. Das Thema Nachhaltigkeit betrifft somit die Industrie, die Politik und jeden von uns persönlich. Aber auch in der Forschung spielt Nachhaltigkeit eine immer größer werdende Rolle. So zum Beispiel die nachhaltige Nutzung von Forschungsdaten, welche Ressourcen schont und den dauerhaften Zugriff auf einmalige historische Daten ermöglicht. „Voraussetzung zur Nutzung dieser Sekundärdaten ist eine ausführliche Dokumentation der Entstehungsbedingungen, Stichproben und Messverfahren“ [Tau19]. Eine nachhaltige Datennutzung ist nach Aussage der Autoren nur dann möglich, wenn existierende Forschungsdaten möglichst allumfassend genutzt werden können. Sie appellieren daher „an alle Forscher, ihre eigenen Daten anderen Nutzern zur Verfügung zu stellen“.

Eine Bereitstellung der eigenen Forschungsdaten ist jedoch nicht immer möglich oder sinnvoll, wie unsere in [ASH21b] vorgestellte Interview-Studie zeigt. Hierfür haben wir Rostocker Datennutzer aus Forschung und Wirtschaft zu ihren Beweggründen für oder gegen die Veröffentlichung ihrer Daten befragt. Wie erwartet variieren die Antworten so stark wie die Ziele der Datennutzung selbst. Gründe, die gegen eine Veröffentlichung von (Forschungs-)Daten sprechen, sind beispielsweise privacy-spezifisch (etwa im Sinne der DSGVO[1]), wirtschaftlicher, finanzieller, militärischer oder politischer Natur. Auch Interessen Dritter sowie die Hoffnung auf einen Wettbewerbsvorteil können gegen eine Datenbereitstellung sprechen. Ein nachhaltiges Forschungsdatenmanagement ist so jedoch kaum möglich. Um diesem Problem entgegenzuwirken, wurden 2016 die sogenannten FAIR-Prinzipien (FAIR)[2] entwickelt. Diese bilden eine Leitlinie zur Verbesserung der Auffindbarkeit (***F****indability*), Zugänglichkeit (***A****ccessibility*), Kompatibilität (***I****nteroperability*) und Wiederverwendbarkeit (***R****euseability*) digitaler Ressourcen. Im Vordergrund steht hierbei aufgrund des schnell zunehmenden Volumens, der Komplexität und der Erstellungsgeschwindigkeit der Daten die Garantie der maschinellen Auffindbarkeit dieser.

Ein Ziel der FAIR-Prinzipien ist die Wiederverwendung von Daten. Um dies zu erreichen, müssen die Daten und ihre Metadaten so detailliert beschrieben werden, dass sie in verschiedenen Umgebungen repliziert und/oder kombiniert werden können. Viele Konferenzen, Journale und Drittmittelgeber fordern daher immer häufiger neben Open Access-Artikeln auch die Veröffentlichung der zugrundeliegenden Forschungsdaten. Wir sprechen in diesem Zusammenhang von *Open Data*. Die Veröffentlichung der Originaldaten — nicht Rohdaten, sondern die für die Berechnung aufbereiteten Daten — erlaubt so die Wiederverwendung bereits existierender Daten über das ursprüngliche Forschungsprojekt hinaus.

Auch die Qualität der Forschung selbst, insbesondere das *Forschungsdatenmanagement*, kann durch Open Data verbessert werden. So können durch Herausgabe der Originaldaten bereits im Review-Prozess die Ergebnisse einer geplanten Veröffentlichung auf Reproduzierbarkeit, Replizierbarkeit oder Plausibilität überprüft werden. Denn ein (experimentelles) Ergebnis ist erst dann vollständig gesichert, wenn es unabhängig vom Nutzer reproduziert bzw. repliziert werden kann. Da eine hundertprozentige Reproduzierbarkeit bzw. Replizierbarkeit jedoch oft nicht gewährleistet werden kann, sollen zumindest die digitalen Anteile der Ergebnisse, sogenannte *Artefakte*, wie Softwaresysteme, Skripte zur Durchführung von Experimenten, Eingabedatensätze, im Experiment gesammelte Rohdaten oder Skripte zur Analyse der Ergebnisse reproduzierbar bzw. replizierbar sein. So beschreibt es das ACM Artifact Review and Badging auf seiner Website[3].

Doch was verstehen wir überhaupt unter Reproduzierbarkeit, Replizierbarkeit oder Plausibilität? Die *Association for Computing Machinery*[4] (ACM) hat als eine der größten Vereine unserer Community in einigen Konferenzen

[1] DSGVO: https://dsgvo-gesetz.de
[2] FAIR-Prinzipien: https://www.go-fair.org/fair-principles/
[3] ACM Artifact Review and Badging – Version 1.0: https://www.acm.org/publications/policies/artifact-review-badging
[4] ACM: https://www.acm.org

und Zeitschriften bereits formale Verfahren zur Prüfung der Artefakte eingeführt. Wir wollen uns an die dortigen Terminologien halten (Quelle: ACM, 08.11.2022):

- *Wiederholbarkeit bzw. Repeatability* (gleiches Team, gleicher Versuchsaufbau):
  Die Messung kann mit der angegebenen Genauigkeit vom selben Team unter Verwendung desselben Messverfahrens, desselben Messsystems, unter denselben Betriebsbedingungen und am selben Ort bei mehreren Versuchen durchgeführt werden. Für computergestützte Auswertungen bedeutet dies, dass ein Forscher seine eigenen Berechnungen zuverlässig wiederholen kann.

- *Reproduzierbarkeit bzw. Reproducibility* (anderes Team, gleicher Versuchsaufbau):
  Die Messung kann mit der angegebenen Genauigkeit von einem anderen Team unter Verwendung desselben Messverfahrens, desselben Messsystems, unter denselben Betriebsbedingungen, am selben oder an einem anderen Ort bei mehreren Versuchen durchgeführt werden. Für computergestützte Auswertungen heißt das, dass eine unabhängige Gruppe mit den eignen Artefakten des Autors das gleiche Ergebnis erzielen kann.

- *Replizierbarkeit bzw. Replicability* (anderes Team, anderer Versuchsaufbau):
  Die Messung kann mit der angegebenen Präzision von einem anderen Team, einem anderen Messsystem, an einem anderen Ort und in mehreren Versuchen durchgeführt werden. Für computergestützte Auswertungen bedeutet dies, dass eine unabhängige Gruppe das gleiche Ergebnis mit Artefakten erzielen kann, die sie völlig unabhängig voneinander entwickelt.

Falls dies nicht möglich sein sollte, soll das Anfrageergebnis zumindest auf *Plausibilität* überprüfbar sein. In unserem Fall bedeutet dies, dass das originale Anfrageergebnis (plausibel) auf das Ergebnis der Anfrageauswertung auf einem veröffentlichten Teil der Daten abgebildet werden kann. Das Attribut soll also demselben Datentyp entsprechen und der Attributwert in einem vorgegebenen Wertebereich bzw. einer vorgegebenen Wertemenge liegen. So kann die Durchschnittsnote 1.6 beispielsweise auf die Note „gut“ abgebildet werden, welche durch Auswertung der Teil-Datenbank entsteht, wenn „gut“ als Intervall $[1.5, 2.4]$ interpretiert wird.

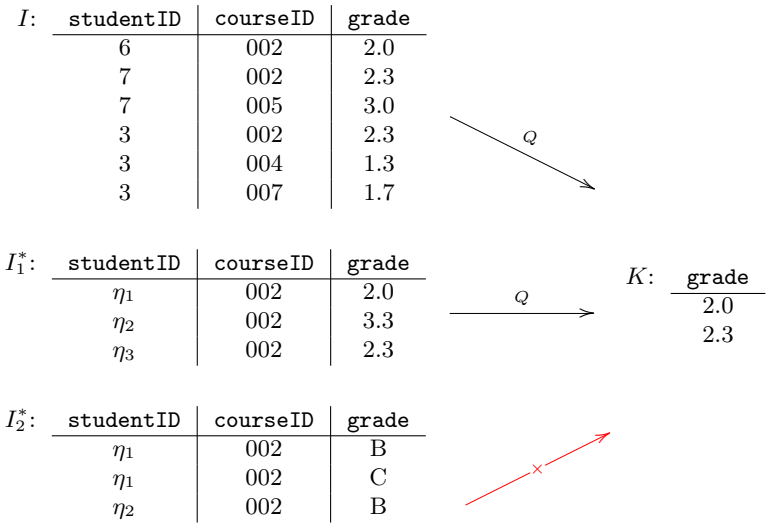


Abbildung 1.1. Reproduzierbarkeit, Replizierbarkeit und Plausibilität am Beispiel: Die Veröffentlichung der Teil-Instanz I_1^* ermöglicht wegen $Q(I_1^*) = Q(I)$ die Reproduktion sowie Replizierbarkeit des Anfrageergebnisses K. Mit Hilfe von I_2^* kann K lediglich auf Plausibilität überprüft werden.

Schauen wir uns dies an einem konkreten Beispiel an: Gegeben sei eine Universitätsdatenbank bestehend unter anderem aus den beiden Relationen `Student`(`studentID`, `lastName`, `firstName`, `studies`) und `Grade`(`courseID`, `studentID`, `semester`, `grade`). Das vollständige Beispiel kann in Anhang A nachgelesen werden. Sei für unser Beispiel hier Q die Anfrage, welche die Noten eines bestimmten Kurses — hier 002 — ausgibt und I eine gegebene Quell-Instanz (siehe Abbildung 1.1), welche durch Verbund und Projektion der beiden Relationen `Student` und `Grade` entstanden ist. Dann liefert die Auswertung von Q zwei Tupel mit den Noten 2.0 und 2.3. Seien weiter I_1^* und I_2^* zwei Teil-Instanzen von I nach Definition 6.2, welche wir der Community bereitstellen. Unabhängig davon, wer die Anfrage Q auswertet, gilt $Q(I_1^*) = Q(I)$. Das Anfrageergebnis $K = Q(I)$ ist somit aus I_1^* reproduzierbar bzw. replizierbar. Wählen wir alternativ I_2^*, so ist K aus I_2^* wegen $Q(I_2^*) \neq Q(I)$ nicht reproduzierbar

bzw. replizierbar. Interpretieren wir jedoch die Note B als Generalisierung der Noten 2.0 und die Note C als Generalisierung der Note 3.3, so ist K zumindest plausibel.

In der vorliegenden Arbeit *Provenance Management unter Verwendung von Schemaabbildungen mit Annotationen* stellen wir einen neuen Ansatz vor, welcher eben diese Überprüfbarkeit eines (zu veröffentlichenden) Forschungsergebnisses ermöglicht. Hierfür werten wir eine gegebene Anfrage, genannt *Auswertungsanfrage*, aus und invertieren diese anschließend wieder. Durch die Invertierung der Anfrage — meist die Identitätsabbildung erweitert auf das Quell-Schema — können wir die für das Anfrageergebnis relevanten Quell-Tupel bestimmen. Die Veröffentlichung dieser Teil-Datenbank ermöglicht uns dann die Reproduzierbarkeit, Replizierbarkeit oder Plausibilität eines Forschungsergebnisses zu garantieren ohne die gesamte Forschungsdatenbank zu kennen. Im Falle der obigen Anfrage Q genügt es somit die beiden Tupel $(\eta_1, 002, 2.0)$ und $(\eta_3, 002, 2.3)$ mit `courseID` $= 002$ zu kennen (siehe Instanz I^* ohne das rote Tupel $(\eta_2, 002, 2.3)$ in Abbildung 1.2). Die anderen beiden Quell-Tupel $(7, 005, 3.0)$ und $(3, 004, 1.3)$ sind für die Auswertung von Q irrelevant. Auch die Kenntnis der `studentID` ist zur Beurteilung der Nachvollziehbarkeit des Anfrageergebnisses nicht notwendig.

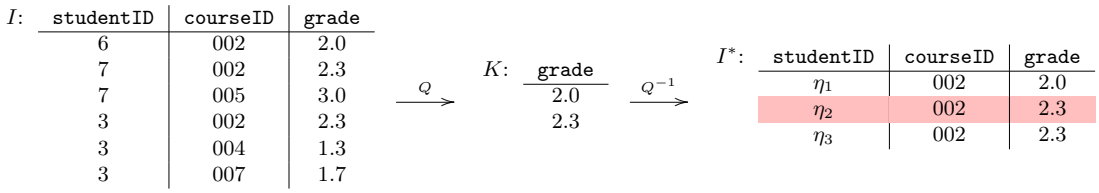


Abbildung 1.2. Berechnung einer (minimalen) Teil-Datenbank am konkreten Beispiel mit kalkuliertem Informationsverlust in `studentID` sowie der Tupel $(7, 005, 3.0)$ und $(3, 007, 1.7)$; das rot hinterlegte Tupel $(\eta_2, 002, 2.3)$ kann durch zusätzliche Provenance-Informationen bestimmt werden.

Da das Auswerten einer Anfrage immer einen natürlichen Informationsverlust mit sich bringt, erweitern wir die Anfrage um zusätzliche *Provenance-Informationen*. Diese beschreiben die Herkunft bzw. den Entstehungsprozess eines Anfrageergebnisses, was den Informationsverlust in einem gewissen Maße ausgleicht. Die zusätzlichen Informationen ermöglichen so die Angabe einer verbesserten Teil-Datenbank mit weniger Nullwerten sowie der Replizierbarkeit verlorengegangener Duplikate, was den oben erwähnten Review-Prozess zusätzlich unterstützt. Im Falle der Anfrage Q von oben entspräche dies dem rot hinterlegten Tupel $(\eta_2, 002, 2.3)$, welches nun zusätzlich in I^* enthalten ist (siehe Abbildung 1.2).

Unter Provenance-Informationen verstehen wir dabei sowohl die klassischen *Zeugenbasen* (siehe Definition 3.37) und *Provenance-Polynome* (siehe Definition 3.38) der ***why***- sowie der ***how***-Provenance als auch zusätzliche Annotationen, welche wir in separaten *Side Tables* (siehe Definition 6.10) abspeichern. Hierzu gehören insbesondere Attributwerte, welche durch Verdichtung wie etwa bei der Aggregation verloren gehen würden oder falsch rekonstruiert werden wie etwa bei der Vereinigung. Wann und in welchem Maße zusätzliche Provenance-Informationen notwendig sind, werden wir insbesondere in den Abschnitten 6.3 und 7.3 diskutieren.

Neben dem Nachhaltigkeitsaspekt sowie der Verbesserung von Review-Prozessen können Provenance und Open Data zudem dabei helfen Falschmeldungen aufzudecken. So kommt es immer wieder zur Veröffentlichung vermeintlicher Forschungsergebnisse, deren Datengrundlage jedoch nicht überprüft werden kann. Ein solcher Skandal ereignete sich beispielsweise zu Beginn der Covid-19 Pandemie (Quelle: Spiegel Wissenschaft, 14.06.2020). Nach Veröffentlichung einer wissenschaftlichen Studie über die Gefährdung eines Covid-Medikaments wurden weltweit zugehörige Medikamentenstudien abgebrochen. Das Problem an dieser Stelle war jedoch, dass die Autoren der Studie keinen Zugang zu den zugehörigen *Originaldaten* hatten. Grund hierfür war nach Aussage der Firma, welche die Daten erhoben hatte, der Schutz ihrer Patientendaten. Schließlich wurde die Vermutung laut, dass es diese Daten nie gegeben hätte. Wir erkennen an dieser Stelle einen Konflikt zwischen dem Schutz von Daten (*Privacy*) sowie dem Wunsch bzw. der Notwendigkeit der Nachvollziehbarkeit eines Forschungsergebnisses (*Provenance*).

Recheriert man den Begriff *Privacy*, bezieht sich dieser in der Regel auf den Schutz personenbezogener Daten. So definiert beispielsweise Wikipedia den Begriff Privacy, zu Deutsch *Privatsphäre*, als „den nicht-öffentlichen Bereich, in dem ein Mensch unbehelligt von äußeren Einflüssen sein Recht auf freie Entfaltung der Persönlichkeit wahrnimmt. Das Recht auf Privatsphäre gilt als Menschenrecht und ist in allen modernen Demokratien verankert“ (Quelle: Wikipedia, 12.01.2022). Die Europäische Datenschutz-Grundverordnung hingegen definiert Privacy, zu

Deutsch *Datenschutz*, als „Informationen, die sich auf eine identifizierte oder identifizierbare natürliche Person beziehen“, wobei eine Person „direkt oder indirekt, insbesondere mittels Zuordnung zu einer Kennung [...] oder zu einem oder mehreren besonderen Merkmalen [...] identifiziert werden kann“ (Quelle: DSGVO, 12.01.2022). Beide Definitionen beziehen sich auf den Schutz personenbezogener Daten. Die Einhaltung von Privacy-Aspekten besteht in diesem Fall in der Anonymisierung personenbezogener Daten. In der Wissenschaft betrifft dies insbesondere Patienten- oder Nutzerdaten, welche in medizinischen Studien, auf Social Media oder bei der Nutzung von Geräten und Software gesammelt werden. Der Umgang mit sensiblen Daten wird dabei oft in sogenannten *Privacy Policies* oder *Data Policies* festgehalten.

Um die *Nachvollziehbarkeit*, d.h. die Reproduzierbarkeit, Replizierbarkeit oder Plausibilität, eines veröffentlichten Forschungsergebnisses gewährleisten zu können, müssen die Wissenschaftler daher verpflichtet werden, neben den Ergebnissen ihrer Auswertungen auch die zugehörigen Daten und Metadaten — wie Messmethoden, Gerätebezeichnungen o. ä. — zu veröffentlichen, ohne dabei jedoch die geltenden Privacy und Data Policies zu verletzen. Unser Ansatz zur Lösung dieses Problems ist die Bestimmung des minimalen Teils der originalen Daten, genannt *(minimale) Teil-Datenbank*, welche die Nachvollziehbarkeit des Forschungsergebnisses garantieren. Alle übrigen Daten können so bis zur allgemeinen Veröffentlichung weiterhin privat gehalten werden, ohne dass ungewollte, weitergehende Auswertungen durch Dritte möglich sind.

Dass auch nicht-personenbezogene (Forschungs-)Daten schützenswert sein können, zeigt das Beispiel von *MOSAiC*[5]. Das internationale Forschungsprojekt ist die „größte Arktisexpedition aller Zeiten“ (Quelle: Alfred-Wegener-Institut, 12.01.2022). Mit riesigem Aufwand wurden in verschiedenen Studien Daten über die zentrale Arktis gesammelt. Dafür driftete der deutsche Forschungseisbrecher Polarstern ein Jahr lang durch das Nordpolarmeer. Da das Projekt aus internationalen, teilweise öffentlichen Geldern finanziert wird, sollen die Daten im Anschluss international und frei zugänglich bereitgestellt werden. Für ihren Aufwand bei der Datenerhebung erhalten die Expeditionsteilnehmer und -Partner nach der MOSAiC Policy[6] bis 2023 jedoch ein Vorrecht auf die Verarbeitung der gesammelten Daten. Innerhalb der ersten Jahre sind die Daten somit schützenswert. Nach Ablauf der Policy-Frist können die MOSAiC-Daten dann bedenkenlos veröffentlicht werden.

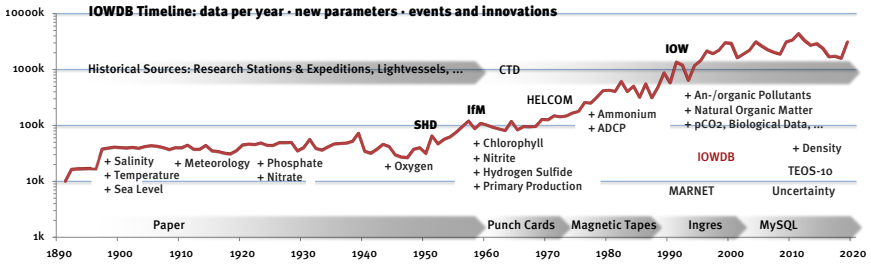


Abbildung 1.3. Datenzuwachs (pro Jahr) in IOWDB, einer Forschungsdatenbank des IOW [BFJ14].

Eine Datensperre von drei bis fünf Jahren, in der Regel aber mind. bis zum Projektende ist dabei nichts ungewöhnliches, für länger laufende Projekte jedoch nicht umsetzbar. Hier besteht oft eine andere Problematik. Durch die lange Projektlaufzeit von mehreren Jahren, manchmal auch Jahrzehnten, unterliegen die Datenbanken zwangsläufig Änderungen im Schema sowie im gespeicherten Datenbestand. Schauen wir uns hierzu die „Sauerstoff“-Datenbanken [Man20] des *Leibniz-Instituts für Ostseeforschung Warnemünde (IOW)*[7] an. Diese durchläuft in regelmäßigen Abständen einen aufwendigen Evolutionsprozess, sodass eine Rückverfolgbarkeit der frühen Auswertungsergebnisse auf der originalen Datenbankinstanz gegebenenfalls gar nicht mehr möglich ist. In diesem Fall muss die originale Datenbankinstanz durch Invertierung der Evolution zunächst rekonstruiert werden, bevor die zugehörige (minimale) Teil-Datenbank bestimmt werden kann (siehe Kapitel 7). Andersherum kann eine einmal bestimmte Teil-Datenbank durch Evolution stets auf dem neuesten Stand gehalten werden, sodass auch neue Anfragen auf der originalen Teil-Datenbank ermöglicht werden.

Im Kontext des Forschungsdatenmanagements nehmen wir zudem an, dass die Datenbestände unserer Forschungsdatenbanken niemals weniger werden. So verzeichnen wir in der IOWDB, einer der Forschungsdatenbanken des

[5] Projektwebsite MOSAiC: `https://mosaic-expedition.org`
[6] MOSAiC Data Policy: `https://mosaic-expedition.org/wp-content/uploads/2020/12/mosaic_datapolicy.pdf`
[7] Institutswebsite IOW: `https://www.io-warnemuende.de`

IOW, beispielsweise einen Datenzuwachs von mehr als 1000k Datensätzen pro Jahr (siehe Abbildung 1.3). Datenlöschungen liegen in diesem konkreten Fall hingegen nicht oder nur in Ausnahmefällen vor.

Unser Ansatz in ProSA, welchen wir insbesondere in den Kapiteln 5 und 10 vorstellen werden, ermöglicht es uns, die veröffentlichten Ergebnisse auch bei sich ändernden Datenbankschemata reproduzierbar, replizierbar und plausibel zu halten. Zudem sollen sie vergleichbar und robust gegen Interpretations- oder Auswertungsfehler sein. Dabei speichern wir statt jeder Datenbankversion selbst nur eine (minimale) Teil-Datenbank vor bzw. nach der Evolution. Bei sehr großen Datenbeständen (Petabyte und mehr), die sich häufig ändern, kann die durch die Teil-Datenbank garantierte Datenreduktion somit insbesondere Kosten im Vergleich zum Speichern des ganzen Datenbank-Snapshot sparen.

Insgesamt ergeben sich aus den geschilderten Fallbeispielen verschiedene, zunächst unabhängige Fragestellungen, die wir mit unserem Ansatz gemeinsam untersuchen wollen: Kann man Privacy-wahrende Auswertungen von Daten mit den Fragestellungen der Data Provenance (siehe Abschnitt 3.4.1) kombinieren? Wie kann man automatisch eine abstrahierte, anonymisierte Sicht auf die Originaldaten für Gutachter generieren, die es trotzdem ermöglicht, die Reproduzierbarkeit bzw. Replizierbarkeit (siehe oben) des Ergebnisses zu erhalten, oder zumindest die Plausibilität des Ergebnisses feststellen zu können? Können die Forschungsergebnisse auch Jahre später noch fehlerfrei rekonstruiert werden?

Doch was besagen Provenance und Privacy überhaupt? *Provenance* beschreibt die Herkunft eines Objektes bzw. dessen Entstehungsprozess. Je nach Quelle werden hierbei drei oder vier verschiedene Typen von Provenance unterschieden: *Data Provenance*, *Workflow Provenance*, *Information System Provenance* und *Provenance Metadata* [HDL17]. Wir konzentrieren uns im Folgenden insbesondere auf die Data Provenance, welche die Rückverfolgung von (aggregierten) Datensätzen bis zu den originalen Datensätzen beschreibt. Wie im MOSAiC-Beispiel angedeutet, können Forschungsdaten unabhängig von ihrem Inhalt schützenswert sein. Wir verstehen Privacy daher als den Schutz von (Forschungs-)Daten im Allgemeinen.

Fassen wir abschließend noch einmal zusammen: Was alle Beispiele gemeinsam haben, ist die Frage nach den Rohdaten bzw. den für die vorliegende Analyse aufbereiteten Daten. Wie im Beispiel des Review-Prozesses sollen die Quelldaten dabei möglichst allumfassend vorliegen. Hierfür erweitern wir unsere Auswertungsanfrage durch zusätzliche Provenance-Informationen. Bei personenbezogenen Daten oder sensiblen Forschungsdaten sind jedoch gewisse Privacy-Aspekte nicht zu vernachlässigen, was die Teil-Datenbank wiederum einschränkt. Wir sprechen von einer *anoynmisierten (minimalen) Teil-Datenbank*. Bei den Sauerstoff-Daten des IOW hingegen liegt die Schwierigkeit in den sich ändernden Schemta der zugrundeliegenden Datenbank, welche sich bei einem Zeitraum von 100 Jahren nicht vermeiden lassen. Hieraus ergeben sich für uns verschiedene Fragestellungen, welche wir im Verlauf der Arbeit diskutieren werden:

- Wie kommen die veröffentlichten Ergebnisse zustande? Wie kann ein veröffentlichtes Ergebnis verifiziert werden, d.h. reproduziert, repliziert oder zumindest auf Plausibilität getestet werden?
- Welche weiteren Informationen wie Data Provenance und zusätzliche Annotationen sind notwendig, um die Teil-Datenbank zu maximieren, d.h. möglichst vollständig rekonstruieren zu können?
- Welche zusätzlichen Privacy-Aspekte sind zu beachten? Welche Anonymisierungen sind möglich und/oder notwendig bzw. sinnvoll?
- Wie kann ein Ergebnis auch nach Jahren nachvollziehbar, d.h. replizierbar, reproduzierbar oder plausibel bleiben? Was ist zu beachten, sollte sich das Datenbankschema in der Zwischenzeit verändert haben?

Ziel der vorliegenden Arbeit ist die Bereitstellung aller für die Replizierbarkeit bzw. Reproduzierbarkeit eines Ergebnisses relevanten Originaldaten. Hierfür bestimmen wir einen mittels Data Provenance und zusätzlichen Annotationen erweiterten (minimalen) Teil der Originaldaten, welcher zusätzlich zum Forschungsergebnis veröffentlicht wird. Diese (minimale) Teil-Datenbank unterliegt dabei potenziellen Privacy-Aspekten und soll auch bei sich ändernden Datenbanken eingesetzt werden können.

1.1. Allgemeine Problemstellung

Zur Veranschaulichung der in dieser Arbeit behandelten Problematik, stellen wir uns folgende Situation vor: Eine junge Wissenschaftlerin ist besorgt über die vielen Zeitungsartikel zum Thema Sauerstoffmangel in der Ostsee. Reißerische Titel wie *Todeszonen wachsen rasant: Stirbt die Ostsee?* (Quelle: Frankfurter Allgemeine, Joachim Müller-Jung, 01.04.2014), *Sauerstoffmangel: Todeszonen im Meer* (Quelle: Spektrum.de, 25.10.2018), *Todeszonen in den Ozeanen – wenn dem Meer die Luft ausgeht* (Quelle: weather.com, 08.10.2018) deuten ein schwerwiegendes Problem an, welches unsere Wissenschaftlerin gerne selber überprüfen möchte. Um die Entwicklung des Sauerstoffgehalts der letzten 100 Jahre beurteilen zu können, plant sie daher eine Überblicksstudie, welche die Ergebnisse aller bisher veröffentlichten Studien zusammenfassen und vergleichen soll. Um die Original-Ergebnisse verifizieren zu können, möchte sie die Studienergebnisse selbst replizieren bzw. reproduzieren, wofür sie jedoch die originalen Forschungsdaten benötigt.

Wir als Forschungsinstitut unterstützen das Vorhaben der jungen Wissenschaftlerin und sind bereit, ihr unsere Forschungsdaten zur Verfügung zu stellen. Selbstverständlich haben wir an unserem Institut ein eigenes Forschungsdatenmanagement, welches wir an dieser Stelle nutzen können. Der Einfachheit halber werden die Forschungsdaten eines jeden Forschungsprojektes in unserem Beispielinstitut separat und im Rohzustand abgespeichert. Für die spätere Nutzung werden die (relevanten) Daten aufbereitet, genannt *validierte Daten*, und in einer zentralen Institutsdatenbank persistent gespeichert. Aufgrund der vielen unterschiedlichen Projekte und der langen Zeitspanne — manchmal mehrere Jahre oder Jahrzehnte —, in welcher die Daten erhoben werden, ist eine Weiterentwicklung der Datenbank kaum zu vermeiden. Für unsere Auswertungen steht uns dabei stets die aktuellste Version der Institutsdatenbank, d.h. eine materialisierte Sicht auf die relevanten Daten zum Zeitpunkt $t + 1$, zur Verfügung. Für die Auswertungen unserer Wissenschaftlerin benötigen wir jedoch eine der früheren Datenbankversionen zum Zeitpunkt t. Wir müssen die alten, für die Rekonstruktion bzw. Replikation der Publikation notwendigen Daten also zunächst aus unserer aktuellen Sicht zurückberechnen.

Auch wenn ein Großteil der Daten unserer Institutsdatenbank frei zugänglich ist, können einige wenige Datensätze oder Metadaten schützenswert sein. Der Schutz kann dabei wie im Beispiel von MOSAiC zeitlich beschränkt und durch eine Data Policy geregelt sein. In diesem Fall werden die Daten nach Ablauf der vorher definierten Frist frei zugänglich. Andere Daten, zum Beispiel personenbezogene Daten, sind unbegrenzt schützenswert. Diese sollen, auch wenn sie für eine veröffentlichte Auswertung relevant waren, niemals selbst veröffentlicht werden. Im Falle unserer Sauerstoff-Daten entspräche dies beispielsweise den einer Messung zugehörigen Metadaten wie dem Namen des Kapitäns oder der Forschungsleiterin, Messdaten aus Sperrzonen, usw.

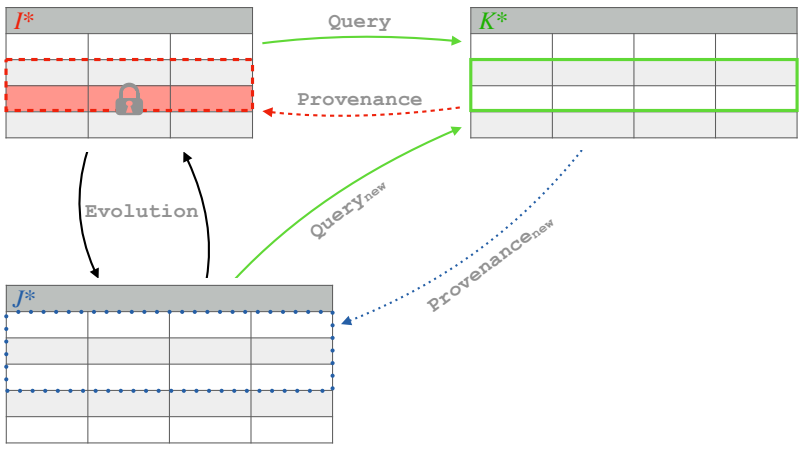


Abbildung 1.4. Auswertungsanfrage, Provenance-Anfrage und Schema-Evolution graphisch dargestellt.

Werfen wir einen Blick auf Abbildung 1.4, welche unseren Ansatz graphisch zusammenfasst: Sei I eine gegebene Quell-Instanz. Nehmen wir an, es gäbe ein veröffentlichtes Forschungsergebnis $K^* \preccurlyeq K$ (grün hervorgehoben) — K^* ist *Ausschnitt* von K (siehe Definition 6.2) — sowie eine zugehörige Auswertungsanfrage $Q : I \to K$. Die um Provenance-Informationen erweiterte Anfrage-Invertierung $Q^{-1} : K^* \to I^*$, die sogenannte *Provenance-Anfrage*, liefert die für die Rekonstruktion des Ergebnisses K^* notwendigen Quell-Tupel. Die so entstandene (minimale)

Teil-Datenbank $I^* \preccurlyeq I$ ist rot gestrichelt hervorgehoben. Bei sich entwickelnden Datenbeständen mit sich ändernden Schemata, wie der über 100 Jahre gewachsenen Sauerstoff-Datenbank am IOW, muss diese Teil-Datenbank jedoch aus der aktuellen Sicht heraus berechnet werden. Dazu invertieren wir zunächst die Evolution mittels $E^{-1} : J \to I$, bestimmen die (minimale) Teil-Datenbank I^* wie zuvor beschrieben und bringen diese anschließend durch Anwendung der Evolution $E : I^* \to J^*$ in den aktuellen Zustand. Wir erhalten die neue (minimale) Teil-Datenbank J^* (blau gepunktet hervorgehoben) sowie eine neue, transformiert Auswertungsanfrage Q_{new}, welche ebenfalls grün hervorgehoben ist. Die Einbindung von Privacy-Aspekten ist in Abbildung 1.4 durch ein Schloss symbolisiert. Der eben beschriebene Ablauf wird in Kapitel 2 sowie [AH21] noch einmal ausführlich vorgestellt. Die Einbindung von Privacy-Aspeketen, in der Abbildung durch ein Schlosssymbol dargestellt, sei an dieser Stelle vernachlässigt. Hierfür verweisen wir auf die Abschnitte 8.3 und 10.7.

Ein Kriterium zur Gewährleistung der Reproduzierbarkeit bzw. Replizierbarkeit von Forschungsergebnissen sind die FAIR-Prinzipien. Wie oben beschrieben, besteht die Schwierigkeit darin, den für die Reproduktion und Replikation eines Anfrageergebnisses relevanten Teil der originalen (Forschungs-)Daten zu bestimmen. Die Notwendigkeit der Datenreduzierung kann dabei auf hohe Kosten bei der Sammlung oder Auswertung der Daten (teuer oder elitär produziert), auf Datenschutzaspekte bei der Auswertung von Einzeldaten, auf die Wahrung des geistigen Eigentums o.ä. zurückzuführen sein. In unserem Fall beschränken sich die relevanten Auswertungen auf konjunktive Anfragen erweitert um einfache arithmetische Operationen und Aggregationsfunktionen (siehe Abschnitt 10.4). Unser Ansatz kann jedoch auf alle Arten von Anfragen erweitert werden, welche als Menge spezieller logischer Ausdrücke zurückgeführt werden können (siehe Abschnitt 11.2).

Insgesamt definieren wir für die vorliegende Arbeit drei zentrale Forschungsfragen, welche im weiteren Verlauf der Arbeit insbesondere in den Kapiteln 5 bis 10 diskutiert werden:

(I.) Wie lässt sich der minimale Teil der ursprünglichen Forschungsdatenbank berechnen, der für die Replizierbarkeit bzw. Reproduzierbarkeit der Auswertungsergebnisse dauerhaft gespeichert werden muss? Welche zusätzlichen Annotationen sind gegebenenfalls notwendig?

(II.) Wie lassen sich die Theorien zu Data Provenance und Schema-Evolution vereinheitlichen?

(III.) Wie lassen sich Data Provenance und Privacy-Aspekte im Fall der Anfrage-Invertierung kombinieren?

Um die verschiedenen Theorien zu vereinheitlichen, stellen wir die Auswertungsanfrage Q sowie die Evolution E als spezielle Schemaabbildungen (siehe Definition 3.16) dar, welche wir mit Hilfe des Chase, einer Familie von Algorithmen zur Verarbeitung von Nebenbedingungen, auswerten werden. In einem zweiten Schritt, der so genannten Backchase-Phase, verwenden wir den Chase erneut, um eine Provenance-Anfrage Q_{prov} auszuwerten und die (minimale) Teil-Datenbank zu bestimmen.

Zielsetzung der Arbeit Ziel der vorliegenden Dissertation ist die Bestimmung einer (minimalen) Teil-Datenbank mit deren Hilfe ein gegebenes Anfrageergebnis reproduziert, repliziert oder zumindest auf Plausibilität überprüft werden kann. Die (minimale) Teil-Datenbank soll dabei durch Data Provenance und zusätzliche Annotationen maximiert (Forschungsfrage I.) und bzgl. Privacy minimiert (Forschungsfrage III.) werden. Zudem soll die (minimale) Teil-Datenbank unter (Schema-)Änderungen konsistent bleiben (Forschungsfrage II.). Unsere Ergebnisse haben wir zudem praktisch in zwei getrennten Systemen umgesetzt: Während ChaTEAU eine verallgemeinerten Implementierung des Chase bietet, welche auf allgemeinen Chase-Objekten und Chase-Parametern arbeitet, erlaubt ProSA die Bestimmung einer anonymisierten (minimalen) Teil-Datenbank bei sich statischen sowie temporalen Datenbanken.

1.2. Notationen und weitere Vorbemerkungen

In Anlehnung an andere, aktuelle wissenschaftlichen Arbeiten sowie populärwissenschaftlichen Bücher verwenden wir im Folgenden das sogenannte *generische* (verallgemeinernde) *Maskulinum*. Sofern in einer konkreten Situation eine geschlechtsspezifische Differenzierung notwendig ist, wird diese dort direkt getroffen. In allen anderen Fällen ist mit der Wahl des generischen Maskulinums wie in [Fro22] beschrieben, „ausdrücklich keine Hervorhebung der Bedeutung männlicher Tätigkeit oder Zurücksetzung von Frauen erbrachten Leistungen verbunden“. Dies Schreibweise dient lediglich der besseren Lesbarkeit der vorliegenden Arbeit.

Zur Veranschaulichung konkreter Sachverhalte nutzen wir im Verlauf der Arbeit stets die selbe Datenbank einer fiktiven Universität. Um den Bezug zu bisherige Veröffentlichungen beizubehalten ist die Beispiel-Datenbank in Englisch gehalten. Die besteht aus den fünf Relationen `Student`(`studentID`, `lastName`, `firstName`, `studies`), `Participant`(`courseID`, `studentID`), `Course`(`courseID`, `title`), `Grade`(`courseID`, `studentID`, `semester`, `grade`) und `Lecturer`(`courseID`,`lecturer`), welche im Anhang A noch einmal ausführlich beschrieben sind. Alle Relationen besitzen einen einfach oder zusammengesetzten Schlüssel, welcher durch Unterstreichung hervorgehoben ist. Zusätzlich ist jedes Tupel einer Relation durch eine Provenance-ID eindeutig gekennzeichnet. Die ID setzt sich aus dem ersten Buchstaben des Relationennamens sowie einer fortlaufenden Nummer zusammen. So ist C_7 Provenance-ID des siebten Tupels (007, Law and Science) der Relation `Course`. Eine einmal vergeben Provenance-ID wird nach Löschung des zugehörigen Tupels nicht erneut vergeben. Für einen einfacheren Überblick notieren wir die Provenance-ID stets als letztes Attribut im Schema bzw. rechts der Relation.

Die in den Kapiteln 6 bis 10 vorgestellten Ergebnisse basieren hauptsächlich auf den in den Überschriften angegebenen Veröffentlichungen. Eine Liste aller relevanten Veröffentlichungen sowie der betreuten Abschlussarbeiten und studentischen Projekte (themen-bezogener Auszug) ist zudem am Ende eines jeden Kapitels hinterlegt. Die vollständigen Literaturangaben sind im Literaturverzeichnis zu finden.

Des Weiteren nutzen wir die folgenden Notationen: Alle neu definierten Begriffe werden *kursiv* hervorgehoben. Relationen- und Attributnamen sowie SQL-Code und besondere Begriffe sind durch eine entsprechende Schriftart — `Schreibmaschinenschrift`, KAPITALSCHRIFT oder ***fett-kursiv*** — gekennzeichnet. Eine Zusammenfassung aller Symbole und Abkürzungen ist ebenso wie eine Liste aller in der Arbeit untersuchten Systeme und Projekte in den Verzeichnissen in Teil V zu finden. Zusätzliches Material, wie die Beispiel-Datenbank, die Benchmark-Anfragen, die vollständigen Beweise für die Angabe der CHASE-Inversentypen, verschiedene Ein- und Ausgabe-Dateien für ChaTEAU und ProSA sind in den Anhängen A bis F zu finden. Sie wurden ausgelagert, um den Lesefluss nicht zu unterbrechen.

1.3. Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in vier Teile: *Einleitung* (Kapitel 1 und Kapitel 2), *Grundlagen und State of the Art* (Kapitel 3 bis 4), *Ergebnisse der Dissertation* (Kapitel 5 bis 10) sowie einige *Schlussbemerkungen* (Kapitel 11 und Kapitel 12). Der Schwerpunkt der Arbeit liegt auf den in Abbildung 1.5 blau hervorgehobenen Abschnitten. ChaTEAU (siehe Kapitel 9) ist ebenfalls im Rahmen der vorliegenden Arbeit entstanden. Es dient an dieser Stelle insbesondere als Implementierung des CHASE, der sowohl die Auswertungsanfragen als auch die Schema-Evolution und die Provenance-Anfragen in Form von Schemaabbildungen formalisiert.

Teil I Wir starten unsere weiteren Ausführungen mit einer ausführlichen Problembeschreibung. Hierfür definieren wir die in Abschnitt 1.1 anhand eines fiktiven Beispiels vorgestellte Problembeschreibung. Wir schließen die Einleitung mit einem Abriss der in der Dissertation erreichten Ergebnisse (siehe Kapitel 2), welche im dritten Teil der Arbeit ausführlich diskutiert werden.

Teil II Die für das Verständnis der vorliegenden Arbeit notwendigen Techniken und Begrifflichkeiten der verschiedenen Themenbereiche sind in Kapitel 3 zusammengefasst. Hierzu gehören neben dem Chase-Algorithmus, den wichtigsten Begriffen der Data Provenance und der Schema-Evolution auch verschiedene Anonymisierungsmaße und -methoden. Wir schließen den zweiten Teil mit einer Analyse verschiedener bereits existierender Chase- und Provenance-Systeme in Kapitel 4.

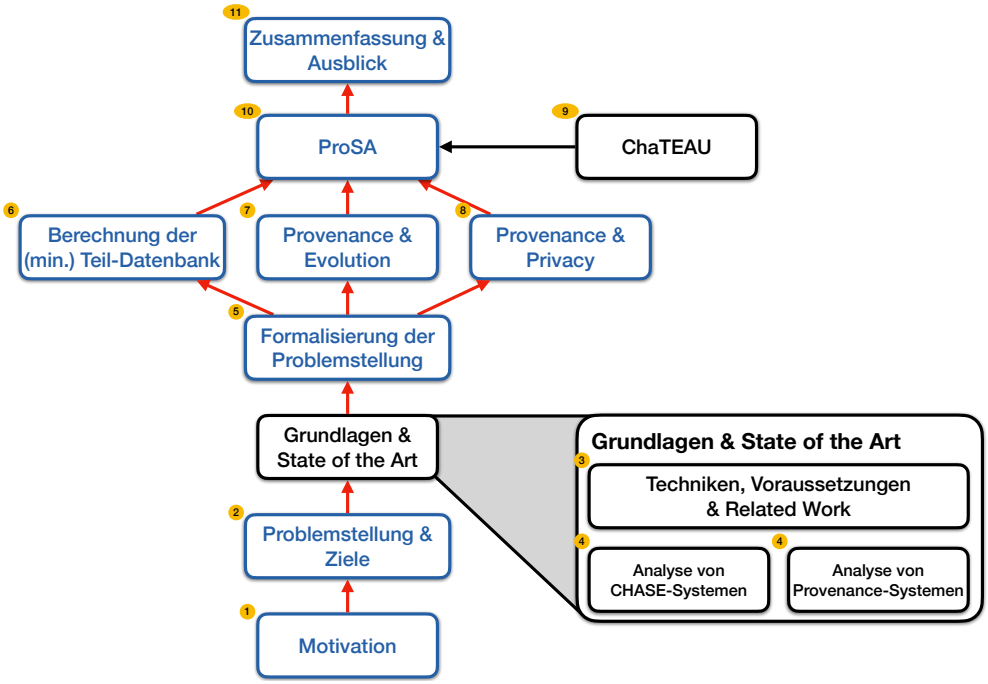


Abbildung 1.5. Aufbau der Arbeit; Schwerpunkte der Dissertation blau hervorgehoben.

Teil III Die Ergebnisse der Dissertation bilden den Hauptteil der vorliegenden Arbeit. Wir starten mit einer Formalisierung der Problemstellung. Anschließend definieren wir unsere Variante des Chase&Backchase zur Berechnung der (minimalen) Teil-Datenbank und optimieren diese durch zusätzliche Annotationen. Wir bestimmen zudem den Chase-Inversentyp der gängigsten konjunktiven Auswertungsanfragen (siehe Kapitel 6). Anschließend erweitern wir die Problemstellung auf temporale Datenbanken und bestimmen die Chase-Inversentypen der gängigsten Schemamodifikationsoperatoren (siehe Kapitel 7). Um den Verlust der Datentypen vorzubeugen, definieren wir zudem einen neuen Provenance-Typ, die sogenannte ***what***-Provenance. Eine Einschränkung der Teil-Datenbank zur Einhaltung von Privacy-Aspekten bildet den Schwerpunkt von Kapitel 8. Hierfür anonymisieren wir die ***where***-, ***why***- und ***how***-Provenance [CCT09; GT17] und schauen, welchen Einfluss dies auf die Teil-Datenbank sowie das hierauf berechnete Anfrageergebnis hat. Als Anonymisierungsmethode wählen wir die Generalisierung und als Maß die k-Anonymität [Sam01].

ChaTEAU, welches wir in Kapitel 9 vorstellen werden, ist eine Implementierung des Chase-Algorithmus. Das System ist im Rahmen dieser Dissertation in verschiedenen studentischen Projekten entstanden.

In Kapitel 10 werden die einzelnen Teil-Ergebnisse der Kapitel 6 bis 9 zu einem Gesamtablauf zusammengefügt. Zudem erweitern wir die zulässige Anfragesprache um einfache Aggregatfunktionen sowie die Gruppierung. Da ProSA nur SQL-Anfragen als Eingabe erlaubt, müssen diese zunächst für die Verarbeitung durch den Chase in eine Menge von *source-to-target tuple generating dependencies*, kurz (s-t) tgds, transformiert und invertiert werden. Gleiches gilt für die Darstellung der Evolution. Auch die Schemaevolutionsoperatoren müssen zunächst in eine Menge von (s-t) tgds transformiert werden. Ein Durchlauf am konkreten Beispiel sowie ein kurzer Überblick über die Implementierung bilden den Abschluss des Kapitels.

Teil IV Wir schließen die Arbeit mit einer Zusammenfassung und Diskussion der erzielten Ergebnisse sowie einem Ausblick auf mögliche Erweiterungen und Anwendungsmöglichkeiten des hier vorgestellten Ansatzes.

Schwerpunkte der Arbeit Aus der Problemstellung (siehe Kapitel 2) ergeben sich die Forschungsfragen (I.), (II.) und (III.), welche in den Kapiteln 6 bis 8 ausführlich diskutiert werden. Die Ergebnisse der einzelnen Fragestellungen werden anschließend in Kapitel 10 zur Beantwortung der allgemeinen Problemstellung zusammengesetzt. Dazu zählen:

- *Formale Problemstellung* (Kapitel 5)
- *Berechnung der minimalen Teil-Datenbank* (Kapitel 6)
  - Definition der ergebnisäquivalenten und tp-relaxten Chase-Inverse
  - Definition eines Algorithmus zur Invertierung von s-t tgds
  - Bestimmung und Klassifikation der Chase-Inverse der grundlegenden Auswertungsoperationen
  - Entwicklung eines Chase&Backchase zur Bestimmung der (minimalen) Teil-Datenbank
- *Vereinheitlichung von Provenance und Evolution* (Kapitel 7)
  - Berechnung der (minimalen) Teil-Datenbank im Falle von temporalen Datenbanken
  - Klassifikation relevanter Evolutionsoperatoren (im Kontext von Forschungsdatenmanagement)
  - Bestimmung und Klassifikation der Chase-Inverse der grundlegenden Evolutionsoperatoren
  - Definition der ***what***-Provenance
- *Privacy im Falle der Anfrage-Invertierung* (Kapitel 8)
  - Interview-Studie zum Verständnis der Begriffe Provenance, Privacy und Forschungsdatenmanagement
  - Vorstellung verschiedener Anonymisierungsmethoden
  - Anonymisierung der ***where***-, ***why***- und ***how***-Provenance
- *ProSA: **Pro**venance Management using **S**chema mappings with **A**nnotations* (Kapitel 10)
  - Konzeptueller Ablauf von ProSA (Vereinigung der Kapitel 6 bis 8)
  - Transformation einer SQL-Anfrage in eine Menge von s-t tgds
  - Invertierung der Auswertungsanfrage
  - Einbindung von Aggregatfunktionen
  - Einbindung von Provenance (als Erweiterung von ChaTEAU)
  - Einbindung von Privacy (Wahl eines geeigneten Anonymisierungsmaßes sowie einer -methode)
  - Erweiterung des Chase&Backchase
  - Implementierung des ProSA-Systems

Am Ende dieser Arbeit werden wir in der Lage sein, die für die Rekonstruktion einer beliebigen Auswertungsanfrage notwendigen Quelldaten zu bestimmen, auch wenn sich das zugrundeliegende Schema im Laufe der Jahre geändert haben sollte. Abhängig von der Wahl der Provenance sowie zusätzlicher Annotationen ist es uns dabei möglich, Privacy-Aspekte der Daten zu berücksichtigen.

2. Problembeschreibung und Ziele

Schwerpunkt dieses Kapitels ist die ausführliche Problembeschreibung (siehe Abschnitt 2.1) sowie Festlegung der in dieser Arbeit bearbeiteten Forschungsfragen (siehe Abschnitt 2.2). Anschließend stellen wir das Forschungsdatenmanagement als unser Anwendungsgebiet vor (siehe Abschnitt 2.3). Den Abschluss bildet schließlich eine Konkretisierung unserer Problemstellung im Kontext von Forschungsdatenmanagement, welche wir in Abschnitt 2.4 beschreiben werden.

2.1. Problembeschreibung *(basierend auf [Aug20])*

In Abschnitt 1.1 haben wir drei Forschungsfragen aufgestellt, welche im Laufe der vorliegenden Arbeit *Provenance Management unter Verwendung von Schemaabbildungen mit Annotationen* diskutiert werden sollen. Im Folgenden schauen wir uns diese noch einmal genauer an.

(I.) Berechnung der (minimalen) Teil-Datenbank Gegeben seien eine Anfrage, genannt *Auswertungsanfrage*, sowie das zugehörige Anfrageergebnis (siehe Abbildung 2.1, grün hervorgehoben). Ziel unserer Bemühungen ist die Bestimmung der für die Replikation bzw. Reproduktion des Anfrageergebnisses relevanten Quell-Tupel. Dabei werden insbesondere die für die Auswertung nicht notwendigen Tupel herausselektiert und unterdrückt. Die daraus resultierende (minimale) Teil-Datenbank (rot hervorgehoben) ist als Grundlage für eine erneute Auswertung oft jedoch nicht mehr ausreichend. Dies gilt insbesondere für den Anteil der Teil-Datenbank, welcher aus Ergebnis und Anfrage automatisch und ohne zusätzliche Annotationen berechnet wurde. Grund hierfür ist der „natürliche“ Informationsverlust, welcher durch die Anwendung von Operatoren wie der Projektion oder Aggregation entsteht. Wir benötigen daher zusätzliche Informationen, welche den Informationsverlust — konkrete Attributwerte oder ganze Quell-Tupel — ausgleichen können. Diese werden in sogenannten *Side Tables* separat abgespeichert. Zusammen mit diesen Annotationen können wir mit den Zeugenbasen der Data Provenance unter anderem Duplikate rekonstruieren, überzählige Tupel aussortieren oder konkrete Attributwerte sichern, was die Prelizierbarkeit bzw. Reproduzierbarkeit des Anfrageergebnisses sichtbar erhöht.

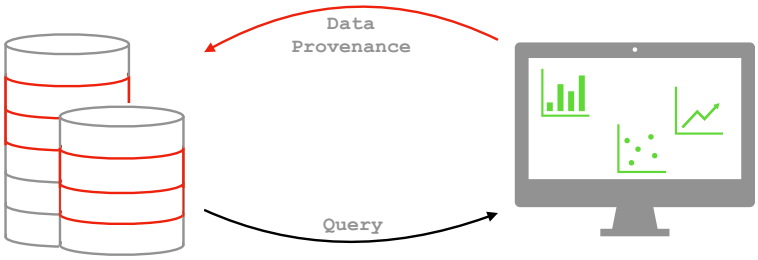


Abbildung 2.1. Berechnung der (minimalen) Teil-Datenbank [Aug20].

Ist die (minimale) Teil-Datenbank korrekt berechnet, stimmen die beiden Anfrageergebnisse — ausgewertet auf der Original-Datenbank und der (minimalen) Teil-Datenbank — überein. Zudem stellen wir folgende Forderungen, welche die (minimale) Teil-Datenbank zusätzlich erfüllen soll:

(1) die Anzahl der relevanten Quell-Tupel bleibt bzgl. der gegebenen Anfrage erhalten;

(2) die berechnete Teil-Datenbank ist Ausschnitt der Original-Datenbank, d.h. sie kann homomorph auf diese abgebildet werden;

(3) die Teil-Datenbank ist eine intensionale Beschreibung der Original-Datenbank in Bezug auf die gegebene Anfrage.

Unser Ansatz zur Bestimmung der (minimalen) Teil-Datenbank sowie ihre Erweiterung um zusätzliche Provenance-Informationen wie die Zeugenbasen und Side Tables werden in Kapitel 6 ausführlich diskutiert.

(II.) Vereinheitlichung von Provenance und Evolution Provenance-Anfragen werden in der Regel nur auf „festen“, d.h. zeit-invarianten Datenbanken gestellt. Unsere Kombination von Data Provenance und (Schema-)Evolution soll die Auswertung von Provenance-Anfragen nun auch für *temporale*, d.h. sich entwickelnden Datenbanken ermöglichen. Unsere Ergebnisse hierzu werden in Kapitel 7 ausführlich vorgestellt. Zusammengefasst kann die (minimale) Teil-Datenbank (siehe Abbildung 2.2, rot hervorgehoben) durch eine anschließende Evolution in den aktuellen Datenbankzustand (blau hervorgehoben) überführt werden. Dabei gilt: Die „neuen“ Tupel der aktuellen, evolutionierten Datenbank spielen für die Auswertung der originalen Anfrage keine Rolle. Zudem werden im Kontext des Forschungsdatenmanagements in der Regel neue Tupel erzeugt und keine Tupel gelöscht. Ein Informationsverlust auf Datenebene ist an dieser Stelle also nicht zu erwarten. Da wir auch hier die Schemaabbildungen nach Definition 3.16 sowie den Chase nach Definition 3.22 verwenden wollen, beschränken wir uns im Folgenden auf die Evolution des Datenbankschemas.

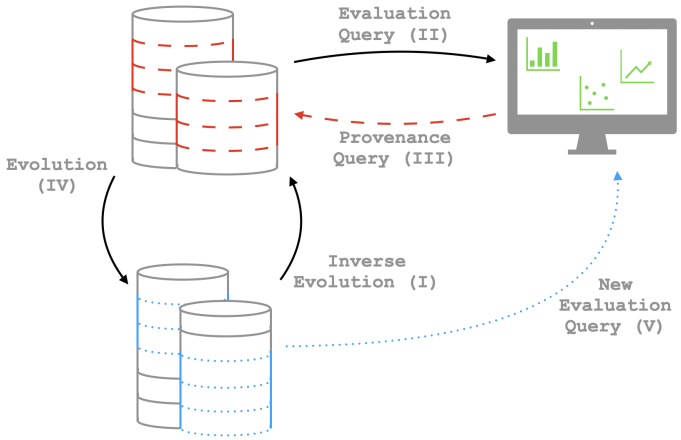


Abbildung 2.2. Kombination von Auswertungsanfrage, Evolution und zusätzlichen Provenance-Informationen [AH21].

Gegeben sei erneut eine Auswertungsanfrage, das zugehörige Anfrageergebnis sowie eine weiterentwickelte Version der originalen Datenbank zum Zeitpunkt $t + 1$. Diese beschränken wir zunächst auf die zum Zeitpunkt der Auswertung vorliegenden Tupel. Ausgehend von dieser aktuellen materialisierten Sicht (unten links) liefert die Invertierung der Evolution (**I**) die ursprüngliche Datenbankinstanz (oben links). Das Ergebnis der Auswertungsanfrage (oben rechts, grün) kann wie zuvor mit Hilfe zusätzlicher Provenance-Informationen invertiert werden (**II** + **III**) und wir erhalten die oben beschriebene (minimale) Teil-Datenbank zum Zeitpunkt t (oben links, rot). Im Falle einer temporalen Datenbank müssen die (minimale) Teil-Datenbank sowie die originale Auswerungsanfrage nun noch in das aktuelle Schema transfomiert werden (**IV** + **V**). Wir erhalten so eine (minimale) Teil-Datenbank zum Zeitpunkt $t + 1$ (unten links, blau) basierend auf der aktuellen materialisierten Sicht sowie eine neue Auswertungsanfrage (blau gepunktet). Die neue Auswertungsanfrage kann somit als Komposition der ursprünglichen Auswertungsanfrage sowie der inversen Evolution definiert werden. Es genügt daher, wenn wir uns eine der beiden (minimalen) Teil-Datenbanken (rot oder blau) abspeichern. Die jeweils Andere kann dann mit Hilfe der (inversen) Evolution aus dieser berechnet werden.

(III.) Privacy im Falle von Anfrage-Invertierung Zusätzliche Provenance-Informationen ermöglichen die Angabe einer bestmöglichen (minimalen) Teil-Datenbank, welche neben der richtigen Tupelanzahl auch konkrete Attributwerte enthalten kann. Wie viele Attributwerte rekonstruiert werden können, hängt dabei von den zur Verfügung stehenden Informationen ab (siehe Abschnitt 8.2). So können beispielsweise die Quell-Tupel aggregierter Ergebnisse nicht ohne zusätzliche Annotationen rekonstruiert werden. In vielen Fällen ist eine so exakte Rekonstruktion jedoch nicht notwendig oder gewünscht. Die Gründe hierfür sind vielfältig. So können diese Privacy-bezogen, wirtschaftlicher, finanzieller oder politischer Natur sein, wie eine von uns im Rahmen einer Bachelorarbeit durchgeführte Interview-Studie zeigt (siehe Abschnitt 8.1).

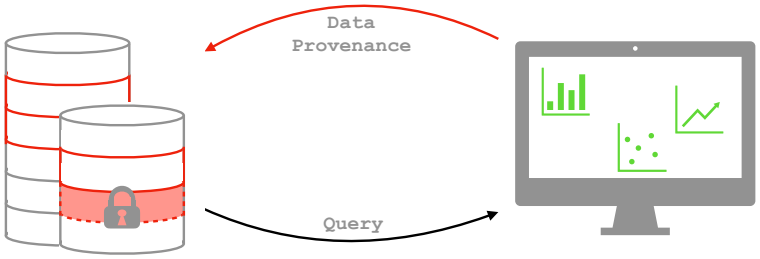


Abbildung 2.3. Privacy im Falle von Anfrage-Invertierung [Aug20].

Auch personenbezogene Daten gelten als schützenswert und dürfen nicht ohne eine gewisse Pseudonymisierung bzw. Anonymisierung veröffentlicht werden (siehe Abbildung 2.3, dargestellt durch ein Schlosssymbol). Es ist daher notwendig, eine generalisierte Teil-Datenbank zu erzeugen, die einerseits die Provenance-Kriterien erfüllt und andererseits nicht im Widerspruch zu gegebenen Privacy-Aspekten steht. Dies impliziert einen natürlichen Interessenkonflikt zwischen der Veröffentlichung der Originaldaten (*Provenance*) und dem Schutz dieser Daten (*Privacy*). Diesen werden wir in Kapitel 8 ausführlich diskutieren.

2.2. Ziele der Dissertation

Ziel der vorliegenden Dissertation ist die Bestimmung einer anonymisierten (minimalen) Teil-Datenbank, mit deren Hilfe ein Anfrageergebnis bei gegebener Auswertungsanfrage reproduziert, repliziert oder zumindest auf Plausibilität überprüft werden kann. Die Teil-Datenbank soll dabei folgen Nebenbedingungen unterliegen:

(1) die Anzahl der relevanten Quell-Tupel bleibt bzgl. der gegebenen Anfrage erhalten[1];

(2) die Teil-Datenbank kann homomorph auf die Original-Datenbank abgebildet werden;

(3) die Teil-Datenbank ist eine intensionale Beschreibung der Original-Datenbank;

(4) die Teil-Datenbank ist bzgl. Data Provenance und zusätzlichen Annotationen maximal in Bezug auf Reproduzierbarkeit bzw. Replizierbarkeit des Auswertungsergebnisses, d.h. sie enthält möglichst viele konkrete Attributwerte;

(5) die Teil-Datenbank ist bzgl. Privacy-Aspekten minimal, d.h. sie enthält so wenig konkrete Attributwerte wie möglich;

(6) im Fall temporaler Datenbanken genügt die Bestimmung einer (minimalen) Teil-Datenbank, welche durch Evolution zu jedem beliebigen Zeitpunkt materialisiert werden kann.

Unser Anwendungsgebiet, an welchem wir die drei Forschungsfragen diskutieren werden, ist das Forschungsdatenmanagement. Unser Ansatz lässt sich jedoch auch auf andere Anwendungsfelder übertragen, wie wir im Ausblick (siehe Abschnitt 11.2 noch diskutieren werden.

2.3. Anwendungsgebiet Forschungsdatenmanagement

Für den Begriff *Forschungsdatenmanagement* finden sich gleichermaßen viele Definitionen wie unterschiedliche Anwendungsfälle. So passiert es nicht selten, dass in einem Institut ohne übergreifendes Forschungsdatenmanagement jede Forschungsgruppe ihr eigenes Forschungsdatenmanagement gemäß der eigenen Bedürfnisse und Herausforderungen definiert. Wie die Technische Universität Dresden auf ihrer Website schreibt ist „das übergeordnete Ziel von Forschungsdatenmanagement jedoch immer dasselbe: eine möglichst umfassende und nachhaltige In-Wert-Setzung Ihrer Daten“ (Quelle: TU Dresden, 19.01.2022). Wir verstehen unter dem Begriff Forschungsdatenmanagement alle Aktivitäten, welche mit der Aufbereitung, Speicherung, Archivierung und Veröffentlichung von Forschungsdaten

[1] Wir gehen davon aus, dass Quell-Datenbank bereits auf die für die Auswertung der Anfrage relevanten Tupel eingeschränkt ist. Enthält sie hingegen Tupel, welche für die Auswertung der Anfrage absehbar nicht benötigt werden, kann dieses Kriterium niemals garantiert werden.

verbunden sind. Dabei handelt es sich um methodische, konzeptionelle, organisatorische sowie technische Maßnahmen und Verfahren zur Handhabung aller Daten, die sogenannten *Forschungsdaten*, welche im Rahmen der wissenschaftlichen Forschung entstehen. Der Zyklus — genannt *Forschungsdaten-Lifecycle* —, den die Daten dabei durchlaufen besteht, wie in Abbildung 2.4 dargestellt, aus sechs Phasen:

- ***Planung des Forschungsvorhabens:*** Eine gute Planung des Forschungsvorhabens sorgt für eine strukturierte Datenerfassung und -verarbeitung. Hierzu gehören unter anderem die Lokalisierung bereits vorhandener Daten, die Definition von Speicherort und -format sowie das Einholen von Nutzungsrechten an den zu sammelnden Daten.
- ***Erhebung der Daten:*** Die Erhebung und Erfassung der Daten sowie die Ergänzung von Metadaten ist fachspezifisch. So können Messdaten, Texte, Artefakte und vieles mehr Forschungsdaten sein. Auch aggregierte und ausgewertete Forschungsdaten können selbst als Forschungsdaten aufgefasst werden [ASH21b].
- ***Aufbereitung und Analyse der Daten:*** Die Aufbereitung und Auswertung der Daten bilden den Kern der wissenschaftlichen Tätigkeit. Hierzu gehören unter anderem die Digitalisierung der Daten (sofern notwendig), die Speicherung und Verwaltung, Prüfung, Validierung und Bereinigung der Daten. Sofern erforderlich, werden die Daten zudem anonymisiert. Nach der Aufbereitung können die Daten beschrieben, analysiert und interpretiert werden.
- ***Teilung (Sharing)*** und ***Publikation der Daten:*** Die Publikation der Forschungsergebnisse sowie die Veröffentlichung der Forschungsdaten ist für viele Wissenschaftler das wichtigste Standbein guter Wissenschaft. Dafür muss jedoch geklärt werden, welche Daten öffentlich bereitgestellt werden dürfen und wie der Zugang hierzu geregelt wird.
- ***Archivierung der Daten:*** Eine langfristige Archivierung sichert die dauerhafte Verfügbarkeit und Nachvollziehbarkeit der Daten zu. Die Migration der Daten sowie die Erstellung von Backups ermöglicht zudem deren Nachnutzung im eigenen System.
- ***Nachnutzung der Daten:*** Für eine fehlerfreie Nachnutzung müssen die bisherigen Ergebnisse geprüft und die Forschungen rezensiert werden. Falls noch nicht geschehen, müssen Zugriffsrechte geklärt und die Daten gegebenenfalls öffentlich bereitgestellt werden (*Open Data*).

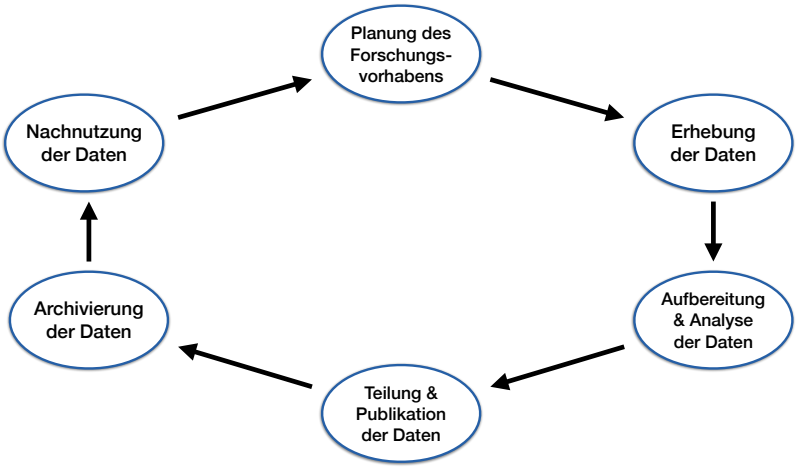


Abbildung 2.4. Forschungsdaten-Lifecycle.

Weitere Details können unter anderem auf `https://www.forschungsdaten.info` nachgelesen werden. Dies ist eine Website, welche das Forschungsdatenmanagement vieler namenhafter Institute und Universitäten aus dem Süden Deutschlands sowie Österreich und der Schweiz beschreibt (Stand: 20.01.2022).

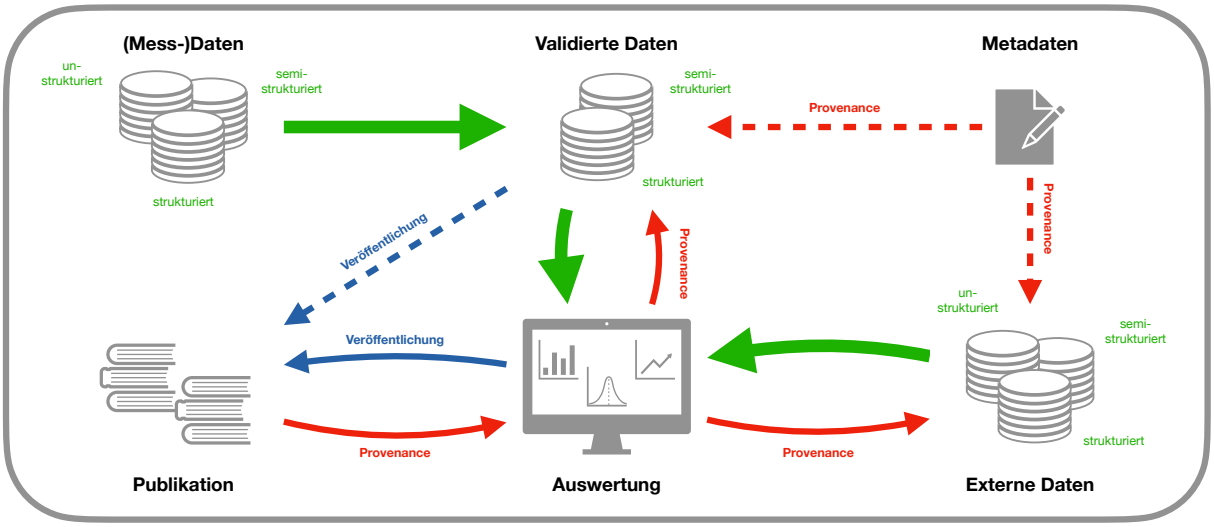


Abbildung 2.5. Provenance im Forschungsdatenmanagement: Datenfluss (grün), Veröffentlichung (blau) und Provenance (rot); gestrichelte Pfeile sind optional.

Ein „professioneller Umgang mit Forschungsdaten schützt vor Datenverlust sowie Missbrauch und ermöglicht ein späteres Nachvollziehen der Forschungsergebnisse und eine zukünftige Nachnutzung der Daten. Werden die Grundsätze des Forschungsdatenmanagements bei der Planung und Umsetzung des Forschungsprojektes beachtet, kann die Gefahr eines Datenverlustes minimiert werden“ (Quelle: Universität Kassel, 20.01.2022). Ein professionelles Forschungsdatenmanagement wird dabei immer häufiger von Forschungsinstituten und -förderern, Journalen oder Konferenzen gefordert. Bespiele hierfür sind etwa der DFG-Kodex *Leitlinien zur Sicherung guter wissenschaftlicher Praxis*[2] oder die Forschungsleitlinien[3] und Fortbildungsangebote[4] der Universitätsbibliothek Rostock. Leitlinien für eine optimale Aufbereitung der Daten für die Verarbeitung von Mensch und Maschine bieten beispielsweise die FAIR-Prinzipien[5]. Eine Sammlung von Werkzeugen sowie weiteren Informationen zum Management von Forschungsdaten findet sich zudem unter `http://www.digipres.org` (Stand: 20.01.2022).

Provenance im Forschungsdatenmanagement Ein neues Forschungsprojekt beginnt in der Regel mit der Definition einer Forschungsfrage. Nach der Konzeptentwicklung schließt sich dann eine (aufwendige) Datenerhebung an. Der Weg von den (Mess-)Daten bis hin zur Publikation umfasst dabei einige Zwischenschritte. Die unstrukturiert oder (semi-)strukturiert vorliegenden Daten werden zunächst aufbereitet und validiert, d.h. auf Plausibilität geprüft, sodass diese anschließend (semi-)strukturiert vorliegen. Sowohl bei der Datenerhebung als auch bei der Validierung werden zudem Metadaten erhoben, welche zwar nicht veröffentlicht werden, aber für spätere Provenance-Analysen von Interesse sein. In vielen Projekten werden neben den eigenen Daten zusätzlich noch externe Daten verarbeitet. Validierte und externe Daten fließen dabei gleichermaßen in die Auswertung mit ein. Es entsteht der in Abbildung 2.5 grün hervorgehobene Datenstrom.

Für ein gutes Forschungsdatenmanagement sollen neben dem Auswertungsergebnis selbst in Zukunft auch die validierten Daten veröffentlicht werden (blau hervorgehoben). Eine Veröffentlichung der externen Daten steht uns an dieser Stelle jedoch nicht zu, wie der fehlende blaue Pfeil in Abbildung 2.5 zeigt. Erneut ist die Publikation Ausgangspunkt unserer Provenance-Auswertungen. Sie ermöglichen uns die Rückverfolgung eines veröffentlichten Ergebnisses über die Auswertung bis hin zu den validierten oder externen Daten. Eine Rekonstruktion der originalen (Mess-)Daten hingegen ist in der Regel nicht ohne Weiteres möglich.

[2] DFG-Kodex: `https://www.dfg.de/foerderung/grundlagen_rahmenbedingungen/gwp/`
[3] Leitlinien und Regelungen der Universität Rostock: `https://www.uni-rostock.de/universitaet/organisation/rechtsgrundlagen/leitlinien-und-empfehlungen/`
[4] Universitätsbibliothek: `https://www.ub.uni-rostock.de/wissenschaftliche-services/forschungsdaten/ueberblick/`
[5] FAIR-Prinzipien: `https://www.go-fair.org/fair-principles/`

Unser Ziel ist daher die Bestimmung der für die Nachvollziehbarkeit, Replizierbarkeit bzw. Reproduzierbarkeit der Publikation notwendigen Quell-Tupel. Ob es sich hierbei um validierte oder externe Daten handelt, spielt für uns keine Rolle. Da wir keinen Einfluss auf die Veröffentlichung der externen Daten haben, beschränken wir uns jedoch auf die Rekonstruktion der validierten Daten. Mögliche Ansätze zum Einsatz von Provenance sind in Abbildung 2.5 rot hervorgehoben.

Einschränkungen im Forschungsdatenmanagement Der in dieser Arbeit vorgestellte Ansatz zur Bestimmung einer (minimalen) Teil-Datenbank findet in vielen Bereichen Anwendung, so auch im Forschungsdatenmanagement, einem wichtigen „Werkzeug zur Organisation der eigenen Forschung. Mit der Generierung von Forschungsdaten verfolgen Forscher stets ein konkretes Forschungsziel. Die Hürden, die es dabei zu überwinden gilt, sind jedoch nicht immer von vornherein abschätzbar oder sichtbar. So geschieht es, dass Erwartungen an den Forschungsprozess und die Alltagsrealität oftmals auseinander klaffen.“(Quelle: TU Dresden, 30.07.2023) Bezogen auf unsere Forschungsdaten kann dies bedeuten, dass zu viele oder zu wenig Daten erhoben werden oder diese fehlerhaft und nicht aussagekräftig genug sind. Ein gutes Datenmanagement ist daher essentiell, auch über die Projektlaufzeit hinaus.

Schauen wir uns die für die Auswertung relevanten Daten, d.h. die von uns bestimmte (minimale) Teil-Datenbank, noch einmal genauer an. Sie erhöht die Nachhaltigkeit unseres Forschungsergebnisses sowie unserer Forschungsdaten und bietet eine zusätzliche Kontrollmöglichkeit im Review-Prozess von Journalen und Konferenzen. Dafür soll die (minimale) Teil-Datenbank möglichst exakt auf die gegebene Forschungsfrage angepasst sein, um so Speicherplatz-Kriterien — viele Bibliotheken, Konferenzen und Journale haben gar nicht die Möglichkeit große Datenmengen abzuspeichern — und Privacy-Aspekte — wir denken hier an die Anonymisierung von personenbezogenen Daten (wie im üblichen Krankenhaus-Beispiel) oder das gezielte „Zurückhalten“ von Daten zur Einhaltung von projektspezifischen Data Policies (wie etwa bei MOSAiC) — zu gewährleisten.

Hierfür sind jedoch einige Einschränkungen zu beachten. So können externe Daten, wie in Abbildung 2.5 dargestellt, zwar rekonstruiert werden, dürfen in einige Fällen aus Privacy-Gründen jedoch nicht veröffentlicht werden. Wir vernachlässigen sie daher in unseren folgenden Überlegungen. Sollte eine Veröffentlichung jedoch möglich sein, werden sie genauso behandelt, wie die „eigenen“ validierten Daten. Anders verhält es sich bei den Metadaten. Diese können aus der Anfrage sowie dem publizierten Ergebnis in der Regel nicht mehr rekonstruiert werden. Sie werden daher ebenfalls vernachlässigt.

Zudem soll stets das gesamte (veröffentlichte) Forschungsergebnis rekonstruiert werden können. Die Rekonstruktion einzelner Tupel ist zwar möglich, im Falle des Forschungsdatenmanagements jedoch unüblich. Weitere Einschränkungen gelten zudem bei der Einbindung von Evolution und Privacy. So halten sich die Schemaänderungen trotz langer Laufzeit in Grenzen. Bei unserer Studie einer konkreten Forschungsdatenbank des Leibniz-Institutes für Ostseeforschung Warnemünde ergab sich beispielsweise eine Frequenz von einer Schemaänderung alle 10 bis 20 Jahre [Aug+20]. Die Frequenz variiert selbstverständlich abhängig vom Projekt, liegt in der Regel jedoch im Bereich von mehreren Monaten oder Jahren (je nach Laufzeit des Projektes). Ein gutes Forschungsdatenmanagement kann also dabei helfen, die Anzahl der geplanten Schemaänderungen gering zu halten. Bei den Operationen selbst handelt es sich häufig um das Hinzufügen von Attributen, das Erstellen neuer Relationen sowie das Zusammenlegen von Relationen und Attributen. Löschungen von Relationen oder Attributen treten hingegen eher selten auf.

Viele Forschungsprojekte arbeiten zwar nicht auf personenbezogenen Daten, unterliegen aber dennoch den verschiedensten Privacy-Policen. Der Begriff *Privacy* muss daher für unseren Anwendungsfall angepasst bzw. erweitert werden. Eine entsprechende Definition folgt in Kapitel 8. Auch die Art der Auswertungsanfragen kann eingeschränkt werden. Wir beschränken uns auf die konjunktiven SPJU-Anfragen, d.h. Anfragen mit **S**elektion, **P**rojektion, Verbund (**J**oin) und Vereinigung (**U**nion), welche wir um die Aggregatfunktionen `MIN`, `MAX`, `COUNT`, `SUM` und `AVG`, einfache skalare Funktionen sowie die Gruppierung erweitern. Auf dieser Grundlage können auch viele Anfragen aus den Bereichen Data Science und Machine Learning verarbeitet werden. Wie solche Anfragen auf die klassischen SPJUA-Anfragen, d.h. SPJU-Anfragen mit **A**ggregation, transformiert werden können, kann unter Anderem in [MH17; MMH19] nachgelesen werden.

2.4. Problemstellung im Kontext des Forschungsdatenmanagements

Fassen wir die Problemstellung sowie ihre Einschränkungen im Zusammenhang mit der Verwaltung von Forschungsdaten anhand von Abbildung 2.6 noch einmal zusammen: Gesucht ist eine anonymisierte (minimale) Teil-Datenbank I^*_{anon}, welche bzgl. zusätzlicher Provenance-Informationen maximiert und bzgl. gegebener Privacy-Aspekte minimiert werden soll. Gegeben seien hierfür eine Auswertungsanfrage Q, eine Datenbankinstanz I zum Zeitpunkt t mit zugehörigem Anfrageergebnis $K = Q(I)$ sowie eine Evolution E mit $J = E(I)$. Ziel des hier vorgestellten Verfahrens ist die Rekonstruktion von $I^* \preccurlyeq I$[6] (zum Zeitpunkt t) bzw. $J^* \preccurlyeq J$ (zum Zeitpunkt $t+1$), wobei I^* homomorph auf I und J^* homomorph auf J abgebildet werden kann. Details hierzu folgen in Kapitel 5. Dabei ergibt sich I^* durch Invertierung der Auswertungsanfrage Q und J^* durch Invertierung der Evolution E. Beide Instanzen können durch zusätzliche Provenance-Informationen sowie Privacy-Aspekte ergänzt bzw. eingeschränkt werden, wobei folgende Einschränkungen und Bedingungen gelten (siehe Abbildung 2.6):

- Es wird stets das gesamte Ergebnis invertiert, d.h. $K^* = K$ (grün hervorgehoben).
- Die Anfrage Q^{-1} ist die direkte Invertierung von Q (rot hervorgehoben). Sie ist in vielen Fällen eine Quasi-Inverse (siehe Definition 6.1). Die Anfrage Q^{-1}_{prov} ist die um Provenance-Informationen erweiterte Invertierung der Auswertungsanfrage Q. Auch sie ist in den meisten Fällen eine Quasi-Inverse.
- Die um Provenance erweiterten (minimalen) Teil-Datenbanken $I^* \preccurlyeq I$ (rot hervorgehoben) und $J^* \preccurlyeq J$ (blau hervorgehoben) lassen sich homomorph auf I und J abbilden. Ein Tupel aus I^* muss somit (bis auf Nullwerte) auch in I enthalten sein. Genaueres hierzu folgt in Kapitel 5.
- Die anonymisierte (minimale) Teil-Datenbank bezeichnen wir mit I^*_{anon} (symbolisiert durch das graue Schloss). Sie ergibt sich in unserem Fall durch Generalisierung und Intensionalisierung, kann aber auch durch andere Anonymisierungsmethoden erzeugt werden. Mehr Details hierzu finden sich in Kapitel 8.
- Schema-Evolutionen sind insbesondere bei Langzeitstudien nichts ungewöhnliches. Da eine Evolution innerhalb der Projektlaufzeit jedoch eher ungewöhnlich ist, ist ihre Frequenz meist nicht sonderlich hoch. Näheres hierzu findet sich in Kapitel 7.
- Unser Ansatz arbeitet auf einer speziellen Form von eingebetteten Abhängigkeiten, welche neue Tupel erzeugen oder bestehende Tupel bereinigen können, sogenannte *(s-t) tgds* und *egds* (siehe Definition 3.13). Jede Auswertungsanfrage sowie jede Evolution, welche als solche Abhängigkeit geschrieben werden kann, kann mit unserem Ansatz verarbeitet werden. Hierzu gehörigen insbesondere konjunktive SPJUA-Anfragen (siehe oben). Details hierzu folgen in Kapitel 6.

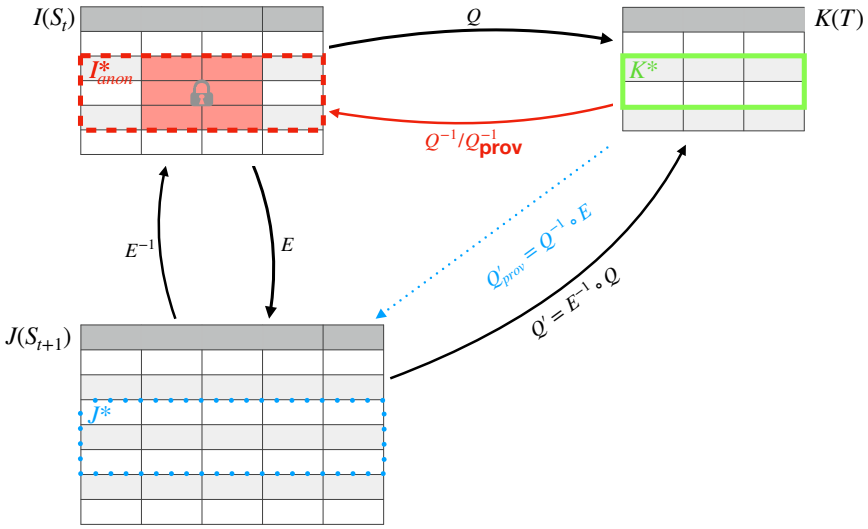


Abbildung 2.6. Berechnung der (minimalen) Teil-Datenbanken $I^* \preccurlyeq I$ und $J^* \preccurlyeq J$ unter Berücksichtigung von zusätzlichen Provenance-Informationen und Privacy-Aspekten bei sich ändernden Datenbanken.

[6] I^* ist Ausschnitt von I (siehe Definition 6.2)

Der Gesamtablauf, welcher in Abbildung 2.6 dargestellt ist, wird in Kapitel 10 ausführlich beschrieben. Unser konkretes Ziel ist die Bestimmung einer anonymisierten (minimalen) Teil-Datenbank I^*_{anon} mit deren Hilfe wir die Reproduzierbarkeit, Replizierbarkeit oder Plausibilität eines Forschungsergebnisses garantieren können. Dies erhöht unter anderem die Nachhaltigkeit der Forschungsdaten und bietet ein zusätzliche Kontrollmöglichkeit im Review-Prozess von Journalen und Konferenzen.

Bevor wir die hier vorgestellte Problemstellung in Kapitel 5 formalisieren wollen, benötigen wir jedoch ein grundlegendes Verständnis für die Begriffe *Provenance*, *Privacy* und *Evolution*. Grundlage für die Verarbeitung der Auswertungsanfragen sowie der Evolution wird der Chase-Algorithmus sein. Er wird ebenfalls in Kapitel 3 ausführlich vorgestellt werden. Anschließend testen wir in Kapitel 4 verschiedene Chase- und Provenance-Systeme, bevor wir ab Kapitel 5 die Ergebnisse der vorliegenden Dissertation vorstellen werden.

Teil II.

Grundlagen und State of the Art

3. Techniken, Voraussetzungen und Related Work

Die Idee des Promotionsprojekts beruht auf der Kombination verschiedener Grundtechniken. Hierzu gehören neben dem Chase (siehe Abschnitt 3.3), auch die Chase-inversen Schemaabbildungen nach Fagin et al. (siehe Abschnitt 3.2) sowie die Provenance-Polynome und Zeugenbasen der Data Provenance (siehe Abschnitt 3.4). Mit Hilfe dieser Techniken berechnen wir in Kapitel 6 für eine gegebene Auswertungsanfrage Q sowie eine Datenbankinstanz I die zugehörige (minimale) Teil-Datenbank I^*, dessen Angabe wir in Kapitel 7 auch für temporale Datenbanken ermöglichen werden. Die hierfür benötigten Schemaevolutionen gehören ebenso zu den hier vorgestellten Grundlagen (siehe Abschnitt 3.5), wie die Anonymisierungsmaße und -methoden (siehe Abschnitt 3.6), welche wir zur Berücksichtigung von Privacy-Aspekten benötigen werden (siehe Kapitel 8).

Wir interpretieren die Auswertungs- und Evolutionsanfragen als konjunktive Formeln erster Ordnung. Wir beginnen daher mit einem kurzen Abriss der Logik erster und zweiter Stufe in Abschnitt 3.1 sowie den wichtigsten Begriffen aus dem Bereich relationale Datenbanken in Abschnitt 3.2. Anschließend stellen wir die Grundtechniken der vorliegenden Dissertation vor: Chase, Data Provenance, Schema-Evolution sowie Privacy-Aspekte.

3.1. Logik erster und zweiter Stufe

Die *klassische Logik* ist durch zwei Eigenschaften charakterisiert:

- Jede Aussage hat einen von genau zwei Wahrheitswerten, meist „falsch“ oder „wahr“ (*Prinzip der Zweiwertigkeit* [Wuc99]).
- Der Wahrheitswert jeder zusammengesetzten Aussage ist eindeutig durch die Wahrheitswerte ihrer Teil-Aussagen bestimmt (*Prinzip der Extensionalität*, [Var03])

Der Teil der klassischen Logik, welcher sich mit der Struktur von Aussagen beschäftigt heißt *Aussagenlogik*. Diese untersucht Aussagen auf ihre Zusammensetzung aus anderen, *atomaren*, d.h. nicht weiter zerlegbaren, Teil-Aussagen. So besteht die Aussage *„Max studiert Informatik und hat in Mathe eine 3.0 geschrieben“* beispielsweise aus den Teil-Aussagen *„Max studiert Informatik“* und *„Max hat in Mathe eine 3.0 geschrieben“*. Beide Teil-Aussagen sind atomar, da sie selbst nicht weiter zerlegbar sind. Die Untersuchung der Aussagen auf ihre innere Struktur ist Aufgabe der sogenannten *Prädikatenlogik*, einer Erweiterung der Aussagenlogik. Wir beschränken uns in dieser Arbeit insbesondere auf die Logik erster Stufe.

Logik erster Stufe Die *Logik erster Stufe* ermöglicht es, mathematische Strukturen wie Gruppen, Körper, Vektorräume oder Graphen sowie die gängigen Modelle der Informatik wie Transitionssysteme, endliche Automaten, relationale Datenbanken oder Schaltkreise zu beschreiben. Wir konzentrieren uns hier auf die Syntax-Definitionen. Definitionen und Aussagen über die Semantik können etwa in [EFT07] nachgelesen werden.

Definition 3.1 (Alphabet, 1. Stufe, [EFT07])**.** Das *Alphabet einer Sprache erster Stufe* umfasst folgende Symbole:

(a) eine Menge von *Variablen* $\mathcal{V}$: $v_0, v_1, v_2, ...$;

(b) die *Junktoren*: $\neg$, $\wedge$, $\vee$, $\rightarrow$ und $\leftrightarrow$;

(c) die *technische Zeichen*: $\equiv$, (und);

(d) eine (möglicherweise leere) Menge von n-stelligen *Prädikats-* oder *Relationssymbolen* $\mathcal{P}$;

(d) eine (möglicherweise leere) Menge von n-stelligen *Funktionssymbolen* $\mathcal{F}$;

(f) eine (möglicherweise leere) Menge von *Konstanten* $\mathcal{C}$.

Die unter (a) bis (c) genannten Zeichen bezeichnen wir mit $\mathcal{A}$, die unter (d) bis (f) genannten Zeichen mit $\mathcal{S}$, wobei gilt: $\mathcal{S}$ kann leer sein und $\mathcal{A} \cap \mathcal{S} = \emptyset$. Weiter gilt: Eine Sprache wird durch ihre *Symbolmenge* $\mathcal{S}$ bestimmt, wobei sich das Alphabet $\mathcal{A}_\mathcal{S}$ aus $\mathcal{A}_\mathcal{S} = \mathcal{A} \cup \mathcal{S}$ ergibt. ■

Sei nun $\mathcal{A}$ ein gegebenes Alphabet. Dann heißt eine endliche lineare Reihe von Zeichen aus $\mathcal{A}$ *Zeichenreihe* oder *Wort* über $\mathcal{A}$. Die Menge der Zeichenreihen über $\mathcal{A}$ bezeichnen wir mit $\mathcal{A}^*$. Weiter heißt eine Menge M abzählbar, wenn sie nicht endlich ist und es eine surjektive Abbildung von $\mathbb{N}$ auf M gibt. Eine Menge M heißt *höchstens abzählbar*, wenn sie endlich oder abzählbar ist.

Definition 3.2 ($\mathcal{S}$-Terme, 1. Stufe, [EFT07])**.** *$\mathcal{S}$-Terme* sind genau diejenigen Zeichenreihe in $\mathcal{A}_\mathcal{S}^*$, welche durch eine endliche Folge von Anwendungen der folgenden Regeln entstehen können:

(T1) Jede Variable ist ein $\mathcal{S}$-Term.

(T2) Jede Konstante aus S ist ein $\mathcal{S}$-Term.

(T3) Sind t_1, ..., t_k $\mathcal{S}$-Terme und ist f ein n-stelliges Funktionssymbol aus $\mathcal{S}$, dann ist auch $f(t_1, ..., t_k)$ ein $\mathcal{S}$-Term.

Nichts sonst ist ein $\mathcal{S}$-Term. Die Menge der $\mathcal{S}$-Terme bezeichnen wir mit $T^\mathcal{S}$. ■

Definition 3.3 ($\mathcal{S}$-Ausdrücke, 1. Stufe, [EFT07])**.** $\mathcal{S}$-Ausdrücke sind genau diejenigen Zeichenreihen in $\mathcal{A}_\mathcal{S}^*$, welche durch eine endliche Folge von Anwendungen der folgenden Regeln entstehen können:

(A1) Für zwei $\mathcal{S}$-Terme t_1, t_2 ist $t_1 \equiv t_2$ ein $\mathcal{S}$-Ausdruck.

(A2) Sind $t_1, ..., t_k$ $\mathcal{S}$-Terme und ist p ein n-stelliges Prädikatssymbol, dann ist $p(t_1, ..., t_k)$ ein $\mathcal{S}$-Ausdruck.

(A3) Ist φ ein $\mathcal{S}$-Ausdruck, dann ist auch $\neg\varphi$ ein $\mathcal{S}$-Ausdruck.

(A4) Sind φ und ψ $\mathcal{S}$-Ausdrücke, dann auch $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \to \psi)$ und $(\varphi \leftrightarrow \psi)$ $\mathcal{S}$-Ausdrücke.

(A5) Ist φ ein $\mathcal{S}$-Ausdruck und x eine Variable, dann sind $(\exists x)\varphi$ und $(\forall y)\psi$ $\mathcal{S}$-Ausdrücke.

Nichts sonst ist ein $\mathcal{S}$-Ausdruck. $\mathcal{S}$-Ausdrücke, welche mit (A1) und (A2) gewonnen werden können heißen *atomar*, weil sie nicht aus anderen $\mathcal{S}$-Ausdrücke zusammengesetzt werden können. ■

Definition 3.4 (Prädikatenlogik erster Stufe, [EFT07])**.** Sei $\mathcal{L}_I^\mathcal{S}$ die Menge der $\mathcal{S}$-Ausdrücke, dann heißt $\mathcal{L}_I^\mathcal{S}$ die *Sprache der Prädikatenlogik erster Stufe* über $(\mathcal{V}, \mathcal{P}, \mathcal{F}, \mathcal{C})$. ■

Ist der Bezug zur Symbolmenge $\mathcal{S}$ klar, so schreiben wir statt $\mathcal{S}$-Term auch *Term*. Gleiches gilt für die $\mathcal{S}$-Ausdrücke. $\mathcal{S}$-Ausdrücke werden oft als *Formeln* bezeichnet. Außerdem gilt: Ist $\mathcal{S}$ höchstens abzählbar, so sind $\mathcal{T}^\mathcal{S}$ und $\mathcal{L}^\mathcal{S}$ abzählbar.

Logik zweiter Stufe Die *Logik zweiter Stufe* erweitert die Prädikatenlogik erster Stufe um die Möglichkeit der Quantifizierung über alle Relationen. Sie ist daher echt ausdrucksstärker als die der ersten Stufe. Sie ermöglicht die Darstellung der Peano-Axiome, wichtige Sätze wie der Kompaktheitssatz (siehe [EFT07]) gelten allerdings nicht mehr.

Definition 3.5 (Alphabet, 2. Stufe, [EFT07])**.** Sei $\mathcal{S}$ eine Symbolmenge aus Prädikats- und Funktionssymbolen sowie Konstanten wie in $\mathcal{L}_I^\mathcal{S}$. Das *Alphabet einer Sprache zweiter Stufe* enthält neben $\mathcal{A}_\mathcal{S}$ zusätzlich eine abzählbare Menge von n-stelligen *Relationssymbolen* $\mathcal{V}^n$. ■

Definition 3.6 ($\mathcal{S}$-Ausdrücke, 2. Stufe, [EFT07])**.** Es gelten die Regeln (A1) bis (A5). Diese werden wie folgt erweitert:

(A6) Ist X eine n-stellige Relationsvariable und sind $t_1, ..., t_n$ $\mathcal{S}$-Terme, so ist $Xt_1...t_n$ ein $\mathcal{S}$-Ausdruck. Im Datenbank-Kontext schreiben wir auch $X(t_1, ..., t_n)$.

(A7) Ist φ ein $\mathcal{S}$-Ausdruck und ist X ein Relationsvariable, so ist $\exists\varphi$ ein $\mathcal{S}$-Ausdruck. ■

Definition 3.7 (Prädikatenlogik, 2. Stufe, [EFT07])**.** Sei $\mathcal{L}_{II}^{\mathcal{S}}$ die Menge der erweiterten $\mathcal{S}$-Ausdrücke. Dann heißt $\mathcal{L}_{II}^{\mathcal{S}}$ die *Sprache der Prädikatenlogik zweiter Stufe* über $(\mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P})$. ■

Sei $\mathcal{L}_{II}$ das durch die Sprachen $\mathcal{L}_{II}^{\mathcal{S}}$ und ihre Modellbeziehung bestimmte logische System, die sogenannte *Logik zweiter Stufe*. Analog sei $\mathcal{L}_I$ die Logik erster Stufe.

Beispiel 3.1 (Peano-Axiome)**.** Die Peano-Axiome (P1) bis (P3) lassen sich nicht in der Prädikatenlogik erster Stufe formulieren. Grund hierfür ist das Induktionsaxiom (P3). Es beschreibt eine Aussage über alle Teilmengen der betrachteten Grundmenge und kann ohne Regel (A7) nicht quantifiziert werden. (P3) ist ein Ausdruck der zweiten Stufe und kann mit Hilfe der Symbolmenge $\mathcal{S} = \{0, \varphi\}$ dargestellt werden. Dabei gilt: 0 ist Konstantensymbol, genannt *Nullelement*, und φ einstelliges Funktionssymbol, genannt *Nachfolgerfunktion*.

(P1) $\forall x\ \neg\varphi x \equiv 0$

(P2) $\forall x\ \forall y\ (\varphi x \equiv \varphi y \rightarrow x \equiv y)$

(P3) $\forall X\ ((X0 \wedge \forall x(Xx \rightarrow X\varphi x)) \rightarrow \forall y \forall x)$ □

Hinweis: Wir haben uns in diesem Abschnitt bewusst an die „klassische“ Notation gehalten, wie sie in der Mathematik üblich ist. Im Folgenden werden wir jedoch hauptsächlich die von Fagin et al. sowie Green et al. eingeführte Notation verwenden. Dies gilt insbesondere bei der Definition von (s-t) tgds (siehe Definition 3.13), egds (siehe Definition 3.14) sowie der so-tgds (siehe Definition 3.15) in Abschnitt 3.2.

3.2. Relationale Datenbanken

„Eine Datenbank ist eine Sammlung von Daten, welche so organisiert ist, dass sie relevante Aspekte der Realität modelliert und Prozesse unterstützt, die diese Informationen benötigen“, [GMS12]. Das bekannteste Datenbankmodell ist das 1970 eingeführte *Relationenmodell* [Cod70]. Dieses besteht aus einer Menge von Relationen (siehe Abbildung 3.1) sowie einer Menge von Integrationsbedingungen (siehe Definition 3.8 bis 3.15) zwischen und innerhalb der Relationen. Eine *Relation* kann anschaulich als Tabelle verstanden werden. Die Spalten entsprechenden den *Attributen* A_i und die verschiedenen Zeilen den *Tupeln* t_i der Relationen. Eine Zelle entspricht schließlich einem konkreten *Attributwert* aus $\mathbb{D}(A_i)$.

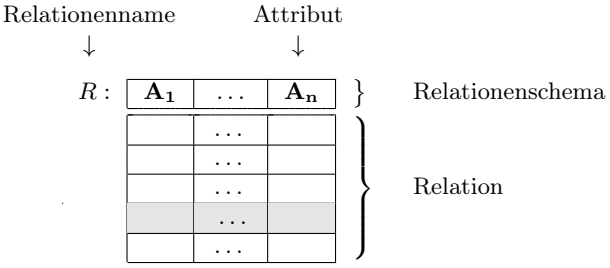


Abbildung 3.1. Relationenschema mit Relationennamen R und Attributen A_i. Relation bestehend aus verschiedenen Tupeln; ein Tupel ist grau hinterlegt.

Sei hierfür $\mathcal{U} = \{A_1, \ldots A_n\}$ eine nicht leere, endliche Menge, das sogenannte *Universum*. Die Elemente $A_i \in \mathcal{U}$ mit $n \in \mathbb{N}$ nennen wir *Attribute*, deren Werte über der *Domäne* $\mathbb{D}(A_i)$ definiert sind, und eine endliche Folge von Relationssymbolen R_i, wobei jedes $R_i \subseteq \mathcal{U}$ eine feste Stelligkeit hat, *Datenbankschema* S. Weiter heißt eine konkrete Menge von Attribute aus $\mathcal{U}$ *Relationenschema* und eine Folge $(R_1^I, ..., R_k^I)$, wobei jedes R_i^I eine endliche Relation mit der gleichen Arität wie R_i ist, *Instanz* I über R. Dabei besteht jedes Tupel $(x_1, ..., x_n)$ in R_i^I aus *Konstanten* c_i, später auch aus *Nullwerten* η_i (siehe Abschnitt 3.3).

Die Operationen, welche über einer gegebenen Menge von Relationen ausgeführt werden können, sind in der sogenannten *relationalen Algebra* definiert. Hierzu gehören unter anderem das Filtern, Verknüpfen, Umbenennen oder Aggregieren von Relationen. Seien hierzu α, c zwei reelle Konstante, R_1, R_2 zwei Relationenschemata, t_i ein konkretes Tupel, $A_i, A_j \in \mathcal{U}$ zwei Attribute, $\theta \in \{<, \leq, =, \neq, \geq, >\}$ ein Vergleichsoperator und $\omega \in \{+, -, \cdot, :\}$ eine skalare Operation. Dann ist die Relationenalgebra unter den in Tabelle 3.1 zusammengefassten Operationen abgeschlossen.

Attributselektion:	$\sigma_{A_i \theta A_j} := \{t \mid t \in r \wedge t(A_i) \theta t(A_j)\}$
Konstantenselektion:	$\sigma_{A_i \theta c} := \{t \mid t \in r \wedge t(A_i) \theta c\}$
Projektion:	$\pi_{A_i}(r) := \{t(A_i) \mid t \in r\}$
Natürlicher Verbund:	$r_1 \bowtie r_2 := \{t \mid t(R_1 \cup R_2) \wedge \exists t_1 \in r_1 : t_1 = t(R_1) \wedge \exists t_2 \in r_2 : t_2 = t(R_2)\}$
Kartesisches Produkt:	$r_1 \times r_2 := \{t \mid t(R_1 \cup R_2) \wedge R_1 \cap R_2 = \emptyset$ $\wedge \exists t_1 \in r_1 : t_1 = t(R_1) \wedge \exists t_2 \in r_2 : t_2 = t(R_2)\}$
Schnittmenge:	$r_1 \cap r_2 := \{t \mid t \in r_1 \wedge t \in r_2\}$
Vereinigung:	$r_1 \cup r_2 := \{t \mid t \in r_1 \vee t \in r_2\}$
Differenz:	$r_1 - r_2 := \{t \mid t \in r_1 \wedge t \notin r_2\}$
Umbenennung:	$\beta_{A_i \leftarrow A_j}(r) := \{t' \mid \exists t \in r : r'(R - \{A_i\}) = t(R - \{A_i\}) \wedge t'(A_j) = t(A_i)\}$
skalare Operationen:	$\alpha \omega A_i := \{t \mid t \in r \wedge \alpha \omega t(A_i)\}$
Aggregatfunktionen:	`MIN`/`MAX`: kleinster oder größter Wert einer Spalte
	`COUNT`: Anzahl der Werte innerhalb einer Spalte
	`SUM`: Summe aller Werte innerhalb einer Spalte (nur für numerische Wertebereiche)
	`AVG`: arithmetisches Mittel einer Spalte (nur für numerische Wertebereiche)

Tabelle 3.1. Relationenenalgebra-Operationen, nach [HSS18].

Die fünf Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM`, `AVG`, die skalaren Operationen $+, -, \cdot, :$ sowie die Umbenennung von Attributen und Relationen gehören nicht zu den klassischen Relationenalgebra-Operationen, wie sie beispielsweise in [HSS18] definiert werden. Diese zusätzlichen Operationen sind für viele unserer späteren Anfragen jedoch relevant, weshalb wir von nun an immer von der erweiterte Relationenalgebra ausgehen werden. In Abschnitt 10.4 erweitern wir die Liste unserer Operationen zudem um die Gruppierung.

Abhängigkeiten Neben Relationen- und Datenbankschemata sowie den zugehörigen Instanzen benötigt das relationale Datenbankmodell *Integritätsbedingungen*, welche die Beziehungen zwischen den einzelnen Relationen beschreiben. Diese Bedingungen halten den Datenbank-Zustandes konsistent, indem sie beispielsweise dafür sorgen, dass keine unkorrekten Daten in die Datenbank gelangen, z.B. durch Tippfehler, logische Ungereimtheiten und Fehlern bei der Software. Typisches Beispiele für Integritätsbedingungen sind etwa Schlüssel- und Fremdschlüsselbeziehungen, funktionale und mehrwertige Abhängigkeiten sowie Verbundabhängigkeiten und *Inklusionsabhängigkeiten* (IND, siehe Definition 3.9), siehe [Mai83]. Für die in dieser Arbeit untersuchten Fragestellungen sind dabei insbesondere zwei Arten von Abhängigkeiten von Bedeutung. Hierfür werden die klassischen *funktionalen Abhängigkeiten* $A_1 \rightarrow A_2$ (FD, siehe Definition 3.8) und *Verbundabhängigkeiten* $\bowtie [A_1, A_2]$ (JD, siehe Definition 3.10) zu sogenannten *eingebetteten Abhängigkeiten* (siehe Definition 3.12), genauer zu *equality generating dependencies* (egd, siehe Definition 3.14) und *(source-to-target) tuple generating dependencies* ((s-t) tgd, siehe Definition 3.13), verallgemeinert.

Definition 3.8 (funktionale Abhängigkeit (FD), [HSS18])**.** Gegeben seien eine Relation $r(R)$ und zwei Attributmengen $X, Y \subseteq R$. Eine *funktionale Abhängigkeit* (FD) ist ein Ausdruck der Form $X \rightarrow Y$. Dabei *genügt* die Relation r der FD $X \rightarrow Y$ genau dann, wenn

$$| \pi_Y(\sigma_{X=x}(r)) | \leq 1 \text{ für alle } X\text{-Werte } x.$$

■

Definition 3.9 (Inklusionsabhängigkeit (IND), [HSS18])**.** Seien S ein Datenbankschema, $r_1(R_1), r_2(R_2)$ zwei Relationen und $X \subseteq R_1, Y \subseteq R_2$ zwei Attributmengen. Eine *Inklusionsabhängigkeit* ist ein Ausdruck der Form $R_1[X] \subseteq R_2[Y]$. Dabei *genügt* eine Datenbank der IND $R_1[X] \subseteq R_2[Y]$ genau dann, wenn

$$\pi_X(r_1) \subseteq \pi_Y(r_2).$$

■

Definition 3.10 (Verbundabhängigkeit (JD), [HSS18])**.** Sei $S = \{R_1, ..., R_n\}$ ein Datenbankschema über R. Eine *Verbundabhängigkeit* ist ein Ausdruck der Form $\bowtie [R_1, ..., R_n]$. Dabei *genügt* die Relation r der JD $\bowtie [R_1, ..., R_n]$ genau dann, wenn

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

■

Definition 3.11 (Volle Abhängigkeit)**.** Eine *volle Abhängigkeit* (engl. full dependency) ist eine Formel erster Ordnung ohne ∃-Quantor im Head der Form

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

wobei x und y Tupel von Variablen und $\phi(x, y)$ sowie $\psi(x)$ Konjunktionen von atomaren Formeln sind. ■

Definition 3.12 (Eingebettete Abhängigkeit, [GMS12])**.** Eine *eingebettete Abhängigkeit* (engl. embedded dependency) ist eine Formel erster Ordnung der Form

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

wobei x, y und z Tupel von Variablen und ϕ und ψ Konjunktionen von atomaren Formeln sind. Dabei heißt $\phi(x, y)$ *Body* und $\psi(x, z)$ *Head* der Abhängigkeit. Weiter gelte, dass Gleichheitsatome nur im Head der Abhängigkeit auftauchen und diese nicht ∃-quantifiziert sind. ■

(Source-to-target) tuple-generating dependencies und equality-generating dependencies sind Unterklassen der eingebetteten Abhängigkeiten, welche zusätzliche Nebenbedingungen für den Body oder den Head fordern. Dabei gilt: Während s-t tgds als Inter-Datenbankabhängigkeiten angesehen werden können, welche Beschränkungen zwischen zwei Datenbanken darstellen, entsprechen tgds den datenbank-internen Abhängigkeiten. Tgds sind somit ebenso wie Verbundabhängigkeiten ein Spezialfall der s-t tgds (siehe Definition 3.13). Abhängigkeiten, welche Gleichheitsatome im Head erzeugen, können durch egds (siehe Definition 3.14) dargestellt werden. Ihre bekanntesten Vertreter sind funktionale Abhängigkeiten sowie Schlüssel. Fremdschlüsselbedingungen sowie Inklusionsabhängigkeiten wiederum sind spezielle tgds.

Definition 3.13 ((source-to-target) tuple generating dependency ((s-t) tgd))**.** Eine *source-to-target tuple-generating dependency* (s-t tgd) ist eine Sequenz der Form

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

Dabei ist $\phi(x)$ eine Konjunktion von Atomen über S, d.h. Ausdrücken der Form $R(x_1, ..., x_n)$, und $\psi(x, y)$ eine Konjunktion von Atomen über T. Gilt $S = T$, so sprechen wir von einer *tuple-generating dependency*. ■

Definition 3.14 (equality generating dependency (egd))**.** Eine *equality-generating dependency* (egd) ist für zwei Variablen x_1 und x_2 aus x definiert durch

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

Dabei ist $\phi(x)$ Konjunktion von Atomen über S. ■

Neben der *global-and-local-as-view-Abbildung (GLAV)*, wie die s-t tgds auch genannt werden, unterscheiden Fagin et al. in ihren Arbeiten noch zwei weitere Arten von Abhängigkeiten, die *global-as-view dependencies (GAV)* sowie die *local-as-view dependencies (LAV)* [Fag+11b]. Beide sind für die Invertierung von s-t tgds wichtig, werden hier aber nicht weiter vertieft. Für Details zur Invertierung und Komposition von (s-t) tgds sei stattdessen auf [Fag+11b] und [Kas22] verwiesen. In der Regel liefert die Komposition zweier s-t tgds wiederum eine s-t tgd (siehe Tabelle 3.2). In einigen Fällen, wie etwa bei der Komposition einer GLAV mit einer LAV, kann jedoch eine *second-order tgd* (siehe Definition 3.15) entstehen, d.h. eine Abhängigkeit, welche in geeigneter Weise mit ∃-quantifizierten Funktionen und mit Gleichheiten erweitert ist. Weiter gilt: Jede endliche Menge von s-t tgds ist äquivalent zu einer s-o tgd. In ProSA können second-order tgds wie

$$\exists f \ \exists g \ \forall x \ \forall y \ (S(x) \wedge (g(y) = f(x)) \rightarrow T(x, y)),$$

welche insbesondere bei der Invertierung der Auswerungsanfrage oder der Evolution entstehen können, jedoch leicht vermieden werden (siehe Abschnitt 6.2).

$\circ$	GAV	LAV	GLAV
GAV	GAV	GLAV	GLAV
LAV	s-o tgd	s-o tgd	s-o tgd
GLAV	s-o tgd	s-o tgd	s-o tgd

Tabelle 3.2. Komposition verschiedener Arten von Abbildungen [Kas22].

Definition 3.15 (second-order tuple generating dependency (s-o tgd), nach [Fag+05b])**.** Sei $\mathcal{S}$ ein Quell-Schema und $\mathcal{T}$ ein Ziel-Schema. Eine *second-order tuple generating dependency* (s-o tgd) ist eine Formel der Form:

$$\exists \mathcal{F}\ ((\forall x_1(\phi_1 \rightarrow \psi_1)) \wedge ... \wedge (\forall x_n)(\phi_n \rightarrow \psi_n))),$$

wobei gilt:

- Jedes Element $f \in \mathcal{F}$ ist ein Funktionssymbol.
- Jedes ϕ_i ist Konjunktion von
  - atomaren Formeln der Form $S(x_{i_1}, ..., x_{i_k})$, wobei $S \in \mathcal{S}$ ein Relationssymbol der Stelligkeit $\text{stel}(S) = k$ ist und $x_{i_1}, ..., x_{i_k}$ (nicht notwendigerweise verschiedene) Variablen in x_i sind;
  - Gleichheiten der Form $t = t'$, wobei t und t' Terme über x_i und $\mathcal{F}$ sind.
- Jedes ψ_i ist eine Konjunktion von atomaren Formeln $T(t_1, ..., t_l)$, wobei $T \in \mathcal{T}$ ein Relationssymbol der Stelligkeit $\text{stel}(T) = l$ ist und $t_1, ..., t_l$ Terme über x_i und $\mathcal{F}$ sind.
- Jede Variable in x_i erscheint in einer atomaren Formel von ϕ_i. ■

In ProSA interpretieren wir die Auswertungsanfrage Q, die Provenance-Anfrage Q^{-1}_{prov} sowie die Evolution E (siehe Abbildung 2.6) als sogenannte *Schemaabbildungen*, welche anschließend über den Chase (siehe Abschnitt 3.3) verarbeitet werden können.

Schemaabbildungen Eine *Schemaabbildung* beschreibt die Zuordnung eines Quell-Schemas S zu einem Ziel-Schema T (siehe Definition 3.16) — wir wählen S_t statt S und S_{t+1} statt T bei temporalen Datenbanken. Diese beeinflusst auch die zugehörigen Instanzen, welche so transformiert werden, dass die Ziel-Instanz die Abhängigkeitsmenge Σ — in unserem Falle eine Menge von (s-t) tgds und egds — erfüllt.

Definition 3.16 (Schemaabbildung, nach [Fag+11b])**.** Eine *Schemaabbildung* $\mathcal{M}$ ist ein Tripel (S, T, Σ), bestehend aus einem Quell-Schema S, einem Ziel-Schema T sowie einer Menge von Abhängigkeiten Σ, welche die Beziehungen zwischen S und T beschreibt. ■

Die Schema-Evolution wird in der über sogenannte *Schemaevolutionsoperatoren* (siehe Definition 3.17) beschrieben. Um nun die Auswertungsanfrage Q wie auch die Evolution E über den Chase verarbeiten zu können, interpretieren wir diese als Schemaabbildungen nach Definition 3.18 bzw. nach Definition 3.19.

Definition 3.17 (Schemamodifikationsoperator (SMO), nach [CMZ08])**.** Ein *Schemamodifikationsoperator* (engl. Schema Modification Operator, SMO) ist eine Funktion, die als Eingabe ein relationales Schema und die zugrunde liegende Datenbank erhält und als Ausgabe eine (modifizierte) Version des Quell-Schemas und eine migrierte Version der Datenbank erzeugt. ■

Definition 3.18 (Schemaabbildung für eine Auswertungsanfrage)**.** Sei Q eine gegebene Auswertungsanfrage, formalisiert als eine Menge Σ von (s-t) tgds. Seien weiter S und T zwei gegebene Quell- und Ziel-Schemata. Dann beschreibt $\mathcal{M} = (S, T, \Sigma)$ die hierauf definierte Schemaabbildung. ■

Definition 3.19 (Schemaabbildung für eine SMO)**.** Sei E ein gegebener Schemaevolutionsoperator, formalisiert als eine Menge Σ von (s-t) tgds. Seien weiter S_t und S_{t+1} zwei gegebene Schema-Versionen. Dann beschreibt $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ die hierauf definierte Schemaabbildung. ■

Eine gegebene Auswertungsanfrage Q kann also als Menge Σ von (s-t) tgds dargestellt und anschließend als Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ verarbeitet bzw. über dem Quell-Schema S ausgewertet werden. Die für die Invertierung der Anfrage notwendige inverse Schemaabbildung $\mathcal{M}^*$ ist definiert als Tripel (T, S, Σ') über dem Quell-Schema T, dem Ziel-Schema S sowie einer Menge von Abhängigkeiten Σ', welche inverse zu Σ ist. Da Schemaabbildungen oft nicht exakt oder eindeutig invertiert werden können [Fag+08], schreiben wir $^{-1}$ lediglich für exakte und eindeutige inverse Abbildung und * für allgemeine inverse Schemaabbildungen $\mathcal{M}^*$. Wann eine eindeutige Invertierung der Auswertungsanfrage Q oder der Evolution E möglich ist und wie die zugehörigen Inversen aussehen, wird in den Abschnitten 6.3 und 7.3 beschrieben.

3.3. Chase-Algorithmus

Der Chase-Algorithmus, kurz Chase, ist ein in der Datenbanktheorie universell einsetzbares Werkzeug zur Integration von Abhängigkeiten. Er wurde vor mehr als 40 Jahren für die Bestimmung eines optimierten Datenbankentwurfes entwickelt [ABU79; MMS79], hat im Laufe der Jahre aber immer weiter an Bedeutung gewonnen. So können wir, wie in Tabelle 3.3 zu sehen, nicht nur sechs verschiedene Anwendungsszenarien klassifizieren, sondern auch eine Vielzahl an prototypischen Implementierungen festhalten. Hierzu gehören beispielsweise die semantische Anfrageoptimierung, die Beantwortung von Anfragen mit Hilfe von Sichten (engl. Answering Queries using Views (AQuV)) [DH13], der Datenaustausch und -integration (engl. Data Exchange und Data Integration) [Fag+05a], das Data Cleaning [Gee+13] sowie viele Weitere.

Die Anwendung des Chase auf eine Datenbankinstanz I und eine Abhängigkeitsmenge Σ garantiert die Gültigkeit der Regeln aus Σ in I. Wir können also sagen, dass der Chase die Regeln in die Datenbankinstanz „einarbeitet“. Der Chase kann jedoch nicht nur Instanzen verarbeiten. Auch Anfragen können vom Chase verarbeitet werden. Wir unterscheiden daher zwei Fälle:

(a) **Chase auf Instanzen**: Während s-t tgds neue Tupel erzeugen, bereinigen tgds bzw. egds durch Nullwert-Ersetzungen die gegebene Datenbank solange, bis sie alle gegebenen Abhängigkeiten erfüllt [Ben+17; GMS12]. Der Chase auf Instanzen kann u.a. für den Datenaustausch, die Datenintegration, die Anfragenauswertung auf unvollständige Datenbanken oder das Data Cleaning verwendet werden.

(b) **Chase auf Anfragen**: Egds bzw. tgds arbeiten Integritätsbedingungen und s-t tgds Sichten in einer gegebenen Anfrage ein [DH13]. Der Chase auf Anfragen kann u.a. für die semantische Anfrageoptimierung, das Query-Rewriting oder AQuV verwendet werden.

Es gibt bereits erste Ansätze, den Chase auf (beliebige) Objekte und Parameter zu verallgemeinern [Aug+22]. Dies ist möglich durch (1) die vordefinierte Hierarchie von Abhängigkeiten wie JD $\preceq$ tgd $\preceq$ s-t tgd und FD $\preceq$ egd — FDs sind ein Spezialfall von egds — und (2) die Darstellung von Anfragen als Instanzen. Beides wollen wir später in Kapitel 9 noch einmal vertiefen.

Der Chase-Algorithmus Kern des Chase-Algorithmus ist das Finden und Ausführen von Homomorphismen (siehe Definition 3.20). Homomorphismen finden wir bei der Chase-Ausführung mehrfach. Seien hierfür Σ eine Menge von Abhängigkeiten der Form $\forall x \forall y : \phi(x, y) \rightarrow \exists z : \psi(x, z) \in \Sigma$, I eine gegebene Instanz und $I' = \text{Chase}_\Sigma(I)$ das Ergebnis der Chase-Anwendung. Dann sind Homomorphismen von $\phi(x, y)$ nach I, von $\psi(x, z)$ nach I' sowie von I nach I' möglich (siehe Abschnitt 9.1). Die spezielle Anwendung dieser Homomorphismen bezeichnen wir als *Trigger*, definiert in Definition 3.21.

Definition 3.20 (Homomorphismus, nach [Fag+11b])**.** Sei $\underline{\text{Const}}$ eine Konstantenmenge, $\underline{\text{Null}} = \{\mu_i \mid i \in \mathbb{N}\}$ eine Menge von *(markierten) Nullwerten* und $\underline{\text{Var}}$ eine Menge von *(ausgezeichneten* und *nicht-ausgezeichneten) Variablen*. Seien I_1 und I_2 zwei Instanzen über einem Datenbankschema S mit Werten aus $\underline{\text{Const}} \cup \underline{\text{Null}}$. Ein *Homomorphismus* $h : I_1 \rightarrow I_2$ ist eine Abbildung von $\underline{\text{Const}}(K) \cup \underline{\text{Null}}(K)$ nach $\underline{\text{Const}}(J) \cup \underline{\text{Null}}(J)$, sodass

- $\forall c \in \underline{\text{Const}} : h(c) = c$
- $\forall t \in I_1(\mathcal{R}_i(A_1, \ldots, A_k)) : h(t) \in I_2(\mathcal{R}_i(h(A_1, \ldots, A_k)))$

Existiert zudem ein zweiter Homomorphismus $h' : I_2 \to I_1$, heißt I_1 *äquivalent zu* I_2. ■

Definition 3.21 ((aktiver) Trigger, nach [Ben+17])**.** Ein *Trigger* ist ein Homomorphismus zwischen einer Abhängigkeit $\sigma \in \Sigma$ und einer Instanz I. Dabei wird der Rumpf von σ auf ein oder mehrere Tupel $t \in I$ abgebildet. Wir nennen einen Trigger zudem *aktiv*, wenn die Anwendung einer tgd neue Tupel erzeugt und die Anwendung einer egd neue Gleichheiten liefert. ■

Der Chase definiert nun, wann und wie die Trigger zur Verarbeitung der Nebenbedingungen dargestellt als Abhängigkeitsmenge Σ erstellt und angewandt werden. Dabei liefert der Algorithmus eine Folge von Instanzen, welche durch Anwendung von s-t tgds, tgds und egds wie folgt entsteht:

Definition 3.22 (Standard-Chase, nach [Fag+05a; GMS12])**.** Der Chase einer Instanz I über einer Abhängigkeitsmenge Σ ist eine (endliche oder unendliche) Sequenz

$$I_i \xrightarrow{\delta_i, h_i} I_{i+1}, \quad (i = 0, 1, \ldots ; I_0 = I)$$

von Regeln, welche die Abhängigkeiten $\delta_i \in \Sigma$ verarbeiten:

- *tgd-Regel:* Seien I_i eine Instanz, $\delta : \phi(\mathbf{x}) \to \exists \mathbf{y} : \psi(\mathbf{x}, \mathbf{y})$ eine tgd und $h : \phi(x) \to I_i$ ein Homomorphismus. Kann h nicht zu einem Homomorphismus $h' : \phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y}) \to I_i$ erweitert werden, ist δ mittels h anwendbar auf I_i und es gilt $I_{i+1} = I_i \cup h'(\psi(\mathbf{x}, \mathbf{y}))$. Dabei ist jede Variable in $\mathbf{y}$ ein (markierter) Nullwert.
- *egd-Regel:* Gegeben seien eine egd $\delta : \phi(\mathbf{x}) \to (x_1 = x_2)$, eine Instanz I und ein Homomorphismus $h : \phi(\mathbf{x}) \to I$ mit $h(x_1) \neq h(x_2)$. Dann gilt:
  - $h(x_1), h(x_2) \in \underline{\text{Const}}$: Die Anwendung von δ auf I liefert kein Ergebnis; es gilt $I_n = \bot$.
  - $h(x_i) \in \underline{\text{Const}} \wedge h(x_j) \in \underline{\text{Null}}$ mit $i \neq j$ und $i, j \in \{1, 2\}$: Ersetze die (markierten) Nullwerte $h(x_j)$ durch die Konstante $h(x_i)$.
  - $h(x_1)$, $h(x_2) \in \underline{\text{Null}}$: Setze $h(x_1)$ und $h(x_2)$ auf dieselben markierte Nullwerte $h(x_1)$ oder $h(x_2)$.

Die tgd-Regel erzeugt zusätzliche Tupel; sie entspricht somit der Ergänzung der Quell-Datenbank gemäß der tgd durch entsprechende neue Tupel. Die egd-Regel hingegen ersetzt alle (markierten) Nullwerte durch entsprechende Konstanten oder (markierte) Nullwerte mit kleinerem Index. Existiert ein $n \in \mathbb{N}$ mit $I_n \models \Sigma$ oder $I_n = \bot$, so spricht man von einem *erfolgreichen* bzw. einem *scheiternden* Chase. ■

Das Grundkonzept des Chase ist in Algorithmus 1 noch einmal zusammengefasst. Neben dem klassischen Chase auf Tableaus [Mai83] sowie dem Standard-Chase auf Instanzen [Ben+17; GMS12], existieren noch weitere Chase-Varianten wie der *Core* Chase [DNR08], der *Oblivious* Chase [MSL09b; GN08; Mar09] sowie der Chase auf Anfragen [Fag+05a]. Sie werden jeweils in Abschnitt 3.3.1 und 3.3.3 noch einmal kurz vorgestellt. Durch Eliminieren des Trigger-Cheks (rot hervorgehoben) kann der Oblivious Chase leicht aus dem Standard-Chase abgeleitet werden. Für den Core Chase gilt dies jedoch nicht. Bei dieser Variante werden alle Chase-Schritte parallel ausgeführt, was im Allgemeinen nur schwer umzusetzen ist. Wir konzentrieren uns in der vorliegenden Arbeit daher auf den Standard-Chaseund verallgemeinern diesen auf ein allgemeines Chase-Objekt ◯ — statt der Unterscheidung zwischen Instanzen und Anfragen — sowie einen allgemeinen Chase-Parameter $*$ zur Integration verschiedener Nebenbedingungen wie Sichten oder spezielle Abhängigkeiten. Die Verallgemeinerung werden wir insbesondere in Kapitel 9 vorstellen.

Weitere Details zum Standard-Chase können beispielsweise in [Ben+17; GMS12] nachgelesen werden. Die bekanntesten Chase-Anwendungsszenarien, ein Überblick über die verschiedenen Chase-Varianten sowie ein Abriss über das Terminierungskriterium der schwache Azyklizität folgen anschließend in den Abschnitten 3.3.2, 3.3.3 und 3.3.4. Schauen wir uns abschließend noch ein konkretes Beispiel an.

Algorithm 1 Standard-CHASE(Σ, I)

Require: Menge von Abhängigkeiten Σ, Datenbankinstanz I
Ensure: Modifizierte Datenbankinstanz I'

```
for all Trigger h bzgl. einer Abhängigkeit σ ∈ Σ do
  if h ist aktiver Trigger then
    if σ ist tgd then
      Füge neue Tupel der Datenbankinstanz I hinzu
    else if σ ist egd then
      if die verglichenen Werte sind unterschiedliche Konstanten then
        CHASE schlägt fehl
      else
        Ersetze Nullwerte durch andere Nullwerte oder Konstanten
```

Beispiel 3.2. Zur Veranschaulichung des CHASE sei erneut die Universitäts-Datenbank aus Anhang A gegeben. Diese enthält unter anderem eine Relation mit den Namen, Studiengängen und Matrikelnummern ihrer Studierenden. Sei zudem eine weitere Relation `Abschluss` gegeben, welche alle abgeschlossenen Abschlussarbeiten inklusive Titel und Note listet. Mit Hilfe des CHASE wollen wir nun alle Abschlussarbeiten pro Studiengang bestimmen. Seien hierfür folgende Schemata gegeben: `Student`(`studentID`, `lastName`, `firstName`, `studies`), `Thesis`(`title`, `studentID`, `grade`) und `TpS`(`title`, `studentID`, `studies`). Sei weiter

$$\begin{aligned} I' \quad = \quad & \{\texttt{Student}(1, \text{Moore}, \text{Donald}, \text{Teaching}), \texttt{Student}(3, \text{Wood}, \text{Jack}, \text{Engineering}), \\ & \texttt{Student}(7, \text{Smith}, \text{Jack}, \text{Engineering}), \texttt{Thesis}(\text{Answer Set Programming}, 3, 1.3), \\ & \texttt{Thesis}(\text{Objektorientierte Datenbanken}, 1, 1.7)\} \end{aligned}$$

eine gegebene Instanz und Σ mit

$$\begin{aligned} \Sigma \quad = \quad & \{\texttt{Student}(\texttt{studentID}, \texttt{lastName}, \texttt{firstName}, \texttt{studies}) \wedge \texttt{Thesis}(\texttt{title}, \texttt{studentID}, \texttt{grade}) \\ & \rightarrow \texttt{TpS}(\texttt{title}, \texttt{studentID}, \texttt{studies}) \end{aligned}$$

die Abhängigkeitsmenge, welche `TpS` fehlerfrei erzeugt. Dann liefert die Anwendung des CHASE:

$$\begin{aligned} I \quad &= \quad \text{CHASE}_\Sigma(I) \\ &= \quad \{\texttt{TpS}(\text{Answer Set Programming}, 3, \text{Engineering}), \texttt{TpS}(\text{Objektorientierte Datenbanken}, 1, \text{Teaching})\}. \end{aligned}$$

Liegen in `TpS` Duplikate vor, können diese beispielsweise durch Reduktion von Nullwerten erkannt und gestrichen werden. Sei hierzu `TpS`(η_1,3,η_2) ein weiteres Tupel in I' und

$$\begin{aligned} \Sigma' \quad = \quad & \{\texttt{TpS}(\texttt{title}_1, \texttt{studentID}, \texttt{studies}_1) \wedge \texttt{TpS}(\texttt{title}_2, \texttt{studentID}, \texttt{studies}_2) \\ & \rightarrow \texttt{title}_1 = \texttt{title}_2 \wedge \texttt{studies}_1 = \texttt{studies}_2\} \end{aligned}$$

eine weitere Abhängigkeitsmenge. Dann liefert die erneute Anwendung des CHASE:

$$\begin{aligned} I'' \quad &= \quad \text{CHASE}_{\Sigma'}(I'') \\ &= \quad \text{CHASE}_{\Sigma'}(I' \cup \{\texttt{TpS}(\eta_1, 3, \eta_2)\}) \\ &= \quad \{\texttt{TpS}(\text{Answer Set Programming}, 3, \text{Engineering}), \texttt{TpS}(\text{Objektorientierte Datenbanken}, 1, \text{Teaching})\}. \end{aligned}$$

□

3.3.1. Die Entwicklung des Chase

Statt auf Instanzen wie in Algorithmus 1 kann der CHASE auch auf Tableaus sowie auf Anfragen angewendet werden. Die zugrundeliegende Idee bleibt dabei immer gleich und kann in wenigen Worten zusammengefasst werden: Für ein Objekt ◯ und eine Menge von Abhängigkeiten $*$ arbeitet der CHASE den Parameter $*$ in das Objekt ◯ ein, sodass $*$ implizit in ◯ enthalten ist. Bildhaft dargestellt gilt also:

$$\text{CHASE}_*(\bigcirc) = \circledast.$$

Der Parameter ◯ sowie das Objekt ∗ sind dabei beliebig austauschbar. So entspricht ◯ stets einem Tableau $\mathcal{T}$ (siehe unten), einer Datenbankinstanz I oder einer Anfrage Q und ∗ einer Menge von JDs und/oder FD oder einer Menge von (s-t) tgds und/oder egds. In Frage kommen frü ∗ auch Anfragen — wie in ProSA —, Sichten — wie bei AQuV —, allgeime Integritätsbedingungen sowie alles, was als Menge von (s-t) tgds/egds dargestellt werden kann (siehe Abbildung 9.1). Eine Verallgemeinerung des Chase auf ein allgemeines Chase-Objekt sowie einen allgemeinen Chase-Parameter folgt in Kapitel 9. Wir beschränken uns an dieser Stelle auf die in der Literatur etablierten Spezialfälle.

Chase auf Tableaus Sei ein Tableau $\mathcal{T}$ gegeben, dann können für ∗ allgemeine Integritätsbedingungen $\mathcal{B}$ eingesetzt werden. In diesem Fall sei $\mathcal{B}$ eine Menge von FDs und JD. Dann liefert die Anwendung des Chase ein neues äquivalentes Tableau $\mathcal{T}'$ mit

$$\text{Chase}_{\mathcal{B}}(\mathcal{T}) = \mathcal{T}',$$

welches für die Untersuchung der obigen Zusammenhänge zu Rate gezogen werden kann. Details hierzu können in [Mai83] nachgelesen werden.

Seinen Ursprung hat der Chase im Jahre 1979. Maier et al. entwickeln in ihrem Artikel „Testing implications of data dependencies“ [MMS79] einen Algorithmus, welcher auf den Tableaudefinitionen von Aho et al. in [ABU79] basiert. Tableaus ähneln dabei Relationen, die anstelle von Werten zwei disjunkte Mengen von Variablen, *ausgezeichnete Variablen* a_i und *nicht-ausgezeichnete Variablen* b_i, enthalten. Das *Tableauschema*, d.h. die Menge der Attribute, welche die Spalten im Tableau bezeichnen, ähnelt hierbei wiederum der Definition eines Relationenschemas (siehe Abbildung 3.2). Was in einer Relation Tupel wären, wird in diesem Falle als *Zeilen* des Tableaus bezeichnet.

	A_1	A_2	A_3	A_4	
Tableauschema					
Summary			a_3		
Zeilen	b_1	'Max'	a_3	␣	w_1
	b_1	␣	␣	b_2	w_2

Abbildung 3.2. Beispiel für ein Tableau $\mathcal{T}$.

Tableaus haben dabei unterschiedlichste Anwendungsmöglichkeiten. So können sie unter anderem für die Darstellung von Datenbanken durch „Informationsschablonen“, zur Darstellung von Abhängigkeiten in Datenbanken und relationenalgebraische Ausdrücken, zur Optimierung von relationenalgebraischen Ausdrücken oder zur Äquivalenzuntersuchung von Datenbankschemata verwendet werden [Heu20]. Hierzu werden die Tableaus durch den *klassischen* Chase verarbeitet. Dieser ermöglicht die Einarbeitung von funktionalen Abhängigkeiten und Verbundabhängigkeiten durch die sogenannte *F-* und *J-Regel*, ähnlich der egd- sowie der tgd-Regel aus Defintion 3.22. Erstere sichert durch das Ersetzen von nicht-ausgezeichneten Variablen durch andere (nicht-)ausgezeichnete Variablen und Konstanten die Einhaltung der funktionalen Abhängigkeiten. Die J-Regel hingegen erzeugt neue verbundtreue Tupel. Insgesamt ergibt sich dann der in Definition 3.23 definierte *klassische* Chase.

Definition 3.23 (klassischer Chase, nach [Mai83])**.** Für ein gegebenes Tableau $\mathcal{T}$ heißt $\mathcal{T}_n$ Chase *von* $\mathcal{T}$ unter der Einschränkung $\mathcal{B}$, kurz $\mathcal{T}_n = \text{Chase}_{\mathcal{B}}(\mathcal{T})$. Dabei ist $\mathcal{T}_n$ Teil einer Folge von Tableaus $\mathcal{T}_0, \mathcal{T}_1, ...$, wobei sich $\mathcal{T}_{i+1}$ mittels F- oder J-Regel aus $\mathcal{T}_i$ herleiten lässt und $\mathcal{T} = \mathcal{T}_0$ gilt. ■

Der klassische Chase ist im Vergleich zum Standard-Chase noch recht „einfach“ zu händeln. So liefert er bei Eingabe eines gültigen Tableaus auch stets ein gültiges Tableau zurück, terminiert nach endlich vielen Schritten und ist konfluent, d.h. jede mögliche Reihenfolge der Anwendung von F- und J-Regeln liefert dasselbe Ergebnistableau. Es erfüllt somit die *finite Church-Rosser Property* [Mai83, Theorem 8.9]. Dabei erzeugt die F-Regel Gleichheiten und die J-Regel neue Tupel durch die Einarbeitung von funktionalen Abhängigkeiten sowie Verbundabhängigkeiten in das gegebene Tableau [Mai83].

Die intuitive Tableau-Definition von oben kann schnell zu den sogenannten *$(\pi \bowtie \sigma)$-Tableaus* erweitert werden. Diese können dann als relationenalgebraische Ausdrücke interpretiert werden. Dazu bestehen die Einträge des Tableuas aus ausgezeichneten Variablen a_i, nicht-ausgezeichneten Variablen b_i, *Konstanten* c_i und sogenannten *blanks* ␣, einer Art Nullwert. Die *Summary* enthält das Ergebnis der Projektion, welches ausschließlich aus ausgezeichneten Variablen sowie Konstanten besteht. Der natürliche Verbund verschiedener mit w_i, genannt *tags*,

gekennzeichneten Zeilen wird über (nicht-)ausgezeichnete Variablen mit gleichem Index symbolisiert und die Konstanten-Selektion auf Gleichheit (=) durch Konstanten. Alle im Schema, aber nicht in der Projektion vertretenen Attribute werden durch aufsteigend nummerierte nicht-ausgezeichnete Variabeln charakterisiert. Alle nun nicht belegten Positionen werden mit einem blank versehen. Das so entstandene Tableau aus Abbildung 3.2 entspricht nun dem relationenalgebraischen Ausdruck:

$$\pi_{A_3}(\sigma_{A_2='\text{Max}'} w_1(A_1, A_2, A_3) \bowtie w_2(A_1, A_4)).$$

Diese *Tableauanfrage* entspricht wiederum der s-t tgd:

$$w_1(b_1, \text{'Max'}, a_3) \wedge w_2(b_1, b_2) \rightarrow R(a_3).$$

Details zur Definition von $\pi \bowtie \sigma$-Tableaus können unter anderem in [Mai83] nachgelesen werden. Wir konzentrieren uns im Folgenden auf Tableauanfragen in Form von (s-t) tgds. Hierzu entfernen wir uns von den $\pi \bowtie \sigma$-Tableaus und wenden uns stattdessen allgemeinen Datenbankinstanzen zu.

Chase auf Instanzen Seien I eine gegebene Datenbankinstanz und Σ eine Menge von (s-t) tgds und egds. Dann liefert der Standard-Chase aus Algorithmus 1 angewandt auf I und Σ eine äquivalente Instanz I' mit

$$\text{Chase}_\Sigma(I) = I'.$$

Während egds die Instanz I durch Reduzierung von Nullwerten „bereinigen“, erzeugen tgds neue Tupel. Verwenden wir statt einer tgd eine s-t tgd, so werden die neuen Tupel nicht im originalen Schema, sondern über einem zweiten Ziel-Schema erzeugt. Wir erhalten so statt einer Beziehung innerhalb einer Datenbank eine Beziehung zwischen zwei Datenbanken. Der Chase auf Instanzen kann so für Anwendungsszenarien wie Data Exchange [Fag+05a], Data Integration oder Data Cleaning [San14] genutzt werden (siehe Abschnitt 3.3.2).

Chase auf Anfragen Seien zuletzt eine Anfrage Q sowie eine Menge von Integritätsbedingungen $\mathcal{B}$ gegeben. Dann liefert der Chase auf Anfragen eine äquivalente, unter den gegebenen Integritätsbedingungen modifizierte Anfrage Q' mit

$$\text{Chase}_\mathcal{B}(Q) = Q'.$$

Der zugehörige Algorithmus kann beispielsweise in [DPT99; Ren19; Zim20] nachgelesen werden.

Werden neben Integritätsbedingungen in $\mathcal{B}$ auch Sichten $\mathcal{V}$ zugelassen, ergibt sich das *Answering queries using views-Problem* (siehe Abschnitt 3.3.2). Es besteht in der Suche nach effizienten Methoden zur Beantwortung einer Anfrage mittels Sichten [Hal01]. Auch die semantische Optimierung von Anfragen ist mit dieser Art des Chase realisierbar [DPT06]. Der Chase wird hierfür um eine zweite Phase, die sogenannte Backchase-*Phase* erweitert. Die Chase-Phase dient hier der Vorbereitung. Die eigentliche Optimierung bzw. das Query Rewriting findet dann in der Backchase-Phase statt.

3.3.2. Chase-Anwendungsszenarien

Es gibt einige typische Datenbankprobleme, bei denen der Chase als grundlegende Lösungsmöglichkeit verwendet werden kann. Hierzu gehören neben der semantischen Optimierung, dem Answering Queries using Views auch Data Exchange, Data Integration oder Certain Answers (siehe unten). Die Anwendbarkeit des Chase hängt dabei stets von der Wahl des zugehörigen Chase-Parameters sowie des Chase-Objektes ab. Die in Tabelle 3.3 zusammengefassten Varianten wollen wir nun im Folgenden etwas genauer betrachten.

Optimierter Datenbank-Entwurf, 0 Ein guter Datenbankentwurf erfüllt die vier Schema- und Transformationseigenschaften *Abhängigkeitstreue* (T1), *3. Normalform* (S1), *Verbundtreue* (T2) und *Minimalität* (S2). Die Datenbank und ihre Abhängigkeiten können über Tableaus dargestellt und über den Chase verarbeitet bzw. getestet werden. So kann der mit dem Chase insbesondere die Implikation von Abhängigkeiten sowie der Test auf Verbundtreue realisiert werden. Dies waren die ersten Chase-Anwendungen der 80er Jahre [Mai83].

	Parameter ⋆	Objekt ◯	Ergebnis ⊛	Ziel	System
0.	Abhängigkeiten	DB-Schema	DB-Schema mit IBs	optimierter DB-Entwurf	
I.	Abhängigkeiten	Anfrage	Anfrage	semantische Optimierung	PDQ Graal Pegasus
II.	Sichten	Anfrage	Anfrage mit Sichten	AQuV	$\text{Prov}_{\text{C\&B}}$
III.	s-t tgds, egds	Quell-DB	Ziel-DB	Data Exchange, Data Integration	Llunatic, ChaseFun
IV.	tgds, egds	DB	modifizierte DB	Data Cleaning	Llunatic, ChaseFun, ChaseTEQ
V.	tgds, egds	unvollständige DB	Anfrageergebnis	Certain Answers	
VI.	s-t tgds, egds, tgds	DB	Anfrageergebnis	invertierbare Auswertung	ProSA

Tabelle 3.3. Chase-Varianten und ihre Implementierungen [AH19].

Anfrageoptimierung, I Die Anfrageoptimierung sucht für eine gegebene Anfrage Q und eine Menge von Integritätsbedinugnen C (mindestens) eine umgeformte Anfrage Q', welche unter C optimiert wurde. Die Umsetzung über den Chase gelingt hier in zwei Phasen, einer Chase- und einer Backchase-Phase. Deutsch et al. definieren hierfür 2006 einen passenden Algorithmus:

Definition 3.24 (C&B-Algorithmus, nach [DPT06])**.** Der C&B läuft in zwei Phasen ab.

- Chase: Die Eingabe-Anfrage Q wird mit den Integritätsbedingungen C zu einem Universalplan U erweitert. Der Universalplan vereinigt somit alle möglichen Wege, Q unter C zu beantworten.
- Backchase: In dieser Phase werden alle C-minimalen Unteranfragen S_Q von U aufgezählt, welche C-äquivalent zu Q sind, d.h. es gilt: $S_Q \equiv_C Q$. Hierfür werden ein oder mehrere Atome der ursprünglichen Anfrage „fallen gelassen“, unter der Bedingung, dass die Head-variablen im neuen Anfrage-Body weiter erscheinen. ■

Die Backchase-Phase erhält ihren Namen daher, dass sie $S_Q \equiv_C Q$ testet, indem sie sowohl $S_Q \subseteq_C Q$ und $S_Q \supseteq_C Q$ unter Verwendung des Chase-Verfahrens überprüft. Das Optimierungsproblem besteht nun darin, die zu Q äquivalente Anfrage S_Q mit den geringsten Kosten zu finden. Implementiert ist das Verfahren zur semantischen Optimierung unter anderem in PDQ, Graal und Pegasus, welche wir uns in Abschnitt 4.1 genauer anschauen werden.

Der C&B-Algorithmus kann durch eine bottom-up-Suche optimiert werden. Da nur minimale Rewritings von Interesse sind, muss in der Backchase-Phase so nicht der gesamte Suchraum aufgespannt und überprüft werden. Außerdem können alle nicht relevanten Obermengen vernachlässigt werden. Dies Optimierungsvariante nennt man *Pruning* [DPT06].

Answering Queries using Views, II Das *Answering-Queries-using-Views*-Problem (AQuV) beschreibt die Suche nach einer effizienten Methode zur Beantwortung einer Anfrage Q über möglichst viele, vorher definierte Sichten $\mathcal{V}$ [Hal01]. Auch allgemeine Integritätsbedingungen $\mathcal{B}$ sollen in die Anfrage mit eingearbeitet werden. Basierend auf der Arbeit von Ileana haben Deutsch und Hull hierfür der Chase um eine zweite Phase, die sogenannte Backchase-*Phase*, erweitert [Ile14; DH13]. In dieser zweiten Phase werden alle Teilanfragen des Universalplans $U \subseteq \text{Chase}_{\mathcal{V}\cup\mathcal{B}}(Q)$ auf Äquivalenz zur Ausgangsanfrage Q geprüft. Existiert eine zu Q minimale, äquivalente Anfrage, so wird diese gefunden. Insgesamt ergibt sich so:

Definition 3.25 (C&B-Algorithmus, [DH13])**.** Der *C&B*-Algorithmus basiert darauf, eine Sicht-Definition als eine Menge $\mathcal{V}$ von eingebetteten Abhängigkeiten auszudrücken und diese zusammen mit einer Menge von Integritätsbedingungen $\mathcal{B}$ in eine Anfrage Q einzuChaseen. Dafür durchläuft der Algorithmus zwei Phasen:

- Chase-Phase: Chasen wir die Anfrage Q mit der Sichtenmenge $\mathcal{V}$ sowie der Menge an Integritätsbedingungen $\mathcal{B}$, erhalten wir eine erweiterte Anfrage $Q_{\mathcal{V}\cup\mathcal{B}}$. Anschließend bestimmen wir den *Universalplan* als Unteranfrage von $Q_{\mathcal{V}\cup\mathcal{B}}$, durch Einschränkung von $Q_{\mathcal{V}\cup\mathcal{B}}$ auf die Sichtenmenge.
- Backchase-Phase: In dieser Phase werde alle Unteranfragen des Universelplans U auf Äquivalenz zu Q (unter $\mathcal{B}$ und $\mathcal{V}$) geprüft. Anschließend werden alle minimalen, äquivalenten Unteranfragen ausgegeben. ■

Das konkrete *Query Rewriting* findet somit in der BACKCHASE-Phase statt. Der Test auf Äquivalenz von Q und allen Unteranfragen von U kann recht aufwendig sein. Die Autoren haben daher mit dem sogenannten *Pruning* sowie der Einbindung der ***why***-Provenance bereits zwei mögliche Optimierungen vorgestellt. Erstere nutzt die Eigenschaft aus, dass die Oberanfragen einer zu Q minimalen, äquivalenten Anfragen selbst nicht minimal sind. Sie können daher für alle weiteren Überlegungen gestrichen werden. Die ***why***-Provenance merkt sich den Ursprung der in der BACKCHASE-Phase neu erzeugten Prädikate. Diese können so direkt auf ihre Sichtdefinition zurückgeführt werden, ohne dass exponentiell viele äquivalente Unteranfragen von U überprüft werden müssen. Details können in [DH13] nachgelesen werden. Implementiert wurde der C&B-Algorithmus unter dem Namen $\text{Prov}_{\text{C\&B}}$ in der Doktorarbeit von Ileana [Ile14].

Statt auf Sichten können Anfragen auch auf allgemeinen Operatoren beantwortet werden. Diese Verallgemeinerung des AQuV-Problems auf eingeschränkte Anfrage-Operatoren nennen wir auch das *Answering Queries using Operators-Problem*. Hierunter fällt beispielsweise auch die „vertrauenswürdige, adaptive Anfrageverarbeitung in dynamischen Sensornetzwerken zur Unterstützung assistiver Systeme“, wie sie in [Gru22] vorgestellt wird. In diesem an der Universität Rostock entwickelten Projekt verwenden die Autoren Techniken zum Umschreiben von Anfragen und zur Eingrenzung von Anfragen, um eine effiziente und datenschutzgerechte Verarbeitung von Anfragen zu erreichen. Um dies zu erreichen, wird die gesamte Netzwerkstruktur von daten-produzierenden Sensoren bis hin zu Cloud-Computern genutzt, um eine Datenbankmaschine zu schaffen, welche aus Milliarden von Geräten besteht, dem *Internet of Things*. Als Ressourcen nutzen wir die auf den jeweiligen Geräten zur Verfügung stehenden Anfrageoperatoren. Hierfür wird die Anfrage in Fragmente und Restanfragen aufteilt. Die Anfragefragmente werden zunächst auf ressourcenbeschränkten Geräten operiert, gefiltert und voraggregiert und anschließend von den Restanfragen auf leistungsfähigeren Geräten wie Cloud-Servern weiterverarbeitet. So werden insgesamt weniger Daten verarbeitet und an einen Cloud-Server weitergeleitet, was den Datenschutz sowie die Datenminimierung befördert [GH17; GH18].

Data Exchange und Data Integration, III Beim *Data Exchange* werden Daten eines Quellschemas (*source*) in die Instanz eines anderen Zielschemas (*target*) überführt. Hierfür verarbeitet der CHASE auf Instanzen s-t tgds, sodass eine Inter-Datenbankbeziehung entsteht. Fagin et al. definieren das Problem wie folgt:

Definition 3.26 (Data Exchange, nach [Fag+05a])**.** Ein Data Exchange-Setting ist ein Tupel $(S, T, \Sigma_{st}, \Sigma_t)$ mit S als Quell-Schema, T als Ziel-Schema ($S \cap T = \emptyset$), Σ_{st} als Menge von s-t tgds, die den Zusammenhang zwischen Quelle und Ziel beschreiben, sowie Σ_t als Menge von Abhängigkeiten über dem Ziel-Schema T. Sei zudem I eine Instanz über S. Das Data Exchange-Problem besteht nun darin, eine neue Instanz J über T so zu materialisieren, dass J die Abhängigkeiten in Σ_t erfüllt und die s-t tgds in Σ_{st} sowohl von I als auch von J erfüllt werden. Eine solche Instanz J ist eine Lösung des Problems. ■

Neben *Llunatic* [Gee+20] ist auch *ChaseFun* [BIL17] für diesen Anwendungsfall geeignet. Details sind in Abschnitt 4.1 zusammengefasst. Beide Systeme sind zudem in der Lage das *Data Integration-Problem* zu bearbeiten. Dabei geht es darum, Daten aus verschiedenen Quellen zu kombinieren und dem Benutzer eine einheitliche Sicht auf diese Daten zur Verfügung zu stellen [Hal01; Hul97; Ull97]. Ein *Data Integration-System* ist dabei wie folgt definitiert:

Definition 3.27 (Data Integration-System, [GMS12])**.** Ein *Data Integration-System* ist ein 4-Tupel der Form $I = (S, G, \Sigma_{SG}, \Sigma_G)$ mit S Quell-Schema und G globalem Schema. Weiter ist Σ_G eine Menge von Integritätsbedingungen über G und $\Sigma_S G$ eine Abbildung von S nach G. ■

Data Cleaning, IV Beim *Data Cleaning* oder *Data Repairing* wird durch die Bestimmung und Reparatur unvollständiger und/oder inkonsistenter Datensätze die Datenqualität verbessert. Hierfür werden Inkonsistenzen mit Hilfe des CHASE auf Instanzen durch Anwendung von egds und tgds behoben. Liegen Duplikate vor, werden diese durch Anwendung der egds erkannt und zusammengelegt. Da wir hier mit relationenalgebraischen Ausdrücken arbeiten, wird dies bei uns jedoch nicht auftreten. Bei einer Datenfusion werden Nullwerte mit Hilfe von egds durch andere Nullwerte oder Konstanten ersetzt. Dies kann im besten Falle eine Tupel-Reduktion nach sich ziehen. Fehlen Tupel, so werden diese mittels tgds ergänzt.

Auch die Säuberung inkonsistenter oder widersprüchlicher Datensätze ist Teil des Data Cleaning. Dies ist ohne spezielle Regeln wie beispielsweise Abstandsfunktionen oder die Einbindung eines Nutzers vom System selbst allerdings nicht zu lösen. Llunatic, ein Chase-System mit dem Schwerpunkt Data Cleaning, führt hierfür spezielle Chase-Bäume und Variablen, sogenannte Lluns ein, mit Hilfe derer der Nutzer Inkonsistenzen selbst bereinigen kann [Gee+20]. Eine ausführliche Beschreibung des *Data Cleaning-Prozesses* (siehe Definition 3.28) ist zudem unter [San14] zu finden. Weitere Chase-Systeme, welche sich mit diesem Anwendungsszenario beschäftigen, sind beispielsweise *ChaseFun* [BIL17] und *ChaseTEQ* [Spe11]. Detail können in Abschnitt 4.1 nachgelesen werden.

Definition 3.28 (Data Cleaning, nach [San14])**.** Ein *cleaning egd* über zwei Schemata S und T ist eine Formel der Form $\forall x(\phi(x) \rightarrow t_1 = t_2)$ mit $\phi(x)$ Konjunktion von relationalen Atomen über $\langle S, T\rangle$ und $t_1 = t_2$ von der Form $x_i = c$ oder $x_i = x_j$ für einige Variablen x_i, $x_j \in x$ und eine Konstante $c \in \underline{\text{Const}}$. Außerdem ist höchstens eine Variable im Head der egd Teil eines Beziehungsatoms über S.

Eine *erweiterte tgd* über zwei Schemata S und T ist eine Formel der Form $\forall x(\phi(x) \rightarrow \exists y : \psi(x, y)$, wobei $\phi(x)$ eine Konjunktion von relationalen Atomen über $\langle S, T\rangle$ und $\psi(x, y)$ eine Konjunktion von relationalen Atomen über T ist.

Die partielle Funktion $User$ nimmt als Eingabe eine beliebige Menge von Zellen C und liefert entweder einen Wert $v \in \underline{\text{Const}}$, welchen die Zellen in C annehmen sollen, oder $\perp$, um anzuzeigen, dass die Änderung der Zellen in C eine inkorrekte Änderung der Datenbank darstellen würde und daher nicht durchgeführt werden sollte.

Sei $D = \underline{\text{Const}} \cup \underline{\text{Null}} \cup \text{lluns}$ eine gegebene Domäne. Dann ist ein *Mapping & Cleaning Szenario über D* ein Tupel $MC = (S, S_a, T, \Sigma_t, \Sigma_e, \sqcap, User)$ mit:

1. Quellschema $S \cup S_a$, Zielschema T und der Menge der maßgeblichen Quell-Relationen S_a ;
2. einer Menge von erweiterten tgds Σ_t;
3. einer Menge von cleaning egds Σ_e;
4. einer partiell-geordnete Spezifikation $\sqcap$ für die Attribute von S und T;
5. einer partiellen Funktion $User$.

Sind die Mengen der tgds Σ_t leer, nennen wir MC auch *Reinigungsszenario.* ■

Certain Answers, V Unter der *Certain Answer* einer Anfrage Q verstehen wir eine Menge von Tupeln t, die in Bezug auf eine unvollstänige Datenbankinstanz I in jeder möglichen Werlt von I eine Antwort auf Q liefert. Wie in [GMS12] beschrieben, entspricht $Q(I) = \{Q(D) \mid D \in I\}$ der Menge an Anfrageergebnissen für eine gegebene Anfrage Q sowie eine Datenbankinstanz I. Dennoch gibt es Tupel, die keine Antwort auf Q sind. Andererseits existieren auch Tupel, die Antwort auf Q bzgl. einiger, aber nicht notwendiger aller mögliche Werte sind. Wir unterscheiden daher immer „sichere“ und „mögliche“ Antworten:

Definition 3.29 (Certain Anwers, [GMS12])**.** Die Menge an *Certain Answers* einer Anfrage Q bzgl. einer unvollständigen Datenbank I ist definiert über:

$$\text{certain}(Q, I) = \bigcap_{D \in I} Q(D).$$

Die Menge an *possible Answers* einer Anfrage Q bzgl. einer unvollständigen Datenbank I ist definiert über:

$$\text{possible}(Q, I) = \bigcup_{D \in I} Q(D).$$

■

Eine Implementierung des Problems mit Hilfe des Chase ist uns nicht bekannt.

Invertierbare Auswertungen, VI Die Invertierung einer Auswertungsanfrage zur Rekonstruktion der Quell-Daten ist Hauptaufgabe des hier vorgestellten Promotionsprojekts (ProSA). Ziel ist die Minimierung von Forschungsdaten unter anderem aufgrund von Privacy-Aspekten und Kapazitätsgründen. Hierfür kombinieren wir den Chase mit zusätzlichen Provenance-Informationen wie Zeugenbasen und weiteren Annotationen, um die (minimale) Teil-Datenbank eines ursprünglichen Forschungsdatensatzes zu bestimmen, auf deren Grundlage wir ein gegebenes Anfrageergebnis reproduzieren bzw. replizieren können. Die Repräsentation der Auswertung durch den Chase bietet zusätzlich den Vorteil, dass sich weitere Phasen des Forschungsdatenmanagements auf dieselbe Art und Weise darstellen lassen und somit nahtlos in die Technik integriert werden können: zu diesen Phasen gehören einfache Datentransformationen, die Datenbereinigungen, die Schema- (und Daten-)Evolution sowie der Datenaustausch zwischen verschiedenen Forschungsdatenbanken. Erste Ergebnisse hierzu sind unter anderen in den Kapiteln 5 bis 10 zu finden.

3.3.3. Chase-Varianten

Neben dem oben beschriebenen Standard-Chase existieren noch weitere Chase-Varianten wie der *Oblivious* Chase oder der *Core* Chase. Diese Chase-Varianten unterscheiden sich leicht in ihrem Ablauf, basieren aber alle auf dem Chase für Tableaus aus dem Jahre 1979. Jede Chase-Variante hat dabei unterschiedliche Eigenschaften in der Logik ihrer zugrundeliegenden Abhängigkeiten, der Terminierung, der Konfluenz sowie ihren Anwendungsmöglichkeiten.

Standard-Chase Unter dem Standard-Chase verstehen wir den in Algorithmus 1 eingeführten Chase-Algorithmus. Dieser wird seit 2005 unter anderem von R. Fagin [Fag+05b] als häufigste Chase-Interpretation verwendet. Weitere Informationen über diese Chase-Variante sind beispielsweise in [GMS12; Ben+17] sowie in der originalen Definition [BV84] zu finden.

Oblivious Chase Der Oblivious Chase vernachlässigt die Prüfung auf aktive Trigger (siehe Algorithmus 1, rot hervorgehoben). Somit ist seine Anwendung um einiges performanter als der Standard-Chase, sofern der Chase trotz fehlendem Trigger-Test terminiert. Die Nicht-Terminierung ist beim Oblivious Chase jedoch wahrscheinlicher als beim Standard-Chase, da egds und tgds auch dann angewandt werden, wenn der Head einer Integritätsbedingung bereits erfüllt ist. Dies kann im Falle von tgds durch die ständige Generierung neuer Tupel zu einer unendlichen Chase-Sequenz führen. In der Literatur werden dabei zwei Varianten des Oblivious Chase unterschieden: der *Oblivious Naive* Chase [CGK08; MSL09b; Cat+09] und der *Oblivious Skolem* Chase [Mar09]. Letzterer kann mit Hilfe einer *Skolemfunktion* die Zahl der neu eingeführten Nullwerte reduzieren, da neue Nullwerte auf bereits existierende Nullwerte abgebildet werden können. Dabei fallen die Nullwerte in sich zusammen, welche von denselben allquantifizierten Variablen abhängen.

Beispiel 3.3. Gegeben seien eine Instanz $I = \{\texttt{Student}(3, \text{Müller}, \text{Max}, \text{Informatik})\}$ sowie eine tgd[1]

$$\sigma : \texttt{Student}(s_{id}, l, f, c) \rightarrow \exists L, F : \texttt{Student}(s_{id}, L, F, c).$$

Seien hierbei s, l, f und c Abkürzungen für die Attribute `studentID`, `lastName`, `firstName` sowie `studies`. Dann liefert die Anwendung des Standard-Chase:

$$\text{Chase}_\Sigma(I) = \{\texttt{Student}(3, \text{Müller}, \text{Max}, \text{Informatik})\}.$$

Für den Oblivious Skolem Chase leiten wir zunächst aus σ die skolemisierte tgd

$$\sigma' : \texttt{Student}(s_{id}, l, f, c) \rightarrow \exists L, F : \texttt{Student}(s_{id}, f_L^\sigma(s_{id}, m_{id}, c), f_F^\sigma(s_{id}, m_{id}, c), c)$$

ab und erhalten dann nach Anwendung des Chase:

$$\begin{aligned} \text{Chase}_\Sigma(I) \quad = \quad & \{\texttt{Student}(3, \text{Müller}, \text{Max}, \text{Informatik}), \\ & \texttt{Student}(3, f_L^\sigma(3, \text{Informatik}), f_F^\sigma(3, \text{Informatik}), \text{Informatik})\}. \end{aligned}$$

[1] $\exists$-quantifizierte Variabeln werden zur besseren Unterscheidung groß geschrieben.

Für den Oblivious Naive Chase leiten wir aus σ die skolemisierte tgd

$$\sigma'' : \texttt{Student}(s_{id}, l, f, c) \rightarrow \exists L, F : \texttt{Student}(s_{id}, f_L^\sigma(s_{id}, l, f, c), f_F^\sigma(s_{id}, l, f, c), c)$$

ab. Diese erhält im Gegensatz zu σ' alle im Rumpf auftretenden Variablen als Eingabe für f. Ein durch σ'' eingeführter Skolem-Term wird dabei sofort durch einen neuen Nullwert ersetzt. Dies erzeugt im Laufe der Chase-Anwendung stetig neue Tupel, denn:

$$\begin{aligned}
\text{Chase}_\Sigma(I) &= \text{Chase}_\Sigma(\{\texttt{Student}(3, \text{Müller}, \text{Max}, \text{Informatik})\}) \\
&= \{\texttt{Student}(3, \text{Müller}, \text{Max}, \text{Informatik}), \\
&\quad \underbrace{\texttt{Student}(3, f_L^\sigma(3, \text{Müller}, \text{Max}, \text{Informatik}), f_F^\sigma(3, \text{Müller}, \text{Max}, \text{Informatik}), \text{Informatik})}_{= \texttt{Student}(3, \eta_1, \eta_2, \text{Informatik})}, \\
&\quad \underbrace{\texttt{Student}(3, f_L^\sigma(3, \eta_1, \eta_2, \text{Informatik}), f_F^\sigma(3, \eta_1, \eta_2, \text{Informatik}), \text{Informatik})}_{= \texttt{Student}(3, \eta_3, \eta_4, \text{Informatik})}, \ \ldots \}
\end{aligned}$$

In beiden Fällen werden alle gefundenen Trigger wie aktive Trigger behandelt. Eine tgd erzeugt somit bei jeder Anwendung ein neues Null-Tupel. Durch die Skolemfunktionen $f_L^\sigma(s_{id}, m_{id}, c)$ und $f_F^\sigma(s_{id}, m_{id}, c)$ endet die Chase-Anwendung im Falle des Oblivious Skolem Chase bereits nach einem Chase-Schritt. Eine Terminierung des Oblivious Naive Chase hingegen ist ausgeschlossen, da die entsprechenden Skolemfunktionen $f_L^\sigma(s_{id}, m_{id}, l, f, c)$ und $f_F^\sigma(s_{id}, m_{id}, l, f, c)$ stetig neue Tupel erzeugen. □

Der Oblivious- sowie der Standard-Chase haben das Problem, dass die Reihenfolge der eingeChaseten Abhängigkeiten nicht-deterministisch ist. Somit ist die Terminierung des Algorithmus für ein gegebenes Chase-Objekt und eine Menge von Integritätsbedingungen (tgds und egds) unentscheidbar — bei der Einschränkung der Integritätsbedingungen auf FDs und JDs bleibt die Terminierung entscheidbar [MMS79]. Aus diesem Grund sind Bedingungen unter denen der Chase terminiert immer noch Stand der Forschung. Das kleine Auswahl an Terminierungskriterien werden wir in Abschnitt 3.3.4 vorstellen.

Core Chase Der *Core* Chase bestimmt durch seine parallele Abarbeitung (sofern existent) eine universelle Lösung. Dafür werden zunächst im sogenannten *parallelen* Chase-*Schritt* alle aktiven Trigger parallel angewandt und anschließend auf ihren Kern reduziert. Dabei entfallen alle Lösungen, die von einer anderen Lösung überdeckt werden. Da der Kern einer Instanz bis auf Isomorphie eindeutig ist, gilt der Core Chase als deterministisch. Aufgrund der hohen Komplexität der parallelen Chase-Schritte findet diese Chase-Variante in der Praxis jedoch kaum Anwendung.

Beispiel 3.4 (basierend auf [GMS12])**.** Betrachten wir erneut die Relation `Student` der Universitäts-Datenbank und schränken diese auf die Attribute `firstName` und `lastName` ein. Die resultierende Relation nennen wir `Name`. Gegeben seien zudem die leere Instanz I sowie eine Menge von Abhängigkeiten Σ mit

$$\begin{aligned}
\Sigma &= \{\emptyset \rightarrow \exists \texttt{FirstName}, \texttt{LastName} : \texttt{Name}(\texttt{FirstName}, \texttt{LastName}) \wedge \texttt{Name}(\texttt{LastName}, \texttt{FirstName}), \\
&\quad \texttt{Name}(\texttt{firstName}, \texttt{lastName}) \wedge \texttt{Name}(\texttt{lastName}, \texttt{firstName}) \\
&\qquad \rightarrow \exists \texttt{MiddleName} : \texttt{Name}(\texttt{MiddleName}, \texttt{MiddleName}), \\
&\quad \texttt{Name}(\texttt{firstName}, \texttt{lastName}) \\
&\qquad \rightarrow \exists \texttt{MiddleName} : \texttt{Name}(\texttt{firstName}, \texttt{MiddleName}) \wedge \texttt{Name}(\texttt{MiddleName}, \texttt{lastName})\}
\end{aligned}$$

Dann liefert die Anwendung des Core Chase:

$$\begin{aligned}
\text{Chase}_\Sigma(I) &= \text{Chase}_\Sigma(\emptyset) \\
&= \text{core}(\{\texttt{Name}(\eta_1, \eta_2), \texttt{Name}(\eta_2, \eta_1), \texttt{Name}(\eta_3, \eta_3), \texttt{Name}(\eta_1, \eta_4), \texttt{Name}(\eta_4, \eta_1), \\
&\qquad \texttt{Name}(\eta_2, \eta_5), \texttt{Name}(\eta_5, \eta_2)\}) \\
&= \{\texttt{Name}(\eta_3, \eta_3)\}.
\end{aligned}$$

Durch die parallele Abarbeitung der drei Abhängigkeiten entstehen zunächst sieben Null-Tupel, welche anschließend auf einen gemeinsamen Kern $\texttt{Name}(\eta_3, \eta_3)$ reduziert werden. □

Vergleich der Chase-Varianten Fassen wir unsere Ergebnisse noch einmal zusammen: Der in Definition 3.22 vorgestellte Chase wird auch als Standard-Chase bezeichnet. Er entspricht der seit 2005 gängigen Chase-Interpretation. Vernachlässigt wir den „aufwendigen“ Check auf aktive Trigger, so erhalten wir den Oblivious Chase. Führen wir hingegen alle Chase-Schritte parallel aus, nennen wir diese Variante den Core Chase.

Jede der drei Chase-Varianten hat ihre theoretische Berechtigung und praktische Anwendungen. Eine Implementierung des Core Chase ist uns aufgrund der parallelen Ausführung der Chase-Schritte nicht bekannt. Für die anderen beiden Chase-Varianten finden sich in Abschnitt 4.1 jedoch verschiedene Beispiele.

Den Zusammenhang zwischen den verschiedenen Chase-Varianten beschreiben die Autoren in [GMS12] zudem wie folgt: Eine Menge von Datenabhängigkeiten ist in $\mathrm{TOC}_\forall[C]$ für $C \in \{\mathrm{SC}, \mathrm{NOC}, \mathrm{SOC}, \mathrm{CC}\}$, wenn für jede Datenbankinstanz D **alle** Chase-Sequenzen von D endlich sind. Eine Menge von Datenabhängigkeiten ist in $\mathrm{TOC}_\exists[C]$ für $C \in \{\mathrm{SC}, \mathrm{NOC}, \mathrm{SOC}, \mathrm{CC}\}$, wenn für jede Datenbankinstanz D **mindestens eine** Chase-Folge von D endlich ist. Terminiert der Chase für alle Sequenzen, so terminiert er auch für mindestens eine Sequenz, d.h. $\mathrm{TOC}_\forall[C] \subsetneq \mathrm{TOC}_\exists[C]$. Außerdem gilt $\mathrm{TOC}_\forall[C] \equiv \mathrm{TOC}_\exists[C]$, da im Falle des core Chase alle Chase-Schritte parallel ausgeführt werden, während die durch den Oblivious Chase generierten Ergebnisse eindeutig sind. Die Ergebnisse lassen sich in folgendem Satz zusammenfassen:

Theorem 3.1 (Chase-Vergleich, [DNR08; One12])**.** *Es gilt:*

1. $TOC_\forall(SC) \subsetneq TOC_\exists(SC) \subsetneq TOC_\forall(CC) \equiv TOC_\exists(CC)$
2. $TOC_\forall(NOC) \equiv TOC_\exists(NOC) \subsetneq TOC_\forall(SOC) \equiv TOC_\exists(SOC) \subsetneq TOC_\forall(SC)$

□

3.3.4. Terminierungskriterien und Konfluenz

Die klassische Chase-Definition auf Tableaus (siehe Definition 3.23) terminiert und ist konfluent. Für seine Erweiterung auf Instanzen oder Anfragen gilt dies in der Regel nicht. Grund hierfür ist jedoch nicht das Chase-Objekt, sondern der verwendete Chase-Parameter. Im klassischen Tableau-Fall verarbeitet der Chase eine Menge von funktionalen Abhängigkeiten (FDs) sowie Verbundabhängigkeiten (JDs). Da es sich hierbei um volle Abhängigkeiten handelt, d.h. Abhängigkeiten ohne $\exists$-Quantor im Head (siehe Definition 3.11), ist der Chase terminierend und konfluent. Arbeitet der Chase hingegen auf eingebetteten Abhängigkeiten (siehe Definition 3.12) wie (s-t) tgds und/oder egds, können Terminierung und Konfluenz nicht garantiert werden.

Das Einfügen von Tupeln mit neuen Variablen oder (Null-)werten kann dazu führen, dass die Ausführung des Chase nicht mehr stoppt. In diesem Fall „befeuern“ sich zwei tgds gegenseitig, sodass bei jeder Anwendung neue Variablen bzw. Nullwerte erzeugt werden. Zudem ist die Reihenfolge, in welcher die Abhängigkeiten angewandt werden entscheidend. Beides wird im folgenden Beispiel verdeutlicht:

Beispiel 3.5. Gegeben seien zwei Relationen $S(n)$ und $R(n, m)$, welche die Namen aller Studierenden sowie eine Liste der studentischen Vertreter in den universitären Gremien beschreibt. Hierbei wird jeder Studierende n durch einen studentischen Sprecher m vertreten. Sei weiter Σ_1 eine Menge von Abhängigkeiten

$$\Sigma_1 = \{\underbrace{S(n) \to \exists M : R(n, M)}_{\sigma_1}, \underbrace{R(n, m) \to S(m)}_{\sigma_2}\}.$$

Sie besagt, dass jeder Studierende einen studentischen Sprecher hat und jeder studentische Sprecher selbst ein Studierender ist. Die Anwendung des Chase liefert für $I_0 = \{S(\mathrm{Max})\}$:

$$\begin{aligned}
\mathrm{Chase}_\Sigma(I_0) &\overset{\sigma_1}{=} \{S(\mathrm{Max}), R(\mathrm{Max}, \eta_1)\} \\
\mathrm{Chase}_\Sigma(I_1) &\overset{\sigma_2}{=} \{S(\mathrm{Max}), R(\mathrm{Max}, \eta_1), S(\eta_1)\} \\
\mathrm{Chase}_\Sigma(I_2) &\overset{\sigma_1}{=} \{S(\mathrm{Max}), R(\mathrm{Max}, \eta_1), S(\eta_1), R(\eta_1, \eta_2)\} \\
\mathrm{Chase}_\Sigma(I_3) &= \{S(\mathrm{Max}), R(\mathrm{Max}, \eta_1), S(\eta_1), R(\eta_1, \eta_2), ...\}
\end{aligned}$$

Der Standard-Chase nach Definition 3.22 terminiert somit nicht.

Sei alternativ Σ_2 mit

$$\Sigma_2 = \{\underbrace{S(n) \to \exists M : R(n, M)}_{\sigma_1}, \underbrace{S(n) \to R(n, n)}_{\sigma_3}\}$$

sowie $I_0 = \{S(\text{Max})\}$ wie zuvor gegeben. Dann liefert die Anwendung des Chase entweder $\{S(\text{Max}, R(\text{Max}), \text{Max}))\}$ oder $\{S(\text{Max}), R(\text{Max}, \eta_1), R(\text{Max}, \text{Max})\}$. Denn:

$$\begin{aligned} \text{Chase}_\Sigma(I_0) &\overset{\sigma_1}{=} \{S(\text{Max}), R(\text{Max}, \eta_1)\} \\ \text{Chase}_\Sigma(I_1) &\overset{\sigma_3}{=} \{S(\text{Max}), R(\text{Max}, \eta_1), R(\text{Max}, \text{Max})\} \end{aligned}$$

$$\text{Chase}_\Sigma(I_0) \overset{\sigma_3}{=} \{S(\text{Max}), R(\text{Max}, \text{Max})\}$$

Die Reihenfolge, in welcher die Abhängigkeiten σ_1 und σ_3 angewendet werden, entscheidet somit über das Ergebnis der Chase-Anwendung. Der Standard-Chase ist somit nicht konfluent. Grund für die Nicht-Terminierung sowie die Nicht-Konfluenz ist in jeweils der $\exists$-Quantor im Head der tgd σ_1. □

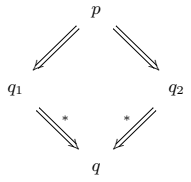
Konfluenz Kann ein Objekt q auf verschiedene Weisen aus einem anderen Objekt p abgeleitet werden, nennen wir das zugehörige System *konfluent*. In unserem Kontext wird diese durch das Church-Rosser-Theorem (siehe Theorem 3.2 basierend auf Definition 3.31) gesichert. Im Original basiert dieses auf der Arbeit [CR36] von Church und Rosser. Sei hierfür der Chase ein Ersetzungssystem (siehe Definition 3.30) für eine Menge von vollen Abhängigkeiten $\mathcal{C}$. Dann ist Q eine Menge von Tableaus über einem Universum $\mathcal{U}$ und $\mathcal{T} \Rightarrow \mathcal{T}'$ eine Folgerung, welche einer Abhängigkeit in $\mathcal{C}$ entspricht, entstanden durch Anwendung der F-Regel oder der J-Regel (siehe Abschnitt 3.3.1 oder [Mai83]).

Definition 3.30 (Ersetzungsystem, nach [Mai83])**.** Ein *Ersetzungssystem* ist ein Paar $(Q, \Rightarrow)$, wobei Q eine Menge von Objekten und $\Rightarrow$ eine anti-reflexive binäre Relation auf Q ist. ■

Definition 3.31 (Church-Rosser-System, nach [Mai83])**.** Für ein Ersetzungssystem $(Q, \Rightarrow)$ heißt ein Objekt $p \in Q$ *irreduzibel*, wenn $p \overset{*}{\Rightarrow} q$ auch $p = q$ impliziert. Weiter heißt das Ersetzungssystem $(Q, \Rightarrow)$ *endlich*, wenn es für jedes $p \in Q$ eine von p abhängige Konstante c gibt, so dass aus $p \overset{*}{\Rightarrow} q$ in $i \leq c$ Schritten möglich ist.

Ein endliches Ersetzungssystem $(Q, \Rightarrow)$ ist ein *endliches Church-Rosser-System* (FCR), wenn für ein beliebiges Objekt $p \in Q$ mit $p \overset{*}{\Rightarrow} q_1$ und $p \overset{*}{\Rightarrow} q_2$ sowie q_1 und q_2 irreduzibel gilt, $q_1 = q_2$. Das heißt, ausgehend von einem beliebigen p landen wir egal wie wir die Transformation anwenden bei demselben irreduziblen Objekt. ■

Theorem 3.2 (Sethi, nach [Mai83])**.** *Ein Ersetzungssystem $(Q, \Rightarrow)$ ist FCR, genau dann, wenn es endlich ist und für jedes Objekt $p \in Q$ mit $p \Rightarrow q_1$ und $p \Rightarrow q_2$ ein $q \in Q$ existiert, so dass $q_1 \overset{*}{\Rightarrow} q$ und $q_2 \overset{*}{\Rightarrow} q$ gilt. Schematisch:*



□

Hieraus folgern die Autoren in [Mai83, Thoerem 8.9], dass die Chase-Berechnung für eine Menge von vollen Abhängigkeiten, d.h. für eine Menge von FDs und JDs, ein FCR-Ersetzungssystem ist und somit stets nur eine gültige Lösung liefert. Der Chase auf Tableaus ist damit konfluent. Ein entsprechender Beweis ist beispielsweise in [Set74; Gra80] oder [BÓ81] zu finden. Für den Standard-Chase mit eingebetteten Abhängigkeiten, wie wir ihn im folgenden auf Instanzen oder Anfragen verwenden werden, kann die Konfluenz in der Regel jedoch nicht garantiert werden.

Terminierung Nach Theorem 3.3 terminiert der Chase auf Tableaus, welcher FDs und JDs verarbeitet, immer. Für den Chase mit eingebetteten Abhängigkeiten, d.h. (s-t) tgds und egds, hingegen definieren wir verschiedene Klassen von Abhängigkeitsmengen, für welche der Chase sicher terminiert. Zu den statischen Kriterien gehören unter anderem die *schwache Azyklizität* (WA), *Safety* (SC), die *super-schwache Azyklizität* (SwA), die *Stratifizierung* (Str) sowie verschiedene Rewriting-Techniken wie *Adn* oder Adn^+. Alle Kriterien sind in [GMS12; GST11] sowie [MSL09a] überblicksmäßig zusammengefasst.

Theorem 3.3 (Chase-Terminierung, nach [Mai83])**.** *Sei $\mathcal{T}$ ein Tableau und $\mathcal{C}$ eine Menge von vollen Abhängigkeiten. Dann ist jede erzeugende Sequenz von $\mathcal{T}$ unter $\mathcal{C}$ endlich, d.h.* $\text{Chase}_{\mathcal{C}}(\mathcal{T})$ *terminiert.*

Der klassische Chase mit FDs und JDs auf Tableaus ist somit konfluent und terminiert. FDs und JDs sind volle Abhängigkeiten ohne ∃-Quantor im Head (siehe Definition 3.11). Wir benötigen zur Anwendung des Chase auf Auswertungsanfragen (siehe Abschnitt 6.4), für das Provenance Management (siehe Abschnitt 10.5) sowie die Anwendung auf Schema-Evolutionen (siehe Abschnitt 7.1) als Chase-Parameter jedoch allgemeine (s-t) tgds und egds. Der Chase auf diesen eingebetteten Abhängigkeiten (siehe Definition 3.12) terminiert im allgemeinen nicht und ist zudem auch nicht konfluent. Daher benötigen wir zugehörige Kriterien zur Garantie von Terminierung und Konfluenz.

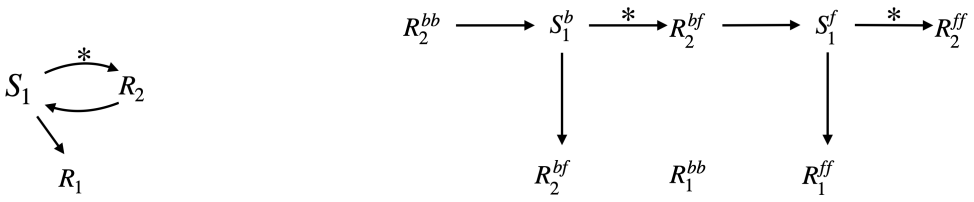
Wir konzentrieren uns hier auf Terminierungskritierien und darunter auf das einfachste und bekannteste Kriterium, die schwache Azyklizität (siehe Definition 3.32). Sie besagt, dass die Struktur bestimmter eingebetteter Abhängigkeiten — problematisch sind insbesondere (s-t) tgds mit ∃-Quantor im Head — die Terminierung des Chase garantiert. Das zugehörige Kriterium basiert auf den strukturellen Eigenschaften des Abhängigkeitsgraphen $dep(\Sigma)$, der aus den gegebenen tgds abgeleitet werden kann.

Definition 3.32 (Schwache Azyklizität, nach [GMS12])**.** Sei Σ eine Menge von tgds über einem festen Datenbankschema. Dann nennen wir den gerichtete Graphen $dep = (pos(\Sigma), E)$ einen *Abhängigkeitsgraphen*, wobei $pos(\Sigma)$ die Menge der Positionen R_i bezeichne. Die Menge der Kanten E ergibt sich dabei wie folgt: Für jede tgd $\phi(x, y) \rightarrow \exists z\ \psi(x, z) \in \Sigma$ und

1. für jedes $x_i \in x$, dass in ϕ an der Position R_i und in ψ an der Position S_j vorkommt, ziehe eine Kante $R_i \rightarrow S_j$ (sofern eine solche Kante noch nicht existiert),
2. für jedes $x_i \in x$, dass in ϕ an der Position R_i vorkommt und für jedes $z_i \in z$, das in ψ an Position T_k vorkommt, ziehe eine besondere Kante $R_i \overset{*}{\rightarrow} T_k$ (sofern eine solche Kante noch nicht existiert).

Die Abhängigkeitsmenge Σ ist *schwach azyklisch* (WA), wenn der zugehörige Abhängigkeitsgraph $dep(\Sigma)$ keinen Zyklus enthält, der durch eine spezielle Kante führt. ■

Beispiel 3.6 (Fortsetzung)**.** Seien $S(n)$ und $R(n, m)$ sowie $\Sigma = \{S(n) \rightarrow \exists M : R(n, M);\ R(n, m) \rightarrow S(m)\}$ wie oben gegeben. Der zugehörige Abhängigkeitsgraph ist in Abbildung 3.3a zu sehen. Er besteht aus einer normalen und einer besonderen Kante für $S(n) \rightarrow \exists M : R(n, M)$ und einer weiteren normalen Kante für $R(n, m) \rightarrow S(m)$. Es ergibt sich somit ein Zyklus durch eine besondere Kante. Die Abhängigkeitsmenge Σ ist somit nicht schwach azyklisch.



(a) Schwache Azyklizität **(b)** Constraint Rewriting

Abbildung 3.3. Abhängigkeitsgraphen für verschiedene Terminierungskriterien (für das Beispiel 6): Der Abhängigkeitsgraph $dep(\Sigma)$ in (a) weist einen Zyklus durch eine besondere Kante auf und fürchtet somit die Nicht-Terminierung. Das Constraint Rewriting aus (b) hingegen sagt eine Terminierung voraus. Grund hierfür ist die Auflösung des Zyklus durch das Hinzufügen von Addornments.

□

Das Kriterium der schwachen Azyklizität garantiert die Terminierung des Chase und kann in polynomialer Zeit berechnet werden [GMS12]. Es ist jedoch sehr streng, sodass Abhängigkeitsmengen existieren können, welche terminieren, ohne dass die schwache Azyklizität dies erkennt (siehe Beispiel 3.3a). Das Kriterium der schwachen Azyklizität wurde daher stetig weiterentwickelt. Die erste Erweiterung ist beispielsweise die *Stratifizierung* (C-Str). Sie zerlegt die Menge der Abhängigkeiten in unabhängige Teilmengen [DNR08], welche sie anschließend auf schwache Azyklizität überprüft. Zyklen durch besondere Kanten, welche mehrere dieser Teilmengen überspannen, können so vernachlässigt werden. Die Stratifizierung garantiert für jede Datenbank D eine Chase-Sequenz, welche in polynomialer Zeit bzgl. D terminiert [MSL09a]. Sie garantiert jedoch nicht die Terminierung aller Chase-Sequenzen.

Eine andere Erweiterung der schwachen Azyklizität ist das *Safety-Kriterium* (SC). Es berücksichtigt *affected positions*, d.h. Positionen, welche mit Nullwerten assoziiert sind [MSL09b]. Das Kriterium der *Safe Restriction* (SR) verbindet die Konzepte der Stratifizierung und der Safety [MSL09a]. Weitere Terminierungskriterien sind zudem die *Inductive Restriction* (IR) sowie die *super-schwache Azyklizität* (SwA). Ersteres verfeinert das Safety-Kriterium, indem Bedingungen in einer geschickteren Weise partitioniert werden [MSL09b]. Die super-schwache Azyklizität hingegen konstruiert statt eines Abhängigkeitsgraphen einen *Triggergraphen*, dessen Kanten die Beziehungen zwischen den Bedingungen beschreiben [Mar09]. Insgesamt ergeben sich so die in Abbildung 3.4 dargestellten Terminierungshierarchien. Es gilt somit:

- WA $\subsetneq$ SC, WA $\subsetneq$ C-Str und C-Str $\nparallel$ SC
- C-Str $\cup$ SC $\subsetneq$ SR und SR $\subsetneq$ IR
- WA $\subsetneq$ SC $\subsetneq$ SwA.

Nachzulesen sind die zugehörigen Beweise beispielsweise in [GMS12].

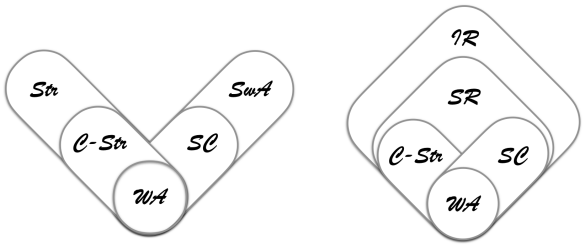


Abbildung 3.4. Terminierungskriterien, zusammengefasst nach [GMS12].

Auch das sogenannte *Constraint Rewriting* [GST11] kann für den Test auf Terminierung herangezogen werden. Dabei wird eine Menge von tgds in eine äquivalente Menge von tgds transformiert, welche die Terminierungseigenschaften auch dann erfüllt, wenn dies für die ursprüngliche tgd-Menge nicht der Fall war. Die gängigsten Rewriting-Techniken sind Adn und Adn$^+$. Letztere beweist schließlich die Terminierung unseres oben angegebenen Beispiels.

Beispiel 3.7 (Fortsetzung)**.** Die Anwendung des Constraint Rewriting-Algorithmus Adn$^+$ liefert für das obige Beispiel bzgl. $S(n)$, $R(n,m)$ und $\Sigma = \{S(n) \to \exists M : R(n,M);\ R(n,m) \to S(m)\}$ einen erweiterten Abhängigkeitsgraphen. Die Knotenmenge des Graphen besteht nun aus den adornten Positionen. Somit ändern sich die Beziehungen zwischen den Knoten und der Zyklus durch eine besondere Kante kann aufgebrochen werden. Der adornte Abhängigkeitsgraph aus Abbildung 3.3b ist somit azyklisch und der Chase terminiert. □

Eine ausführliche Vorstellung der einzelnen Terminierungskriterien kann in [GMS12; GST11] oder [MSL09a] nachgelesen werden. Ein Vergleich der vorgestellten Kriterien anhand eines gemeinsamen Studierenden-Beispiels ist zudem in [Gör20] zu finden.

3.4. Provenance

Provenance beschreibt die Herkunft eines konkreten (meist digitalen) Ergebnisses zurück zu seinen physischen Wurzeln. Dies können neben Sensordaten auch Kunstwerke, Lebensmittel, Forschungsergebnisse oder anderes sein. Im wissenschaftlichen Kontext bezieht sich Provenance dabei stets auf die Rückverfolgung eines konkreten Datensatzes *(Data Provenance)* oder die Nachverfolgung von Arbeitsabläufen *(Workflow Provenance)*. Sie kann *extensional*, d.h. durch Angabe der originalen Daten, oder beschreibend beispielsweise im Sinne *intensionaler Antworten* beantwortet werden.

Neben der Data Provenance sowie der Workflow Provenance unterscheidet die Literatur zudem noch die *Provenance-Metadaten*, welche Metadaten über den Produktionsprozess sammelt sowie die *Information System Provenance*, welche den Prozess auf seinen digitalen Anteil, das Informationssystem, einschränkt. Wir konzentrieren uns im Folgenden auf die Data Provenance. Die anderen drei Typen werden wir nur kurz anreisen, eine detaillierte Beschreibung kann in [HDL17] nachgelesen werden.

Wie in Abbildung 3.5 zu sehen, können die vier Provenance-Typen nach ihrem Abstraktionslevel bzw. ihrer Automatisierbarkeit geordnet werden. So ist die Data Provenance der speziellste Provenance-Typ, welcher weitestgehend automatisiert werden kann. Provenance-Metadaten hingegen lässt sich kaum automatisieren und gilt als allgemeinster Provenance-Typ. Die Information System Provenance und ihre Spezialisierung, die Workflow Provenance, befinden sich zwischen diesen beiden Extremen.

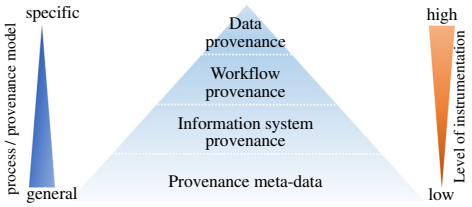


Abbildung 3.5. Provenance-Hierarchie [HDL17].

Provenance-Metadaten *Provenance-Metadaten*, welche alle möglichen Metadaten eines Produktionsprozesses erfassen, sind die allgemeinste Form der Provenance. So bieten sie dem Nutzer den „größten Freiheitsgrad bei der Modellierung, Speicherung oder dem Zugriff auf die Provenance eines beliebigen Typs, dessen Interna nicht offengelegt werden [HDL17]. Diese Freiheit und Allgemeingültigkeit von Provenance-Metadaten verhindert jedoch in der Regel eine automatische Generierung oder Auswertung dieses Provenance-Typs. Sie können daher oft nur manuell bearbeitet werden. Während allgemeine Metadaten darauf abzielen, den Daten eine Bedeutung zuzuweisen, sind Provenance-Informationen beschreibend für den Datenherstellungsprozess. Zusammenfassend werden Provenance-Metadaten als Metadaten definiert, die einen beliebigen Produktionsprozess unter Verwendung eines beliebigen Datenmodells und Berechnungsmodells beschreiben.

Information System Provenance Die Rückverfolgbarkeit von Produktionsprozessen ist Schwerpunkt der *Information System Provenance*. Diese sammelt „Metadaten über Prozesse innerhalb eines Informationssystems“ [HDL17] und steht in der Provenance-Hierarchie über der Provenance-Metadaten und unterhalb der Workflow Provenance. Sie wird je nach Literatur jedoch häufig vernachlässigt [Fre+08; BT07; Tan07].

Workflow Provenance Die *Workflow Provenance* [DF08] arbeitet als Spezialisierung der Information System Provenance auf sogenannten Workflows. Hierbei handelt es sich um einen gerichtete Graphen, dessen Knoten beliebigen Funktionen oder Modulen (mit eigenen Ein- und Ausgaben sowie Parametern) entsprechen. Die Daten- und Kontrollflüsse zwischen den einzelnen Knoten werden durch gerichtete Kanten angegeben. Einer der bekanntesten Standards zur Darstellung dieser Arten von Provenance ist der sogenannte *PROV Standard*[2], welcher „ein Datenmodell, Serialisierungen und Definitionen zur Unterstützung des Austauschs von Herkunftsinformationen im Web“ definiert (Quelle: Wikipedia, 13.07.2022).

[2]PROV Standard: https://www.w3.org/TR/2013/NOTE-prov-overview-20130430/

Data Provenance Die *Data Provenance* [CCT09; GT17] ermöglicht die automatische Erfassung und Verarbeitung von Provenance-Informationen über strukturierte Datenmodelle und deklarative Abfragesprachen mit klar definierter Semantik der einzelnen Operatoren. Sie ist sehr spezifisch und arbeitet rein auf der Daten-Ebene. Sie beschreibt, (1) woher ein Ergebnistupel kommt (***where***), (2) warum (***why***) und (3) wie ein bestimmtes Ergebnis erreicht wurde (***how***) oder aber (4) warum ein erwarteter Wert im Ergebnis fehlt (***why not***), wie in Tabelle 3.4 zusammengefasst.

Anfrage-Typ	**Fragestellung**
where	Woher kommen die Daten?
why	Warum dieses Ergebnis?
how	Wie kommt das Ergebnis zustande?
why not	Warum fehlt ein bestimmtes Element im Ergebnis?

Tabelle 3.4. Provenance-Anfragen.

3.4.1. Data Provenance

Data Provenance beschreibt die Herkunft eines Anfrageergebnisses auf Datenebene. Da sie auf den Originaldaten bzw. ihren Provenance-IDs arbeitet, ist sie sehr spezifisch und kann zudem weitstgehend automatisiert verarbeitet werden. Sei hierfür I eine Datenbankinstanz und Q eine gegebene Anfrage. Dann beschreibt Data Provenance die Herkunft des Anfrageergebnisses $Q(I)$ durch Angabe der zugehörigen ID-Liste, Zeugenbasis [BKT01], eines Provenance-Polynoms [GKT07; ADT11], des Provenance-Graphen [Aca+10], des Provenance Games [KLZ13] oder ähnliches.

Data Provenance unterscheidet typischerweise vier verschiedene Anfrage-Typen — ***where***, ***why***, ***how*** und ***why not*** — und vier verschiedene Antwort-Typen — extensional, intensional, anfragebasiert, modifikationsbasiert. In dieser Arbeit konzentrieren wir uns insbesondere auf ***where***-, ***why***- und ***how***-Anfragen mit *extensionaler Antwort* sowie auf ***why***- und ***why not***-Anfragen mit *intensionaler Antwort*. Im Falle einer extensionalen Antwort lassen sich die ***why***- und (relationen- oder tupel-basierte) ***where***-Provenance aus dem Ergebnis der ***how***-Provenance ableiten. Basierend auf ihrem Informationsgehalt leiten wir folgende Reduktion ab:

$$\textbf{\textit{where}} \preccurlyeq \textbf{\textit{why}} \preccurlyeq \textbf{\textit{how}}.$$

Für die Rekonstruktion der (minimalen) Teil-Datenbank konzentrieren wir uns zunächst auf den ***why***- und ***how***-Nachweis mit extensionalen Antworten. Mit Hilfe von *Provenance-Polynomen* [GKT07; ADT11] und *(minimalen) Zeugenbasen* [BKT01] können wir so die Reproduzierbarkeit bzw. Replizierbarkeit des Anfrageergebnisses gewährleisten. Die Polynome sind durch einen kommutierten Semiring $(\mathbb{N}[X], +, \cdot, 0, 1)$ mit $+$ zur Duplikateliminierung und $\cdot$ für die Kombination von Tupeln definiert (siehe Definition 3.38) und die Zeugenbasen als Menge von Zeugenmengen (siehe Definition 3.37). Duplikate werden hier durch verschiedene Zeugenmengen angegeben. Doch schauen wir uns zunächst ein konkretes Beispiel an.

Wir sind daran interessiert, eine konkrete Rechenvorschrift (***how***-Provenance) oder zumindest alle notwendigen Quell-Tupel (***why***-Provenance) zu spezifizieren. Für das gegebene Beispiel aus Abbildung 3.6 wird das Provenance-Polynom berechnet als $(t_1 \cdot (t_1 + t_2)) + (t_2 \cdot (t_1 + t_2))$, wobei t_1 und t_2 Tupel-IDs der Quellinstanz I sind. Aus den Teil-Polynomen t_1^2, $2t_1t_2$, t_2^2 ergibt die zugehörige Zeugenbasis $\{\{t_1\}, \{t_2\}, \{t_1, t_2\}\}$. Eine minimale Zeugenbasis wäre $\{\{t_1\}\}$ oder $\{\{t_2\}\}$. Durch die anschließende Reduktion der Zeugenbasis auf die enthaltenen Tupel-IDs erhalten wir die für die tupel-basierte ***where***-Provenance relevante Zeugenmenge $\{t_1, t_2\}$ sowie eine Liste der der zugehörigen Relationennamen R (relationen-basierte ***where***-Provenance).

relationen-basierte ***where***-Provenance (Relationennamen)	tupel-basierte ***where***-Provenance (Tupel-IDs)	***why***-Provenance (Zeugenbasen)	***how***-Provenance (Provenance-Polynome)
R	$\{t_1, t_2\}$	$\{\{t_1\}, \{t_2\}, \{t_1, t_2\}\}$	$(t_1 \cdot (t_1 + t_2)) + (t_2 \cdot (t_1 + t_2))$

Relationennamen — Tupelliste — Zeugenbasis

Abbildung 3.6. Reduktion der ***where***-, ***why***- und ***how***-Provenance am konkreten Beispiel.

where*-Provenance** Wie in Abbildung 3.6 dargestellt, können die tupel-basierte ***where-Provenance (siehe Definition 3.34) sowie die relationen-basierte ***where***-Provenance (siehe Definition 3.33) aus der Zeugenbasis der ***why***-Provenance abgeleitet werden. Stehen die originalen Daten nicht zur Verfügung, ist der gespeicherte Informationsgehalt dieser Interpretation der ***where***-Provenance im Vergleich zur ***how***-Provenance allerdings sehr gering. So liefert die tupel-basierte ***where***-Provenance lediglich die richtige Anzahl an Originaltupeln zurück, speichert jedoch keine Informationen über Duplikate im Anfrageergebnis oder die Zusammenhänge der einzelnen beteiligten Relationen. So entfallen beispielsweise alle Tupel, welche zwar an der Auswertung der Anfrage beteiligt waren, jedoch nicht im Ergebnis enthalten sind (siehe Beispiel 3.9). Auch der Verbund oder die Vereinigung zweier Relationen kann bei der ***where***-Provenance nicht mehr rekonstruiert werden.

Definition 3.33 (relationen-basierte ***where***-Provenance)**.** Die *relationen-basierte* ***where****-Provenance* entspricht der Menge der Relationennamen aller am Anfrageergebnis beteiligten Tupel. ■

Definition 3.34 (tupel-basierte ***where***-Provenance)**.** Die *tupel-basierte* ***where****-Provenance* entspricht der Menge aller am Anfrageergebnis beteiligten Tupel-Identifikatoren. ■

Anders als die tupel- sowie die relationen-basierte ***where***-Provenance kann die *attributwert-basierte* ***where****-Provenance* (siehe Definition 3.35) nicht aus der ***why***-Provenance reduziert werden. Grund hierfür ist die zusätzliche Information über den Attributnamen A, welche aus der Zeugenbasis nicht abgeleitet werden kann. Auch legt diese Interpretation der ***where***-Provenance eine Kenntnis der Originaldaten nahe, um die konkrete Angabe der Position des gesuchten Attributwertes (durch Angabe des Relationennamen, der Tupel-ID und des Attributnamen) innerhalb der Datenbankinstanz bestmöglich ausnutzen zu können. Wir gehen im weiteren Verlauf der Arbeit jedoch davon aus, dass die Originaldaten nach Berechnung der Provenance nicht mehr zur Verfügung stehen. Es ist uns also nicht möglich, mit Hilfe der Tupel-Identifikatoren zusätzliche Informationen aus der Quell-Datenbank zu rekonstruieren.

Beispiel 3.8. Sei I die in Tabelle A.1 gegebene Datenbankinstanz bestehend unter anderem aus der Relation `Student`. Sei weiter Q die Anfrage $\pi_{\texttt{studies}}(\texttt{Student})$. Dann ergibt sich für $Q(I)$ folgendes Anfrageergebnis:

`studies`	**attributwert-basiert**	**tupel-basiert**	**relationen-basiert**
teaching	$\{(\text{Student}, S_1, \text{studies})\}$	$\{S_1\}$	Student
mathematics	$\{(\text{Student}, S_2, \text{studies})\}$	$\{S_2\}$	Student
engineering	$\{(\text{Student}, S_3, \text{studies}), (\text{Student}, S_7, \text{studies})\}$	$\{S_3, S_7\}$	Student
computer science	$\{(\text{Student}, S_4, \text{studies}), (\text{Student}, S_5, \text{studies}),$ $(\text{Student}, S_6, \text{studies})\}$	$\{S_4, S_5, S_6\}$	Student
theory	$\{(\text{Student}, S_8, \text{studies})\}$	$\{S_8\}$	Student

Zu sehen sind die drei Interpretationen der ***where***-Provenance. Während die attributwert-basierte ***where***-Provenance explizit auf die Position des originalen Attributwertes verweist, ist die tupel-basierte ***where***-Provenance sehr allgemein. Diese liefert lediglich den zugehörigen Relationennamen. Wir konzentrieren uns im Folgenden daher stets auf die tupel-basierte ***where***-Provenance. □

Definition 3.35 (attributwert-basierte ***where***-Provenance)**.** Ein Attributwert kann über seinen Relationennamen R, seinen Tupelidentifikator t sowie einen Attributnamen A eindeutig zugeordnet werden. Das zugehörige Tupel (R, t, A) nennen wir ***where****-Annotation*. Die *attributwert-basierte* ***where****-Provenance* entspricht der Menge der ***where***-Annotationen aller am Anfrageergebnis beteiligten Tupel. ■

Die formale Definition der attributwert-basierten ***where***-Provenance kann in [BKT01; CCT09] nachgelesen werden. Wir werden sie im weiteren Verlauf der Arbeit nicht weiter vertiefen.

Data Lineage und *why*-Provenance Bereits 2000 beschäftigten sich Cui et al. mit dem „Problem der Datenabstammung in einer Warehousing-Umgebung“ [CW00]. Sie suchten nach einer Möglichkeit, für ein gegebenes Datenelement in einer materialisierten Warehouse-Ansicht die Menge der Quelldatenelemente zu identifizieren, die das Sicht-Element erzeugt haben. Hierfür formalisieren die Autoren das zugehörige Lineage-Problem und entwickeln Lineage-Tracing-Algorithmen für relationale Sichten mit Aggregation. Auch in Datenbank-Visualisierungsumgebungen kommt Lineage zum Einsatz [WS97].

(Data) Lineage beschreibt allgemein die Beziehung zwischen den Tupel eines Endprodukts und seinen Quell-Tupeln [CCT09]. Sei hierfür I eine Quell-Instanz, Q eine Auswertungsanfrage und t ein Ergebnistupel in $Q(I)$. Dann definiert Lineage alle an der Berechnung von t relevanten Quell-Tupel, die sogenannten *Zeugen* (siehe Definition 3.36). Die Definition von Data Lineage ist jedoch keinesfalls eindeutig und wird seit den 2010er Jahren kaum noch verwendet. Sie bildet jedoch die Basis für die heute noch verwendete ***why***-Provenance.

Definition 3.36 (Zeuge, [BKT01])**.** Sei I eine Datenbankinstanz, Q eine Anfrage über I, und t ein Tupel in $Q(I)$. Dann heißt eine Instanz $I_w \subseteq I$ *Zeuge für t bzgl. Q*, wenn $t \in Q(I_w)$. ■

Die Anzahl solcher Zeugen wächst aufgrund einer Vielzahl „irrelevanter“ Quell-Aufzeichnungen jedoch sehr schnell, sodass im schlimmsten Fall die gesamte Quell-Instanz übernommen werden muss. Das Problem des exponentiellen Wachstums lösen Buneman et al. in [BKT01] durch die Einführung einer Zeugenbasis (siehe Definition 3.37). Wie in [BKT01] aufgezeigt, ist diese keinesfalls eindeutig. So können zwei äquivalente Anfragen Q_1 und Q_2 zwei unterschiedliche Zeugenbasen $W_{Q_1,I}(t)$ und $W_{Q_2,I}(t)$ haben, deren minimale Zeugenbasen jedoch wieder übereinstimmen, d.h. $W^{\min}_{Q_1,I}(t) = W^{\min}_{Q_2,I}(t)$. Ist keine konkrete Anfrage Q gegeben oder relevant, schreiben wir für die Zeugenbasis auch kurz w.

Definition 3.37 (Zeugenbasis)**.** Sei I eine Datenbankinstanz, Q eine Anfrage über I und t ein Tupel in $Q(I)$. Die Zeugenbasis $W_{Q,d}(t)$ entspricht der Menge aller Zeugen, d.h. $W_{Q,d}(t) = \{I_{w_1}, ..., I_{w_n}\}$ mit den Zeugen I_{w_i} und $1 \leq i \leq n, k \in \mathbb{N}$. Weiter heißt eine Zeugenbasis $W^{\min}_{Q,I}(t)$ heißt *minimal* für Q, wenn keine Teilmenge von $W_{Q,I}(t)$ selber Zeugenbasis für Q ist. ■

Wie oben beschrieben, spezifiziert die ***why***-Provenance alle zur Berechnung eines Ergebnistupels t relevanten Quell-Tupel. Dies kann durch Angabe der zugehörigen Zeugenbasis beantwortet werden. Die Zeugenbasis ist abhängig von der Auswertungsanfrage Q sowie der Quell-Instanz I, ansonsten aber eindeutig. Die formale Definition der ***why***-Provenance kann in [BKT01; CCT09] nachgelesen werden.

how*-Provenance** Die ***how-Provenance beschreibt, wie ein Anfrageergebnis zustande gekommen ist. Hierfür liefert sie eine konkrete Berechnungsvorschrift in Form eines Provenance-Polynoms. Diese auf der positiven Relationenalgebra definierten Polynome über dem Semiring $(\mathbb{N}[X], +, -, 0, 1)$ mit X als Menge von Tupel-Identifikatoren basieren auf den sogenannten *Provenance-Semiringen* aus Definition 3.38. Sie verarbeiten die Selektion und Projektion, den natürlichen Verbund sowie die Vereinigung und wurde bereits um die Aggregation sowie Gruppierung erweitert [ADT11]. Die formalen Definitionen der *positiven K-Algebra* sowie der Provenance-Polynome können in [GKT07; GT17] nachgelesen werden.

Definition 3.38 (Provenance-Semiring, [GKT07])**.** Sei I eine Datenbankinstanz und X die Menge der zugehörigen Tupel-IDs. Dann ist der *Provenance-Semiring der positiven Algebra für I* definiert als Semiring von Polynomen $(\mathbb{N}[X], +, -, 0, 1)$, wobei $\mathbb{N}[X]$ die Menge der Polynome mit Koeffizienten aus $\mathbb{N}$ und Variablen aus X ist. Weiter sind $+$ und - wie üblich definiert. ■

Kurz gesagt beschreibt die ***how***-Provenance eine Berechnungsvorschrift eines Ergebnistupels t, indem die Anfrage als Polynom über die am Ergebnis beteiligten Tupel-Identifikatoren dargestellt wird. Hierbei entspricht $+$ der Vereinigung sowie Projektion (Duplikateliminierung) und $\cdot$ dem natürlichen Verbund (Kombination) zweier Tupel. Da die Selektion die Tupel selbst nicht verändert, wird sie im Semiring auch nicht dargestellt. Schauen wir uns abschließend noch ein vergleichendes Beispiel an:

Beispiel 3.9. Sei I die in Tabelle A.1 gegebene Datenbankinstanz bestehend unter anderem aus den Relationen `Student`, `Participant` und `Lecturer`. Sei weiter Q die Anfrage `SELECT firstName, lecturer FROM Student NATURAL JOIN Participant NATURAL JOIN Lecturer WHERE studentID` $= 1$. Dann ergibt sich für $Q(I)$ folgendes Anfrageergebnis:

firstName	lecturer	***where***	***why***	***how***	Lineage
Donald	Professor A	$\{S_1, P_1, P_{20}, P_{24}\}$	$\{\{S_1, l_1, P_1\}, \{S_1, L_2, P_{20}\}, \{S_1, L_{10}, P_{24}\}\}$	$S_1L_1P_1 + S_1L_2P_{20} + S_1L_{10}P_{24}$	$\{S_1, L_1, L_4, L_{10}, P_1, P_{20}, P_{24}\}$
Donald	Lecturer A	$\{S_1, P_1\}$	$\{\{S_1, L_2, P_1\}\}$	$S_1L_2P_1$	$\{S_1, L_2, P_1\}$
Donald	Professor B	$\{S_1, P_5\}$	$\{\{S_1, L_3, P_5\}\}$	$S_1L_3P_5$	$\{S_1, L_2, P_1\}$
Donald	Professor C	$\{S_1, P_{12}\}$	$\{\{S_1, L_5, P_{12}\}\}$	$S_1L_5P_{12}$	$\{S_1, L_5, P_{12}\}$

Zu sehen ist die tupel-basierte ***where***-Provenance als Menge der Tupel-Identifikatoren, die Zeugenbasis der ***why***-Provenance, das Provenance-Polynom der ***how***-Provenance sowie die Zeugenmenge der Lineage. Während die Lineage alle am Ergebnis beteiligten IDs zurück gibt, beschränkt sich die ***where***-Provenance auf die IDs des jeweiligen Ergebnistupels. Die „Zwischenrelation“ `Participant` wird in diesem Falle unterschlagen. Weiter lassen die drei Zeugen der Zeugenbasis lassen auf drei Duplikate schließen. Eine konkrete Berechnungsvorschrift liefert schließlich die Provenance-Polynome. Hieraus können wir ablesen, dass die Relationen `Student`, `Participant` und `Lecturer` gejoint wurden und das Tupel (Donald, Professor A) insgesamt drei Mal im Ergebnis auftaucht. □

Für die Integration der Aggregation wird der Semiring K mittels Tensorprodukt um einen kommutativen Monoid M erweitert. Es entsteht der K-Semimodul $K \otimes M$, eine additive abelsche Gruppe $(M, +)$ mit einer Abbildung $K \times M \to M$, deren Elemente als Linearkombination

$$k_1 \otimes l(m_1) +_{K\otimes M} \cdots +_{K\otimes M} k_n \otimes l(m_n),$$

ausgedrückt werden, wobei $l : M \to K \otimes M$ die Elemente von M auf Tensoren abbildet, die den gesamten Raum erzeugen. Die Provenance-Polynome (mit oder ohne Erweiterung um Aggregation) sind jedoch nur auf der positiven relationalen Algebra (d.h. ohne Differenz oder Negation) definiert. Erste Ansätze für eine entsprechende Erweiterung sind unter anderem in [ABS15] zu finden. Wir beschränken uns auch hier auf die Angabe eines konkreten Beispiels:

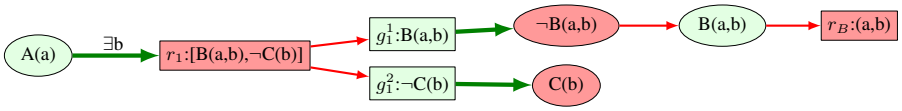
Beispiel 3.10. Sei I die in Tabelle A.1 gegebene Datenbankinstanz bestehend unter anderem aus der Relation `Grade`. Sei weiter Q die Anfrage `SELECT AVG(grade) FROM Student NATURAL JOIN Grade WHERE firstName` $=$ 'Donald'. Dann ergibt sich für $Q(I)$ folgendes Anfrageergebnis:

AVG(grade)	***how***
$2,46$	$2.0 \otimes S_1 \cdot G_1 +_{K\otimes M} 3.7 \otimes S_1 \cdot G_5 +_{K\otimes M} 1.0 \otimes S_1 \cdot G_{12} +_{K\otimes M} 2.3 \otimes S_1 \cdot G_{19} +_{K\otimes M} 3.3 \otimes S_1 \cdot G_{21}$

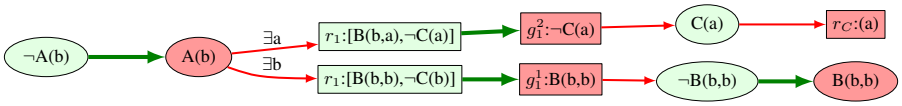
Für die Angabe der zu `AVG(grade)` gehörenden Provenance eignet sich aufgrund der Aggregation insbesondere die ***how***-Provenance. Ebenso wie die Zeugenbasis $\{\{\{S_1, G_1\}, \{S_1, G_5\}, \{S_1, G_{12}\}, \{S_1, G_{19}\}, \{S_1, G_{21}\}\}\}$, gibt das Provenance-Polynom Auskunft über die Zusammensetzung bzw. Entstehung des Ergebnistupels. Das Polynom besteht aus der Summe der einzelnen Teil-Polynome, welche mittels Tensorprodukt mit ihrem zugehörigen Attributwert multipliziert werden. So entspricht $2.0 \otimes S_1 \cdot G_1$ beispielsweise dem Polynom des ersten Teil-Ergebnisses $(\text{Donald}, 2.0)$. Das Polynom speichert neben der Berechnungsvorschrift somit zudem die zu aggregierenden Attributwerte. Diese Information geht bei Verwendung der ***why***-Provenance hingegen verloren. □

Für die Angabe einer extensionalen Provenance-Antwort genügt somit die Berechnung der ***how***-Provenance. Die Berechnung und Speicherung dieser Polynome erfolgt etwa über die schrittweise Verfolgung der algebraischen Grundoperationen der Anfrage oder über graphentheoretische Ansätze wie beispielsweise den Orchestra Provenance-Graph aus Abbildung 4.4c.

why not*-Provenance** Die Frage, warum ein erwartetes Tupel nicht im Anfrageergebnis zu finden ist (why not***-Provenance), kann unter anderem durch *Provenance Games* beantwortet werden [LLV07; HDL17]. Dies ist ein spieltheoretischer Ansatz zur Bewertung von Anfragen, welcher zur Beantwortung der ***how***-, ***why***- sowie ***why not***-Provenance verwendet werden kann. Hierfür wird eine gegebene Anfrage als *Provenance-Graph* interpretiert und zweifarbig eingefärbt.



(a) Game provenance graph $\Gamma(f_{\mathtt{A}}(a))$ for $A(a) \in Q_{\mathtt{neg}}(D)$.



(b) Game provenance graph $\Gamma(f_{\neg\mathtt{A}}(b))$ for $A(b) \notin Q_{\mathtt{neg}}(D)$

Abbildung 3.7. Provenance-Graph für Q_{neg} auf der Datenbank $D = \{B(a,b), B(b,a), C(a)\}$. Sowohl der ***Why***-Graph als auch der ***Why not***-Graph können Blatt-Knoten enthalten, die vorhandene oder fehlende Eingabefakten darstellen [KLZ13].

So zeigt Abbildung 3.7(a) den Provenance-Graphen zum Tupel $t = A(a)$. Da das Tupel $B(a,b)$ in der Quell-Instanz existiert, kann $A(a)$ erfolgreich aus dem Ziel g_1^1 abgeleitet werden (***why***-Provenance). Eine Ableitung aus dem zweiten Ziel g_1^2 ist aufgrund der Nicht-Existenz des Quell-Tupels $C(b)$ hingegen nicht möglich. Betrachten wir alternativ $\neg A(b)$, welches nicht Teil des Anfrageergebnisses $Q_{\text{neg}}(D)$ ist. Das Provenance Game zeigt uns, dass beide Versuche $A(b)$ abzuleiten scheitern. Eine entsprechende Einfärbung ist somit ein Nachweis im Sinne der ***why not***-Provenance. Details zur Erstellung sowie den Gebrauch von Provenance Games kann in [KLZ13] nachgelesen werden. Eine Implementierung der Provenance-Graphen findet sich zudem in Orchestra [Ive+08] sowie GProM [Ara+18], vorgestellt in Abschnitt 4.2.

Während die ***where***-, ***why***- und ***how***-Provenance die Existenz eines Ergebnistupels erklären, kann die Antwort auf auf eine ***why not***-Anfrage genutzt werden, um das Fehlen eines erwarteten Anfrageergebnisses zu erklären. Die für die ***where***-, ***why***- und ***how***-Provenance genutzte extensionale Antwort eignet sich in diesem letzten Falle jedoch nicht. Die Literatur unterscheidet daher bis zu vier verschiedene Provenance-Antworten: *extensionale*, *intensionale*, *anfrage-basierte* und *modifikations-basierte Anfragen*. Während die extensionale Antwort die Originaldaten zurückgibt, liefern die anderen drei Antwort-Typen Beschreibungen des Anfrageergebnisses (siehe Tabelle 3.5). Wir geben hier einen kurzen Überblick, Details können in [HDL17] nachgelesen werden.

Antwort-Typ	**Ergebnis**
extensional	Tupel aus den Originaldaten
intensional	Beschreibung der Daten
anfrage-basiert	relevante Selektionsprädikate
modifikations-basiert	Vorschlag zur minimalen Änderung der Auswertungsanfrage

Tabelle 3.5. Provenance-Antworten.

Extensionale Antwort Das Ergebnis einer relationalenalgebraischen Anfrage ist stets ein aus den Originaldaten extrahiertes oder berechnetes Datum. Die Ausgabe der hierfür benötigten Originaldaten wird als *extensionale Antwort* bezeichnet. Da an dieser Stelle keine weiteren Abstraktionsschritte notwendig sind, eignet sie sich insbesondere für die Beantwortung der ***where***-, ***why***- und ***how***-Provenance. Wir fokussieren uns daher im Folgenden stets auf diesen Antwort-Typ. Neben der extensionalen Antwort existieren noch drei weitere Antwort-Typen, welche eine Beschreibung der Daten (*intensional*), die relevanten Selektionsprädikate (*anfrage-basiert*) oder aber einen Vorschlag zur minimalen Änderung der Auswertung (*modifikations-basiert*) liefern.

Anfrage-basierte Antwort *Anfrage-basierte Antworten* sind ein Spezialfall der intensionalen Antworten. Sie umfassen alle Anfrageoperatoren, die für die mögliche Beseitigung eines fehlenden Anfrageergebnisses t_R verantwortlich sind. Die anfragebasierte Antwort liefert dabei gerade die Selektionsprädikate, die für die Auswertung der ***why***- und ***how***-Provenance relevant sind. Einen Überblick über Ansätze, welche eine anfrage-basierte Antwort erfordern oder verarbeiten können ist in [HDL17] zu finden. Der bekannteste Ansatz ist sicherlich die oben beschriebene ***why not***-Provenance [CJ09].

Intensionale Antwort Im Falle einer *intensionalen Antwort* liefert das System keine konkreten Daten. Stattdessen wird eine Beschreibung der Daten gegeben. Wegen ihrer allgemeinen Anwendbarkeit kann dieser Antwort-Typ für alle vier Provenance-Anfragen eingesetzt werden. Details hierzu können beispielsweise in [Mot94; PY99] nachgelesen werden.

Modifikations-basierte Antwort Die *modifikations-baiserte Antwort* eignet sich insbesondere zur Beantwortung von ***why not***-Anfragen. Das System macht in diesem Falle einen Vorschlag zur minimalen Änderung der Auswertungsanfrage Q, sodass das im Ergebnis $Q(I)$ vermisste Element t_R nach Auswertung der transformierten Anfrage Q' ebenfalls im Ergebnis enthalten ist, d.h. $t_R \in Q'(I)$. [HDL17] liefert hierzu verschiedene Ansätze unter anderem für relationale Anfragen sowie (umgekehrte) Top-k-Anfragen.

3.5. Schemaevolution

Die durch das Erstellen, Löschen, Verketten oder Aufspalten von Tabellen und Attributen auftretenden Schemaänderungen können durch sogenannte *Schemamodifikationsperatoren* (SMOs, siehe Definition 3.17) beschrieben werden. Die häufigsten Änderungen entsprechen hierbei `CREATE Table`, `DROP Table`, `ADD Column`, `DROP Column` und `RENAME Column` [QLS13; WN11]. Eine vollständige Liste ist in Tabelle 3.6 zu sehen.

SMOs
`COPY Table` R `INTO` S
`CREATE Table` R(a,b,c)
`DECOMPOSE Table` R(a,b,c) `INTO` S(a,b), T(b,c)
`DISTRIBUTE Table` / `PARTITION Table` R `INTO` S `WITH` cond, T
`DROP Table` R
`JOIN Table` R, S `INTO` T `WHERE` cond
`MERGE Table` R, S `INTO` T
`RENAME Table` R `INTO` S

SMOs
`ADD Column` d [`AS` const|func(a,b,c)] `INTO` R
`COPY Column` c `FROM` R `INTO` S `WHERE` cond
`DROP Column` c `FROM` R
`MOVE Column` c `FROM` R `INTO` S `WHERE` cond
`RENAME Column` b `IN` R `TO` d
`NOP`

Tabelle 3.6. Schemamodifikationsoperatoren (SMOs), nach [CMZ08; Moo+08].

Formal gesprochen ist ein SMO eine Funktion, welche als Eingabe ein relationales Schema und die zugrunde liegende Datenbank erhält und als Ausgabe eine (modifizierte) Version des Eingabeschemas sowie eine migrierte Version der Datenbank erzeugt. Seien hierfür S_t und S_{t+1} zwei Datenbankschemata einer sich ändernden Datenbank. Dann beschreibt eine SMO die Schemaänderungen von S_t nach S_{t+1}.

Die SMOs können zudem als *disjunctive embedded dependencies* (ded, siehe Definition 3.12) dargestellt werden [CMZ08], eine Formalisierung, welche den für die Anwendung des Chase notwendigen s-t tgds stark ähnelt. In unserem Falle reicht eine Formalisierung als Menge von s-t tgds jedoch aus (siehe Kapitel 7).

3.6. Privacy

Bevor wir uns mit den konkreten Anonymisierungsmaßen und -methoden beschäftigen können, müssen wir zunächst den Begriff *Privacy* definieren. Anschließend diskutieren wir noch mögliche Privacy-Bedrohungen wie *Identity Disclosure*, *Membership Disclosure* und *Attribute Disclosure*.

Der Privacy-Begriff *Privacy*, zu Deutsch *Datensicherheit*, beschreibt in der Regel den Schutz personenbezogener Daten oder die Vermeidung der Re-Identifizierung einer einzelnen Person aus einem öffentlichen Datensatz. Hierfür existieren beispielsweise die europäische *Datenschutz-Grundverordnung*[3] (DSGVO) sowie das deutsche *Bundesdatenschutzgesetz*[4] (BDSG). Neben dem Datenschutz unterscheiden wir in Deutschland neben weiteren Privacy-Varianten noch das *allgemeine Persönlichkeitsrecht* — ein Grundrecht, welches dem Schutz der Persönlichkeit einer Person vor Eingriffen in ihren Lebens- und Freiheitsbereich dient (Quelle: Wikipedia, 13.07.2022) — und die *Datensicherheit*, d.h. der technischen Umsetzung zur Einhaltung des Datenschutzes. Der Privacy-Begriff ist somit nicht eindeutig definiert. Unsere Definition von Privacy ist in Abschnitt 8.1 zu finden.

[3] DSGVO: `https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:02016R0679-20160504&from=EN`
[4] BDSG: `https://www.gesetze-im-internet.de/bdsg_2018/BJNR209710017.html`

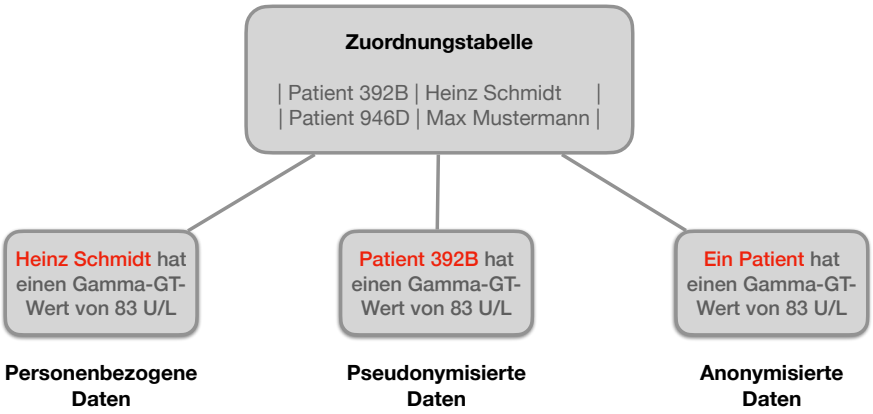


Abbildung 3.8. Pseudnoymisierung und Anonymisierung von Daten am konkreten Beispiel, nach [Ins20].

Die Einhaltung von Privacy-Aspekten ist dabei nicht immer leicht. Problematisch sind insbesondere nicht-anonymisierte personenbezogene Daten wie etwa der Name einer Person (siehe Abbildung 3.8). Diese können jedoch mittels Anonymisierung (siehe Definition 3.39) so verändert oder gelöscht werden, sodass eine Re-Identifizierung der Person nicht mehr möglich ist. In unserem Beispiel wird aus „Heinz Schmidt“ dann „ein Patient“. Ist eine Anonymisierung allerdings nicht gewünscht oder möglich, kann „Heinz Schmidt“ stattdessen zu „Patient 392B“ pseudonymisiert (siehe Definition 3.40) werden. In diesem Falle ist eine Re-Identifizierung nur mit Hilfe zusätzlicher Informationen möglich. Ein Beispiel hierzu liefert etwa Sweeney in [Swe02a]. Sie identifizierte bereits 2002 den Gouverneur von Massachusetts im Verzeichnis der örtlichen Krankenversicherung, indem sie diesen anonymisierten Datensatz – dieser enthielt Informationen über 135000 Patientinnen und Patienten – mit dem Wählerverzeichnis des US-Bundesstaat Massachusetts kombinierte.

Definition 3.39 (Anonymisierung, §3 Absatz 6 BDSG)**.** *Anonymisieren* ist das Verändern personenbezogener Daten derart, dass die Einzelangaben über persönliche oder sachliche Verhältnisse nicht mehr oder nur mit einem unverhältnismäßig großen Aufwand an Zeit, Kosten und Arbeitskraft einer bestimmten oder bestimmbaren natürlichen Person zugeordnet werden können. ■

Definition 3.40 (Pseudonymisierung, §4 Absatz 5 DSGVO)**.** *Pseudonymisierung* die Verarbeitung personenbezogener Daten in einer Weise, dass die personenbezogenen Daten ohne Hinzuziehung zusätzlicher Informationen nicht mehr einer spezifischen betroffenen Person zugeordnet werden können, sofern diese zusätzlichen Informationen gesondert aufbewahrt werden und technischen und organisatorischen Maßnahmen unterliegen, die gewährleisten, dass die personenbezogenen Daten nicht einer identifizierten oder identifizierbaren natürlichen Person zugewiesen werden. ■

Bedrohungen Nach [GLS14] beziehen sich Privacy-Bedrohungen stets auf drei verschiedene Arten von Attributen: direkte Identifikatoren, Quasi-Identifikatoren und sensible Attribute. *Identifizierende Attribute* ermöglichen eine eindeutige Re-Identifizierung von Personen beispielsweise durch den Namen oder eine eindeutige ID (siehe pinke Attribute in Tabelle 3.7). Sie sind in jedem Fall vollständig zu entfernen.

studentID	lastName	firstName	birthday	zipcode	district	grade
1	Moore	Donald	07.08.1988	18057	Rostock-Südstadt	2.3
2	Morgan	Sarah	01.01.2001	18059	Rostock-Südstadt	1.3
3	Wood	Jack	01.01.2001	18055	Rostock-Stadtmitte	2.0
4	Harrison	Elisabeth	22.10.1994	21039	Hamburg-Bergedorf	2.3
5	Williams	John	22.10.1994	21033	Hamburg-Bergedorf	2.7
6	William	Mary	12.02.1997	22765	Hamburg-Altona	1.7
7	Smith	Jack	23.07.2000	20359	Hamburg-Altona	2.0
8	John	Jennifer	01.06.1998	20359	Hamburg-Altona	η

Tabelle 3.7. Beispiel für die Unterscheidung von identifizierenden (pink hervorgehoben), nicht-sensiblen (gelb hervorgehoben) und sensiblen Attributen (grün hervorgehoben); vor der Anonymisierung.

Sensible Attribute sind schließlich alle Arten von besonders schützenswerten Daten wie diagnostizierte Krankheiten, Noten oder andere sehr persönliche Eigenschaften (siehe grünes Attribut). Da das sensible Attribut in der Regeln von besonderem Interesse ist, kann es nicht immer anonymisiert werden. In diesem Fall müssen wir den Rest des Datensatzes angemessen anonymisieren oder das sensible Attribut unterdrücken. Mehr hierzu in Abschnitt 3.6.2

Definition 3.41 (Quasi-Identifikator, [PS17])**.** Ein *Quasi-Identifikator (Quasi-Identifikator (QI))* ist eine Attributkombination, welche in Verbindung mit extern verfügbaren Informationen die eindeutige Identifikation eines Individuums ermöglicht. ■

Definition 3.42 (Äquivalenzklasse bzgl. Quasi-Identifikatoren, [NAC07])**.** Die Äquivalenzklasse QI^* eines Tupels t in einem Datensatz D^* ist die Menge aller Tupel in D^*, die denselben Quasi-Identifikator besitzen. ■

Nicht-sensible Attribute sind Attribute, die weder direkt identifizierend noch sensibel sind, aber einen indirekten Schluss auf eine natürliche Person ermöglichen können. Hierzu gehören beispielsweise deomographische Daten wie das Alter, Geburtsdatum oder Geschlecht einer Person (siehe gelbe Attribute). Eine Menge solcher nichtsensiblen Attribute, welche in Kombination zur Identifikation einer Person führen können, nennen wir auch *Quasi-Identifikator* (siehe Definition 3.41). So ist weder `birthday` noch `zipcode` in Tabelle 3.7 identifizierend, ihre Kombination im Datensatz jedoch selten genug, um für einen Großteil der Tupel als Schlüssel zu dienen. Die Tupel, welche den gleichen Quasi-Identifikator besitzen und daher nicht eindeutig identifizierend sind, werden in einer Äquivalenzklasse (siehe Definition 3.42) zusammengefasst.

Auf dieser Grundlage unterscheiden die Autoren insbesondere drei verschiedene Privacy-Bedrohungen:

- **Identity Disclosure** [Swe02b; XT06]: Ein Datensatz oder Tupel kann trotz Anonymisierung eindeutig einer Person zugeordnet werden.
- **Membership Disclosure** [NAC07]: Ein Angreifer kann mit hoher Wahrscheinlichkeit darauf schließen, dass der Datensatz einer Person in den veröffentlichten Daten enthalten ist.
- **Attribute Disclosure** [Mac+06]: Eine Person kann mit Informationen über ihre sensiblen Eigenschaften in Verbindung gebracht wird. Dies kann der konkrete Wert des sensiblen Attributes oder ein Wertebereich sein, welcher den sensiblen Wert enthält.

Der Artikel [GLS14] zeigt zudem auf, welche Privacy-Modelle zum Schutz der verschiedenen Angriffe genutzt werden können. Hierzu gehören unter anderem *k-Anonymität*, *l-Diversität*, *t-Closeness*, *k-Map* und *δ-Presence*, welche wir uns im Folgenden genauer anschauen werden.

3.6.1. Anonymisierungsmaße

Wir beginnen mit der k-Anonymität und ihren Erweiterungen k-Map, l-Diversität, t-Closeness, δ-Presence. Anschließend beschäftigen wir uns noch mit der Differential Privacy.

k-Anonymität und ihre Erweiterungen Um die oben genannten Bedrohungen zu klassifizieren, unterscheidet die Literatur verschiedene Anonymisierungsmaße. Das populärste Maß zum Schutz vor Identity Disclosure ist die k-Anonymität [Sam01], aber auch k-Map [Swe01] kann an dieser Stelle verwendet werden. Für die Vermeidung von Membership Disclosure entwickelten Nergiz et al. unter anderem die δ-Presence [NAC07]. Die l-Diversität [Mac+06] kann schließlich zur Lösung der Attribute Disclosure verwendet werden. Weitere Anonymisierungsmaße können in [GLS14] nachgelesen werden.

Definition 3.43 (k-Anonymität, [Sam01])**.** Sei r_i eine Basisrelation und QI ein zugehöriger Quasi-Identifikator. Dann ist r_i *k-anonym* bzgl. QI, genau dann, wenn jede Folge von Werten in $\pi_{QI}(r_i)$ mindestens k mal in $\pi_{QI}(r_i)$ vorkommt. ■

Ziel der k-Anonymität (siehe Definition 3.43)ist es, jede Quasi-Identifikator-Äquivalenzklasse (QI*) soweit zu anonymisieren, dass sie mindestens k identische Werte besitzt. Bei der l-Diversität (siehe Definition 3.44) wird zusätzlich die Anzahl verschiedener Attributwerte des sensiblen Attributs innerhalb der QI*-Blöcke gemessen. Weder k-Anonymität noch l-Diversität können jedoch Attribute Disclousure verhindern. Als Erweiterung der l-Diversität entwickelte sich hieraus die t-Closeness (siehe Definition 3.45). Sie verlangt, dass die Verteilung eines sensiblen Attributs in einer beliebigen Äquivalenzklasse nahe an der Verteilung des Attributs in der Gesamttabelle liegt [LLV07]. Mit anderen Worten, der Abstand zwischen den beiden Verteilungen ist nicht größer als ein vorher definierter Schwellwert t.

Definition 3.44 (l-Diversität, [Mac+06])**.** Eine k-anonyme Relation erfüllt l-Diversität, wenn der Wertebereich eines sensiblen Attributs pro Äquivalenzklasse mindestens l verschiedene Werte umfasst. ■

Definition 3.45 (t-Closeness, [LLV07])**.** Eine Äquivalenzklasse gilt als *t-close*, wenn der Abstand zwischen der Verteilung eines sensiblen Attributs in dieser Klasse und der Verteilung des Attributs in der gesamten Tabelle nicht größer ist als ein Schwellenwert t. Eine Tabelle gilt als *t-close*, wenn alle Äquivalenzklassen t-close sind. ■

Neben den drei klassischen Anonymisierungsmaßen k-Anonymität, l-Diversität und t-Closeness existieren in der Literatur (beispielsweise in [GLS14]) noch diverse weitere Maße. Hierzu gehören unter anderem k-Map sowie δ-Presence. Während k-Anonymität die Anzahl der Tupel innerhalb eines Datensaztes zählt, bestimmt k-Map die Anzahl der Tupel innerhalb der größtmöglichen Grundgesamtheit. Im Fall unserer Studierenden beziehen wir uns bei unseren Berechnungen also nicht mehr auf die Studierenden der Universität Rostock, sondern auf alle Studierenden weltweit, was in der Realität jedoch nur schwer umzusetzen ist. So sagen wir, dass ein Datensatz k-Map genügt (siehe Definition 3.46), wenn jede Kombination von Werten für die Quasi-Identifikatoren mindestens k-mal im Re-Identifizierungsdatensatz vorkommt. Dabei wird nicht nur sichergestellt, dass jede Äquivalenzklasse hinsichtlich eines Quasi-Identifikators mindestens k nicht voneinander unterscheidbare Tupel enthält, sondern auch, dass es in der Grundgesamtheit aller Individuen mindestens k Individuen gibt, welche im Datensatz repräsentiert werden.

Definition 3.46 (k-Map, [ED08; Swe01])**.** Sei U eine k-anonyme Datenbank und d eine hieraus erzeugte Teil-Datenbank. Dann genügt d der Eigenschaft *k-Map*, wenn jedes Tupel aus d mindestens zu k Tupeln aus U ähnlich ist. Das heißt: Jedes Tupel der veröffentlichten Teil-Datenbank d muss sich unmissverständlich auf mindestens k Tupel aus U beziehen. ■

Die δ-Presence hingegen (siehe Definition 3.47) gibt einen Bereich an akzeptablen Wahrscheinlichkeit an, welcher beschreibt, dass ein Tupel des generalisierten, privaten Datensatzes D^* auch im öffentlichen Datensatz P enthalten ist. Sie stellt somit den Quotienten aus k-Anonymität und k-Map dar und gibt das Verhältnis zwischen den Individuen im anonymisierten Datensatz und den Individuen in der Grundgesamtheit an.

Definition 3.47 (δ-Presence, [NAC07])**.** Sei P ein extern verfügbarer, öffentlicher Datensatz P. Sei weiter T eine private, zu anonymisierende Datensatz. Dann gilt die δ-Presence $\delta = (\delta_{\min}, \delta_{\max})$ für eine Generalisierung T^* genau dann, wenn

$$\forall T \in P : \delta_{\min} \leq \mathbb{P}(t \in T \mid T^*) \leq \delta_{\max}.$$

■

Für die Vermeidung von Identity Disclosure wird jede Äquivalenzklasse von Quasi-Identifikatoren (QI*) so anonymisiert, dass sie mindestens k identische Werte besitzt. Jedes anonymisierte Tupel einer QI-Äquivalenzklasse setzt also $k-1$ weitere, nicht von diesem Tupel unterscheidbare Tupel voraus (siehe Definition 3.43). Die Tupel des k-anonymen Datensatzes ähnlichen sich anschließend so sehr, dass die Re-Indentifizierung eines Individuums nicht mehr möglich ist. Hierfür kommen insbesondere die Mikroaggregation [DM02], die Generalisierung [Sam01] und die Unterdrückung [Sam01] zum Einsatz.

Techniken zur Vermeidung von Attribute Discosure regeln insbesondere die Verbindung zwischen den Quasi-Identifikatoren und den sensiblen Attributen. Dafür werden zunächst anonyme Gruppen gebildet, welche anschließend iterativ zusammengeführt werden, bis die Verbindung gemäß einem gegebenen Anonymisierungsmaß geschützt

ist. Auch hier können die Generalisierung [Sam01] oder Unterdrückung [Sam01] genutzt werden. Kombinationen sind selbstverständlich auch möglich.

Die Behandlung von Membership Disclosure hingegen ist deutlich aufwendiger als Generalisierung oder Unterdrückung. Wir vernachlässigen sie daher an dieser Stelle. Details hierzu können unter anderem in [NAC07; NC10] nachgelesen werden.

Differential Privacy Eines der bekanntesten semantischen Modelle zur Realisierung von Privacy ist die *Differential Privacy* [Dwo06; Dwo08]. Die Idee hinter dieser Methode besteht darin, dass statistische Datensätze so mit einem zufälligen Rauschen versehen werden, dass zwar Aussagen über die allgemeine Grundmenge, jedoch nicht über einzelne Individuen getroffen werden können. Eine Anfrage Q wird daher nicht direkt an die Datenbank D, sondern an eine randomisierte Anfragefunktion $\mathcal{K}$ (siehe Definition 3.48) gestellt. Diese bestimmt zunächst $Q(D)$ und verrauscht anschließend das Ergebnis so, dass das Einfügen oder Löschen eines einzelnen Tupels das Anfrageergebnis nicht signifikant verändert. Die Analyse des verrauschten Datensatzes muss also unempfindlich gegenüber dem Einfügen oder Löschen eines Datensatzes in bzw. aus dem ursprünglichen Datensatz sein. Nachteilig ist jedoch, dass die Durchsetzung von Differential Privacy nur die Freigabe verrauschter Statistiken ermöglicht. Die oben beschriebenen Bedrohungen können somit nicht gänzlich ausgemerzt werden. So wird beispielsweise in [Cor11] gezeigt, wie man mit speziellen Klassifizierungsalgorithmen dennoch auf sensible personenbezogene Daten schließen kann. Im Laufe der Zeit haben sich daher zahlreiche Erweiterungen entwickelt. Differential Privacy bildet zudem die Grundlage für einige weitere semantische Modelle. Diese können beispielsweise in [Vim+12] nachgelesen werden.

Definition 3.48 (Randomisierte Anfragefunktion, [Dwo06])**.** Eine randomisierte Anfragefunktion $\mathcal{K}$ gewährleistet ϵ-Differential-Privacy, wenn für alle Datensätze D_1, D_2, die sich in höchstens einem Element unterscheiden, sowie für alle $S \subseteq W(\mathcal{K})$ gilt:

$$\mathbb{P}[\mathcal{K}(D_1) \in S] \leq e^{\epsilon} \cdot \mathbb{P}[\mathcal{K}(D_2) \in S].$$

■

Bei der Differential Privacy wird also bestimmt, wie groß der Unterschied zwischen den beiden Datensätzen D_1 und D_2 ist. Die Wahl des Parameters $\epsilon \geq 0$ ist dabei entscheidend für die Informationsweitergabe. Es gilt: „Je kleiner ϵ gewählt wird, desto weniger Informationen werden preisgegeben, aber desto schwieriger ist es auch, die Daten anschließend sinnvoll zu analysieren“ (Quelle: Bundesministerium für Wirtschaft und Energie, 20.08.2022). Mit anderen Worten: „Je größer ϵ ist, desto schwächer ist die Garantie für den Datenschutz “ (Quelle: [PS17]). Der Extremfall $\epsilon = 0$ garantiert somit die perfekte Geheimhaltung, macht den herausgegebenen Datensatz aufgrund der großen Verrauschung jedoch unbrauchbar. In manchen Fällen kann aber auch ein $\epsilon > 1$ sinnvoll sein [PS17]. Die Wahl des Parameters ϵ ist somit entscheidend für die Anwendbarkeit von Differential Privacy. Zudem sollten alle Einträge der beiden Datensätze D_1 und D_2 unabhängig voneinander sein.

3.6.2. Anonymisierungsmethoden

Als Anonymisierungsmethode betrachten wir die Generalisierung, die Unterdrückung von Tupeln und/oder Spalten, die Permutation, intensionale Antworten sowie Slicing und das Verrauschen von Daten. Wir starten mit der Generalisierung.

Generalisieren Unser erster Ansatz zur Bereitstellung von k-Anonymität, die *Generalisierung*, basiert auf Verallgemeinerung von Attributwerten. Eine minimale Generalisierung, „bei der möglichst viele Informationen erhalten bleiben, ist wahrscheinlich NP-schwer“ [PS17], da sie abhängig vom Attribut für einen konkreten Attributwert oder aber auf einer ganzen Spalte durchgeführt wird. Dabei werden die einzelnen Attributwerte verallgemeinert (siehe Definition 3.49), indem (numerische) Werte (soweit möglich) zu Intervallen zusammengefasst oder einzelne Zeichen durch ein $*$ maskiert werden. Dies wird attributweise so lange wiederholt bis ein gegebenes Anonymisierungsmaß (siehe Abschnitt 3.6.1) erfüllt ist. Grundlage hierfür sind stets Konzepthierarchien (siehe Definition 3.50) oder Domänengeneralisierungshierarchien (siehe Definition 3.51). Wir werden dies in Abschnitt 8.3 genauer untersuchen. Ein Beispiel hierfür sei vorweg jedoch in Abbildung 3.9 gegeben.

Definition 3.49 (Generalisierung eines Attributes, nach [Swe01])**.** Sei A ein gegebenes Attribut, dann heißt jede Funktion über A mit $f : A \to B$ *Generalisierung von* A. ■

Definition 3.50 (Konzepthierarchie, [Heb06])**.** Eine *Konzepthierarchie* $(C, \leq)$ ist definiert über eine nicht-leere, endliche Menge C sowie eine partielle Ordnung $\leq$ auf C, wobei $\top$ das bzgl. $\leq$ kleinste und $\bot$ das bzgl. $\leq$ größte Element ist. ■

Definition 3.51 (Domänengeneralisierungshierarchien, nach [Swe01])**.** Sei A ein gegebenes Attribut, dann heißt eine Folge von Funktionen $f_i : A_i \to A_{i+1}$ mit $A_0 \xrightarrow{f_0} A_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{n-1}} A_n$, $A = A_0$ und $|\, A_n \,| = 1$ *Domänengeneralisierungshierarchie von* A. ■

Bezogen auf unser Studierenden-Beispiel aus Tabelle 3.7 ergibt sich folgende Situation: Das Schema besteht aus den sieben Attributen `studentID`, `lastName`, `firstName`, `birthday`, `zipcode`, `distinct` und `grade`. Die ersten drei Attribute sind identifizierend. Sie werden daher bei der Anonymisierung mittels Unterdrückung gestrichen. Gleiches gilt für das sensible Attribut `grade`. Auch dieses wird zunächst unterdrückt. Zu generalisieren sind somit lediglich die drei nicht-sensible Attribute `birthday`, `zipcode` und `district`. Hierfür generalisieren wir zunächst den `zipcode` durch das Einfügen von $*$ (von rechts beginnend) so, dass die Postleitzahlen der einzelnen Stadtteile möglichst allgemein sind. Anschließend verallgemeinern wir die Geburtsdaten auf ihre Jahreszahlen und fassen diesen in Intervallen so zusammen, dass auch das Attribut `birthday` pro Stadtteil nur einen Wert enthält. Wir erhalten so einen 1-anonymen Datensatz (siehe Tabelle 3.8a). Generalisieren wir in einem zweiten Schritt zusätzlich noch die Stadtteile, ist der generalisierte Datensatz sogar 3-divers (siehe Tabelle 3.8b).

birthday	zipcode	district	grade
1988-2001	1805∗	Rostock-Südstadt	2.3
1988-2001	1805∗	Rostock-Südstadt	1.3
01.01.2001	18055	Rostock-Stadtmitte	2.0
22.10.1994	2103∗	Hamburg-Bergedorf	2.3
22.10.1994	2103∗	Hamburg-Bergedorf	2.7
1997-2000	2 ∗ ∗ ∗ ∗	Hamburg-Altona	1.7
1997-2000	2 ∗ ∗ ∗ ∗	Hamburg-Altona	2.0
1997-2000	2 ∗ ∗ ∗ ∗	Hamburg-Altona	η

(a) 1-Anonymität

birthday	zipcode	district	grade
1988-2001	1805∗	Rostock	2.3
1988-2001	1805∗	Rostock	1.3
1988-2001	1805∗	Rostock	2.0
1994-2000	2 ∗ ∗ ∗ ∗	Hamburg	2.3
1994-2000	2 ∗ ∗ ∗ ∗	Hamburg	2.7
1997-2000	2 ∗ ∗ ∗ ∗	Hamburg	1.7
1997-2000	2 ∗ ∗ ∗ ∗	Hamburg	2.0
1997-2000	2 ∗ ∗ ∗ ∗	Hamburg	η

(b) 3-Diversität

Tabelle 3.8. Beispiel für die Unterscheidung von nicht-sensiblen (gelb hervorgehoben) und sensiblen Attributen (grün hervorgehoben); nach der Anonymisierung.

Wie bereits in unserem Beispiel angesprochen, wird die Generalisierung in der Regel mit dem *Unterdrücken* von Spalten oder Zeilen kombiniert. Hierbei werden einzelne, identifizierende Tupel oder ganze Attribute aus einer gegebenen Relation entfernt. Dies ist insbesondere bei Tupeln der Fall, die sich keiner Äquivalenzklasse zuordnen lassen sowie bei identifizierenden Attributen wie beispielsweise Schlüsseln.

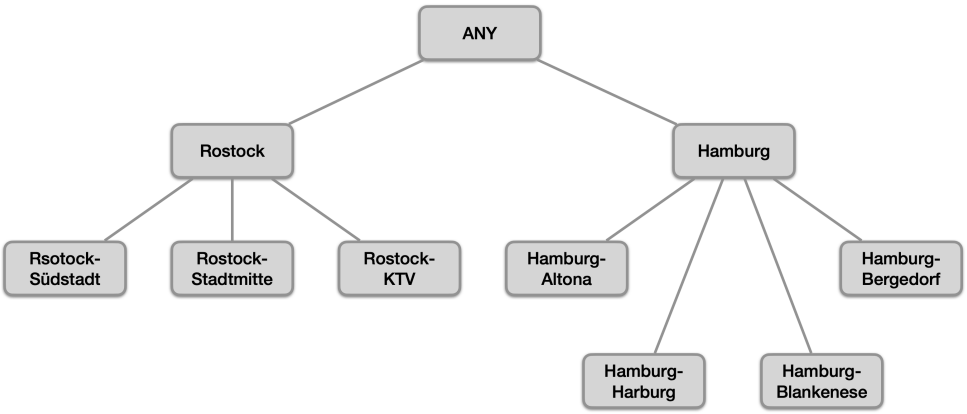


Abbildung 3.9. Domänenegeneralisierungshierarchie für das Attribut `district`.

Unterdrückung von Tupeln und Spalten Auch das Unterdrücken von Tupeln oder Spalten zählt zu den bekanntesten Anonymisierungsmethoden. So werden bei der Anonymisierung Schlüsselattribute unterdrückt (siehe die Attribute `studentID`, `lastName` und `firstName` aus Tabelle 3.7) und Quasi-Identifikatoren (siehe die Attribute `birthday`, `zipcode` und `distinct` aus Tabelle 3.7) soweit wie möglich generalisiert. Ist eine Generalisierung irgendwann nicht mehr möglich, wird der entsprechende Attributwert durch den Defaultwert `ANY` ersetzt. Dies entspricht der „Unterdrückung eines Attributwertes". Das vollständige Unterdrücken der zugehörigen Spalte bzw. des Tupels ist in der Regel nicht notwendig.

Um k-Anonymität oder l-Diversität zu gewährleisten, können zudem ganze Tupel aus der Relation entfernt werden. Dies ist immer dann der Fall, wenn diese eindeutig identifizierbar sind. Der hieraus resultierende Informationsverlust kann unter Umständen geringer sein als der Informationsverlust, welcher durch eine mehrstufige Generalisierung entsteht. In unserem Studierendenbeispiel aus Tabelle 3.8a bedeutet dies, dass das Tupel $(01.01.2001, 18055, \text{Rostock-Stadmitte})$ generalisiert oder unterdrückt werden kann, um mit den anderen beiden Rostock-Tupeln zu einer Äquivalenzklasse zu „verschmelzen". In diesem konkreten Beispiel haben wir uns für eine Generalisierung entschieden.

Permutation Um Unsorted-Matching-Angriffe zu vermeiden, hilft es, die Daten zu permutiert. Wie in [PS17] beschrieben, haben die Tupel einer relationalen Datenbank keine Reihenfolge. In der Praxis wird die Reihenfolge der Tupel aus pragmatischen Gründen oft beibehalten. Durch das Verknüpfen zweier Tabellen kann es so zur Offenlegung persönlicher Daten kommen.

Intensionale Antworten Eine *intensionale Antwort* ist nach [Mot94] eine „Ergänzung der konventionellen Antwort, die entweder eine knappe Beschreibung der Antwort oder verschiedene nützliche Aussagen zur Antwort enthält". Als Beispiel nennt Motro etwa eine Datenbankanfrage. Sie selbst ist eine intensionale Aussage und ihre Antwort eine Erweiterung dieser intensionalen Antwort, welche jedoch vollständig aus extensionalen Informationen besteht. Eine Datenbankanfrage hat somit stets einen intensionalen als auch extensionalen Charakter. Basierend auf diesen Erläuterungen definieren wir extensionale und intensionale Antworten wie folgt:

Definition 3.52 (Extensionale und intensionale Antwort)**.** Gegeben seien eine Datenbank D sowie eine Anfrage Q. Sei weiter $Q(I)$ das zugehörige Anfrageergebnis. Die *extensionale Antwort* zu $Q(I)$ entspricht der Angabe der konkreten Ergebnistupel. Die *intensionale Antwort* liefert stattdessen eine allgemeine Beschreigung des Anfrageergenisses $Q(I)$, nicht jedoch die Daten selbst. ■

Intensionale Antworten sind nach [Mot94] eine Ergänzung extensionaler Antworten, welche das Ergebnis zusätzlich verdeutlichen und den Informationsgehalt des Gesamtergebnisses erhöhen können. Motro ist daher einer der Ersten, welcher die existierenden Definition klassifiziert und abstrahiert. Weitere Details sind unter anderem in [Mot94] zu finden.

Im weiteren Verlauf der Arbeit beziehen wir intensionale Antworten stets auf Provenance-Anfragen. Diese können zum Einen extensional durch Angabe der Tupel oder Tupel-Identifikatoren beantwortet werden. Zum anderen ist eine *intensionale Provenance-Antwort* möglich. In diesem Fall werden die Daten abstrahiert und allgemein beschrieben. Dies kann beispielsweise textuell oder durch Generalisierung konkreter Attributwerte geschehen. So können die Abschlussnoten in unserem Beispiel etwa auf die Notenbereiche $[1.0, 1.5]$, $[1.6, 2.5]$ usw. verallgemeinert werden. Während die extensionale Antwort der Tabelle 3.1 entspricht, ist ihre Generalisierung bereits eine intensionale Antwort. Möglich wäre hier auch eine textuelle Beschreibung wie „Die Datenbank enthält vier Studierende aus Hamburg mit befriedigenden Prüfungsergebnissen sowie drei Studierende aus Rostock mit guten Ergebnissen. Die Rostocker Studierenden sind durchschnittlich jünger als die Hamburger Studierenden."

Ein Ansatz zur Generalisierung intensionaler Antworten wird in [PY99] beschrieben. Er besteht aus drei Phasen: Preprocessing, Query Execution and Answer Generation. Hierfür werden zunächst eine Reihe von Konzepthierarchien als Verallgemeinerung der gegebenen Datenbank sowie eine Reihe virtueller Hierarchien erzeugt. Anschließend wird die Anfrage ausgeführt und erweitert. In der Answer Generation-Phase werden anschließend die relevante Anfrage-Attribute mit Hilfe der vorher bestimmten Hierarchien verallgemeinert. Die Wahl der relevanten Attribute ist dabei dem Nutzer selbst überlassen. Weitere Details sind in [PY99] zu finden.

Verrauschen der Daten Grundlage für das Einhalten von Differential Privacy ist das *Verrauschen von Daten*. Hierfür werden diese mit einem Faktor multipliziert oder durch eine spezielle Funktion verrauscht. Auch das Hinzufügen oder Löschen von Tupeln sind denkbar.

Slicing Die Anonymisierungsmethode *Slicing* basiert ebenso wie die Permutation auf dem Prinzip der Vertauschen. Hier werden jedoch statt nur einzelne Tupeln ganze Buckets sowie Attributmengen vertauscht. Kurz gesagt, *Slicing* beschreibt die Partitionierung Attribute oder Tupel einer gegebenen Tabelle. Details zur Anonymisierung mittels Slicing sowie Beispiele können in [Li+12] oder [GH15] nachgelesen werden.

Nachdem wir nun alle für das Verständnis der vorliegenden Arbeit notwendigen Grundlagen definiert haben, untersuchen wir als nächstes den Stand der Technik im Bereich Provenance- sowie Chase-Systeme (siehe Kapitel 4). Anschließend formalisieren wir unsere Problemstellung (siehe Kapitel 5) und stellen die Ergebnisse der Dissertation vor (siehe Kapitel 6 bis 10).

4. Analyse verschiedener CHASE- und Provenance-Systeme

Ziel der vorliegenden Arbeit ist die Entwicklung eines Konzeptes zur Vereinigung von Provenance, Evolution und Privacy zur Konstruktion einer (minimalen) Teil-Datenbank, welche den Bedingungen (I.), (II.) und (III.) aus Abschnitt 1.1 bzw. 2.1 genügt. Das zugehörige Konzept wird in den Kapiteln 5 bis 10 vorgestellt und abschließend implementiert werden. An dieser Stelle wollen wir einen Überblick über bereits existierende Chase-Systeme (siehe Abschnitt 4.1) sowie die bekanntesten Provenance-Systeme (siehe Abschnitt 4.2) geben, welche die Basis für die während der Dissertation entstandenen Systeme ChaTEAU (siehe Abschnitt 9.2) und ProSA (siehe Abschnitt 10.9) bilden werden.

4.1. Analyse verschiedener Chase-Systeme

Systeme, welche die in [Ben+17; GMS12] vorgestellten Chase-Varianten implementieren, sind aktuell noch recht rar gesät. So wurden unseres Wissens nach seit 2009 lediglich ein halbes Dutzend Systeme entwickelt, welche eine Version des Chase- bzw. des Chase&Backchase implementieren. Viele der Systeme wurden als Demonstration auf der VLDB[1] vorgestellt. Jedes Tool hat zudem eine eigene Projekt-Website, auf welcher der Quellcode sowie die wichtigsten Artikel hinterlegt bzw. verlinkt sind. Eine Liste der Websites ist am Ende des Dokuments in Teil V zu finden.

Die Anwendung des Chase auf Instanzen [Ben+17] und Anfragen [DPT99] verhält sich ähnlich, da die Struktur von Anfragen und Instanzen ähnlich ist. Bestehende Chase-Systeme wie PDQ [BLT14], Llunatic [Gee+20] oder Graal [BLT15] sind jedoch auf bestimmte Anwendungsfälle wie die semantische Optimierung, Data Cleaning, Data Exchange oder Query Rewriting beschränkt. Diese verschiedenen Anwendungsfälle können jedoch auf die Verarbeitung von Instanzen bzw. Anfragen reduziert werden. Mit ChaTEAU (siehe Kapitel 9) haben wir ein universales Chase-System entwickelt und implementiert, welches Instanzen und Anfragen auf ein allgemeines Chase-Objekt und einen allgemeinen Chase-Parameter abstrahiert. Grundlage für die Entwicklung unseres eigenen Systems ist jedoch die Analyse der bekanntesten bereits existierenden Systeme.

4.1.1. Auswahl der Chase-Systeme

Basierend auf [Ben+17] ergeben sich sechs Systeme, welche wir genauer untersuchen wollen: ChaseFUN [BIL16], DEMo [PS09], Graal, Llunatic, PDQ und Pegasus [Mei14]. Die dort erwähnten Chase-verwandten Systeme wie beispielsweise DLV [Leo+06] lassen wir bei unseren Analysen aus. Dies würde den vorgesehen Rahmen sprengen. Nach einer erweiterten Literaturstudie entscheiden wir uns zudem dazu auch ChaseTEQ [Spe11] und $\text{Prov}_{\text{C\&B}}$ [DPT99] genauer zu untersuchen.

Neben der Beschreibung in der Literatur und der Auffindbarkeit des Quellcodes, spielt auch der Anwendungsfall der Systeme bei der Analyse eine Rolle. So ist ChaseTEQ das älteste von uns praktisch getestete Chase-System. Es stellt insbesondere die Chase-Terminierung in den Vordergrund und verarbeitet ebenso wie Llunatic und DEMo den Chase auf Instanzen. $\text{Prov}_{\text{C\&B}}$ hingegen implementiert ebenso wie Pegasus und PDQ eine Version des Chase&Backchase. Zusätzlich kombiniert $\text{Prov}_{\text{C\&B}}$ den Algorithmus mit weiteren Provenance-Informationen. Es ist daher sowohl als Chase- als auch als Provenance-System interessant. Alle drei Systeme verarbeiten den Chase auf Anfragen. Auch Graal verarbeitet Anfragen, jedoch mit einer abgewandelten Version des Chase&Backchase, dem sogenannten *forward and backward chaining*. Insgesamt erhalten wir so die in Abbildung 4.1 zusammengefasste Auswahl an Chase-Systemen, welche wir im weiteren Verlauf dieses Abschnitts in einer ausführlichen Literaturrecherche sowie einigen praktischen Tests (rot hervorgehoben) untersuchen werden. Dass nicht alle hier genannten

[1] VLDB: http://vldb.org

Systeme praktisch getestet werden, liegt insbesondere daran, dass wir nicht bei jedem System Zugang zum Quellcode haben und uns die Installation nicht immer fehlerfrei möglich ist.

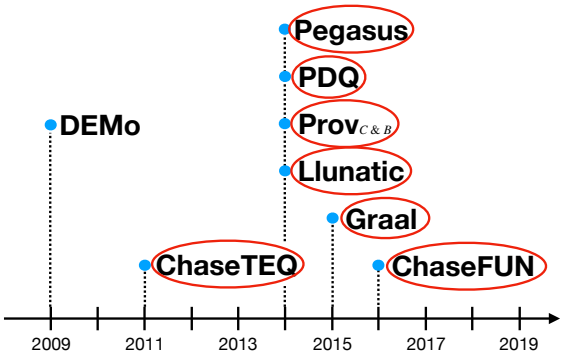


Abbildung 4.1. Auswahl der untersuchten Chase-Systeme; die rot gekennzeichnet Systeme wurden praktisch getestet.

4.1.2. Ergebnisse der praktischen Tests

Für sieben Systeme war es uns möglich den Code bzw. ein lauffähiges Programm zu erhalten (rot hervorgehoben in Abbildung 4.1), von denen wir fünf — ChaseTEQ, PDQ, $\text{Prov}_{\text{C\&B}}$, Llunatic und Graal — praktisch testen konnten. Sie stehen in ihrer aktuellen Version als virtuelle Maschine oder zip-Datei unter `https://nextcloud.informatik.uni-rostock.de/index.php/s/SNXPC8BaxR9i9z5` zur Verfügung. Peagsus und ChaseFUN konnten wir aus technischen Gründen leider nicht testen. Bei DEMo fehlte uns zudem der Quellcode bzw. das Programm selbst, sodass auch hier ein praktischer Test nicht möglich war.

Ziel unserer praktischen Tests ist die Überprüfung der in der Literatur angegebenen Funktionalitäten. Zunächst gilt es daher den jeweiligen in der Literatur angekündigten Anwendungsfall zu verifizieren:

- Llunatic, DEMo und ChaseTEQ implementieren den Chase auf Instanzen,
- Pegasus, PDQ, $\text{Prov}_{\text{C\&B}}$ sowie ChaseFUN implementieren den Chase auf Anfragen,
- Graal implementiert einen Chase-ähnlichen Algorithmus und arbeitet ebenfalls auf Anfragen.

Nachweisen können wir dies, wie in Tabelle 4.1 gezeigt für ChaseTEQ, PDQ, Llunatic und Graal. Unsere praktischen Tests widersprechen hier nicht den Angaben der Literatur. Zu erwähnen ist, dass keines der getesteten Systeme mehr als drei Anwendungsgebiete vorhält.

Hinweis: Da ChaseFUN und Pegasus nicht fehlerfrei aufgesetzt und die Nutzung von $\text{Prov}_{\text{C\&B}}$ nicht abschließend geklärt werden konnte, entfallen alle drei Systeme bei den folgenden Betrachtungen. Gleiches gilt für DEMo, da uns hier der Quellcode nicht zur Verfügung stand. Wir kennzeichnen dies durch (—).

Anwendungsfall	**DEMo**	**ChaseTEQ**	**Pegasus**	**PDQ**	**$\text{Prov}_{\text{C\&B}}$**	**Llunatic**	**Graal**	**ChaseFUN**
Semantic Optimization	—	✗	—	✓	—	✗	(✓)	—
Query Rewriting	—	✗	—	✓	—	✗	✓	—
Data Exchange	—	✗	—	✗	—	✓	✓	—
Data Cleaning / Data Repairing	—	✓	—	✗	—	✓	✗	—
Termination Test	—	✓	—	✗	—	✓	✗	—

Tabelle 4.1. Analyse-Ergebnisse der Tool-Tests. Legende: für den Anwendungsfall geeignet (✓), für den Anwendungsfall nicht geeignet (✗), praktischer Test nicht möglich (—).

In ProSA konzentrieren wir uns stets auf den Chase auf Instanzen. Llunatic und ChaseTEQ scheinen daher auf den ersten Blick besonders interessant zu sein. Um zu testen, ob diese Systeme zusätzlich auch Anfragen verarbeiten können, definieren wir uns sieben Benchmark-Anfragen an unser gegebenes Studierenden-Beispiel (siehe Anhang B, Anfragen B.5 bis B.7), welche im Sinne eines guten Forschungsdatenmanagements ebenfalls in unseren virtuellen Maschinen hinterlegt ist. Diese dienen als Basis zur Untersuchung der semantischen Optimierung. So

testet beispielsweise Anfrage QA_2 (siehe Anfrage 4.1), ob eine vorliegende Schlüssel-Fremdschlüssel-Beziehung erkannt und optimiert werden kann. Die Einbindung von Inklusions- und Verbundabhängigkeiten mit anschließender Optimierung testen wir anhand der Anfragen QA_1 und QA_3.

Schauen wir uns Anfrage QA_2 noch einmal im Detail an (siehe Anfrage 4.1). Die erste Zeile definiert das Schema $R(A_1, A_2, A_3)$ und die zweite Zeile die gegebene Abhängigkeit $\forall a_1, ..., a_5 : R(a_1, a_2, a_3) \wedge R(a_4, a_2, a_5) \rightarrow (a_3 = a_5)$, eine egd. Ziel des Beispiels ist die Optimierung der Anfrage $\pi_{A_1,A_2}(r(R)) \bowtie \pi_{A_2,A_3}(r(R))$ in der dritten Zeile zu $r(R)$ in der letzten Zeile.

Anfrage 4.1 Verbundtreue-Kriterium mit einer FD (QA_2)

`schema:` $R(A_1, A_2, A_3)$
`egd:` $\forall a_1, ..., a_5 : R(a_1, a_2, a_3) \wedge R(a_4, a_2, a_5) \rightarrow (a_3 = a_5)$
`query:` $\pi_{A_1,A_2}(r(R)) \bowtie \pi_{A_2,A_3}(r(R))$
`optimized query:` $r(R)$

Anfragen mit mehreren INDs als Kette von Schlüssel-Fremdschlüssel-Verbindungen sowie Multi-Attribut-Fremdschlüsseln sind durch die Benchmark-Anfragen B.8 bis B.11 vertreten. So entspricht die erste Benchmark-Anfrage QB_1 einer Anfrage mit mehreren INDs als Kette auf einem Attribut. Anfrage QB_2 arbeitet hingegen auf wechselnden Attributen und Anfrage QB_3 zusätzlich mit alternativen Schlüsseln. Die letzte Benchmark-Anfrage QB_4 testet exemplarisch die Kombination einer IND und einer FD (siehe auch Anfrage 4.2).

Anfrage 4.2 Multi-Attribut-Fremdschlüssel mit interner FD (QB_4)

`schema:` $R_1(A_1, A_2, A_3), R_2(A_4, A_5, A_6)$
`IND:` $R_2(A_4, A_5) \subseteq R_1(A_1, A_2)$
`FD:` $A_1 \rightarrow A_2$
`query:` $\pi_{A_4,A_5}(\pi_{A_4,A_5}(R_2) \bowtie \sigma_{A_6=5}(\pi_{A_4,A_6}(R_2)))$
`optimized query:` $\pi_{A_4,A_5}(\sigma_{A_6=5}(r(R_2)))$

Die Benchmark-Anfragen QA_1 und QA_3 werden von den vier CHASE-Systemen ChaseTEQ, PDQ, Llunatic und Graal (korrekt) verarbeitet (siehe Tabelle 4.2). Einfache konjunktive Anfragen ohne verkettete Bedingungen können somit problemlos optimiert werden, d.h. die Anzahl der Prädikate, speziell die Anzahl der Verbünde kann minimiert werden. Da ChaseTEQ jedoch keine egds verarbeiten kann und keinen BACKCHASE implementiert hat, entfallen hier die Anfragen QA_2 sowie QB_1 bis QB_4 (dargestellt durch ein Kreuz ($\mathcal{X}$)). Anders verhält es sich bei PDQ, Llunatic und Graal. Hier werden die Anfragen QB_1 bis QB_3 wie vorgesehen optimiert (dargestellt durch einen Hacken (✓)). Lediglich die Anfragen QA_2 und QB_4 liefern je nach System unterschiedliche Optimierungsergebnisse.

Benchmark-Anfrage	**DEMo**	**ChaseTEQ**	**Pegasus**	**PDQ**	**Prov$_{C\&B}$**	**Llunatic**	**Graal**	**ChaseFUN**
QA_1	—	✓	—	✓	—	✓	✓	—
QA_2	—	$\mathcal{X}$	—	✓	—	✓	✓/$\mathcal{X}$	—
QA_3	—	✓	—	✓	—	✓	✓	—
QB_1	—	$\mathcal{X}$	—	✓	—	✓	✓	—
QB_2	—	$\mathcal{X}$	—	✓	—	✓	✓	—
QB_3	—	$\mathcal{X}$	—	✓	—	✓	✓	—
QB_4	—	$\mathcal{X}$	—	$\mathcal{X}$	—	✓	✓/$\mathcal{X}$	—

Tabelle 4.2. Ergebnisse der Benchmark-Anfragen im Vergleich zu den erwarteten Ergebnissen aus Anhang B, basierend auf [Bre+20]. Legende: Benchmark-Ergebnis erhalten (✓), Benchmark-Ergebnis nicht erhalten ($\mathcal{X}$), praktischer Test nicht möglich (—).

4.1.3. Die Systeme im Detail

Betrachten wir die ausgewählten Systeme nun etwas genauer. Die Reihenfolge, in der wir die Systeme vorstellen, basiert auf ihrem in Abbildung 4.1 dargestellten Veröffentlichungsdatum, d.h. DEMo, ChaseTEQ, Pegasus, PDQ, $\text{Prov}_{\text{C\&B}}$, Llunatic, Graal, ChaseFUN.

DEMo DEMo ist eines der ersten Systeme, welche den Chase für den Datenaustausch nutzt. Es unterstützt den Anwender beim Entwurf „aussagekräftiger Schemaabbildungen, welche die gegebenen Target Constraints enthalten“ [PS09]. DEMo optimiert den Chase mit Hilfe des in [GN08] vorgestellten Algorithmus zur effiziente Kern-Berechnungen beim Datenaustausch. Gesucht ist also die unter allen universellen Lösungen die Lösung, welche bis auf Isomorphie am „kompaktesten“ ist. Insgesamt können so Redundanzen in der Ziel-Datenbank deutlich reduziert werden.

Details können in [PS09] nachgelesen werden. Weitere Veröffentlichungen zu DEMo sind uns nicht bekannt. Ein praktischer Test war uns aufgrund des fehlenden Quellcodes an dieser Stelle leider nicht möglich.

ChaseTEQ *ChaseTEQ*[2] ist ein Prototyp zum Reparieren und Anfragen unvollständiger Datenbanken. Es besteht aus drei Modulen zum Test verschiedener Terminierungs-Techniken (*ChaseT*), zum Erzeugen einer reparierten Datenbanken mit markierten Nullwerten (*ChaseE*) sowie zur Ausführung (eingeschränkter) SQL-Anfragen (*ChaseQ*). Wie der Name des Systems schon vermuten lässt, wird das Data Repairing bzw. Data Cleaning vom Chase übernommen. Implementiert sind jeweils eine Version des Standard-Chase, Naive Oblivious Chase und Skolem Oblivious Chase.

ChaseT implementiert verschiedene Terminierungstests. Neben der schwachen Azyklizität können auch die Kriterien Safety, superschwache Azyklizität sowie C-Stratifizierung überprüft werden. Während letztere in coNP liegt, erfolgen die anderen Tests in polynomieller Zeit. Alle Kriterien können zudem mit den Constraint-Rewriting-Techniken Adn und Adn^+ gekoppelt werden. Dies optimiert beispielsweise die schwache Azyklizität, welche in manchen Fällen fälschlicherweise eine Nicht-Terminierung liefert (siehe Abschnitt 3.3.4). Für uns ist dies das Alleinstellungsmerkmal von ChaseTEQ.

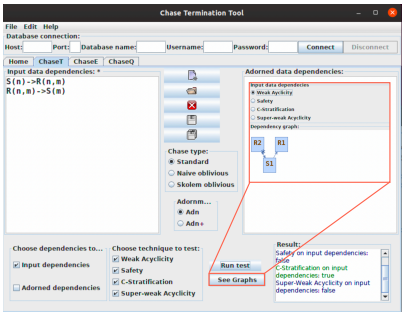
Das System wurde speziell für das Data Repairing entwickelt. Dies geschieht im Modul ChaseE. Hierfür wird die Eingabedatenbank mit Hilfe des Chase verarbeitet und doppelte Nullwerte durch sogenannte *Dummy-Attribute* ersetzt. ChaseQ ermöglicht anschließend die Auswertung eingeschränkter SQL-Anfragen.

ChaseTEQ ist ein Java-Tool, entwickelt im Rahmen der Dissertation von Francesca Spezzano [Spe11]. Es bietet eine Schnittstelle zur Definition von Abhängigkeiten, zum Testen der Chase-Terminierung und zur Reparatur einer gegebenen MySQL-Datenbank. Ein Screenshot der zugehörigen GUI ist in den Abbildungen 4.2a (Ausführung von *ChaseT*) und 4.2b (Ausführung von *ChaseE*) und Literatur zu ChaseTEQ, dem Rewriting-Verfahren Adn sowie den verschiedenen Terminierungskriterien in [FST11; SG10; Spe11; GMS12] zu finden.

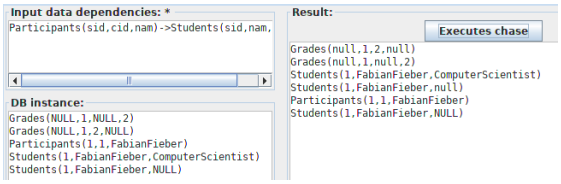
Praktischer Test: Tgds und egds können definiert werden, egds werden bei der Weiterverarbeitung aber weitestgehend ignoriert. Die Terminierungs-Tests funktionieren einwandfrei. Abbildung 4.2c zeigt beispielsweise den Abhängigkeitsgraphen für $\Sigma = \{\texttt{Student(name)} \rightarrow \exists\ \texttt{Leader} : \texttt{R(name, Leader)},\ \texttt{R(name, Leader)} \rightarrow \texttt{Student(Leader)}\}$ und Adn(Σ). Auch wenn ChaseTEQ speziell für das Data Repairing entwickelt wurde, sind einige Einschränkungen zu beachten. So können Leerzeichen in Zeichenketten, die Attributwerte darstellen, nicht verarbeitet werden und die Datentypen `FLOAT` und `DOUBLE` werden nicht erkannt. Auch bei der Darstellung von Nullwerten kommt es zu Problemen.

Schauen wir uns abschließend noch die Benchmark-Anfragen der semantischen Optimierung an: Da ChaseTEQ keine Implementierung des Backchase beinhaltet, können die Anfragen QB_1 bis QB_4 nicht getestet werden. Auch die Verarbeitung der Anfragen QA_2 und QA_3 sind wegen der fehlenden Unterstützung von Vergleichen nicht möglich. Die Anfrage QA_1 kann jedoch problemlos durchgeführt werden. Die Anfrage QA_4 ist ohne die Selektion ebenfalls möglich. Eine Optimierung erfolgt jedoch nicht.

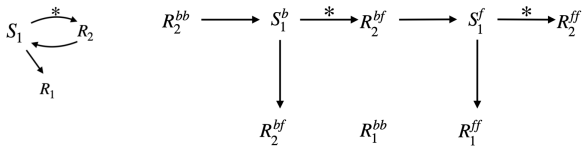
[2] ChaseTEQ-Projektwebsite: `http://wwwinfo.deis.unical.it/chaseteq/index.htm`



(a) Termininierung in ChaseTEQ mit Graph für schwache Azyklizität (rot hervorgehoben)



(b) Ausschnitt des E-Tabs in ChaseTEQ

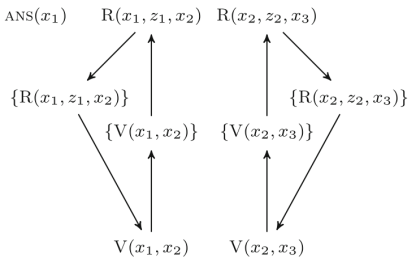


(c) Abhängigkeitsgraphen für Σ und Adn(Σ)

```
= Query =
?(M,Ma) :- student(M, N, V, S, I), noten(Ma, Mo, No).

= Answers =
{Ma->"11",M->"3"}
{Ma->"9",M->"3"}
{Ma->"3",M->"3"}
...
```

(d) Anfragetransformation in Graal



(e) Beispiel für QC-Graph in Pegasus [Mei14]

```
Projection[Matrikelnr/c1,Nachname/c2](
  Scan(Universitaetsangehoerige)
);
Join_SYMMETRIC_HASH(#0=#2)(
  SubPlanAlias(ALIAS109),
  Projection[Matrikelnr/c1,IEF-Nr/c4](
    Scan(IEF-Mitglieder)
  )
);
Projection[c1,c2](
  Join_SYMMETRIC_HASH(#3=#4)(
    SubPlanAlias(ALIAS110),
    Projection[IEF-Nr/c4](
      Selection{(#1=10)}(
        Scan(IEF-Faustballer)
      )
    )
  )
)
```

(f) Anfragetransformation in PDQ

Abbildung 4.2. CHASE-Ergebnisse in verschiedenen Systemen.

Pegasus Pegasus[3] ist ein Werkzeug zur semantischen Anfrageoptimierung. Hierfür wird eine Menge von Abhängigkeiten Σ so in eine gegebene Anfrage Q eingearbeitet, dass eine zu Q äquivalente, aber minimale Anfrage Q' entsteht. Um dies zu gewährleisten, implementiert Pegasus eine Variante des CHASE&BACKCHASE. Die zeitaufwendige BACKCHASE-Phase wird dabei durch graph-basierte Techniken optimiert [Mei14].

In der CHASE-Phase werden zunächst alle Abhängigkeiten aus Σ in die Anfrage Q eingearbeitet. Es entsteht ein *universeller Plan* U, dessen Unteranfragen anschließend auf Äquivalenz zu Q überprüft werden. Für die Optimierung der BACKCHASE-Phase stehen dabei üblicherweise zwei Ansätze zur Verfügung: der universelle Plan wird in disjunkte Fragmente zerlegt (*Online-Query-Fragmentierung*) oder die Abhängigkeiten aus Σ werden aufgespalten und nacheinander abgearbeitet (*Offline-Constraint-Stratifikation*).

Pegasus verallgemeinert diese Ansätze durch die Verwendung des sogenannten *QC-Graphen* (query-chase graph). Dieser repräsentiert alle möglichen CHASE-Sequenzen für jede Unteranfrage von U. Durch Ausnutzen der topologischen Sortierung kann die äquivalente, minimale Anfrage Q' leicht aus dem QC-Graphen abgeleitet werden (siehe Abbildung 4.2e). Diese Variante wird auch *geführter* BACKCHASE genannt. Weitere Details können in [Mei14] nachgelesen werden.

Praktischer Test: Wie in [Bre+20] beschrieben, konnten wir Pegasus trotz Vorliegens des originalen Quell-Codes aufgrund verschiedener technischer Schwierigkeiten nicht testen. Basierend auf der Literatur würden wir jedoch erwarten, dass Pegasus alle im Anhang B zusammengefassten Benchmark-Anfragen löst. Dies liegt daran, dass die Beispiele keine Einschränkungen oder Strukturen enthalten, die in früheren Veröffentlichungen über Pegasus wie [Mei14] nicht erwähnt wurden.

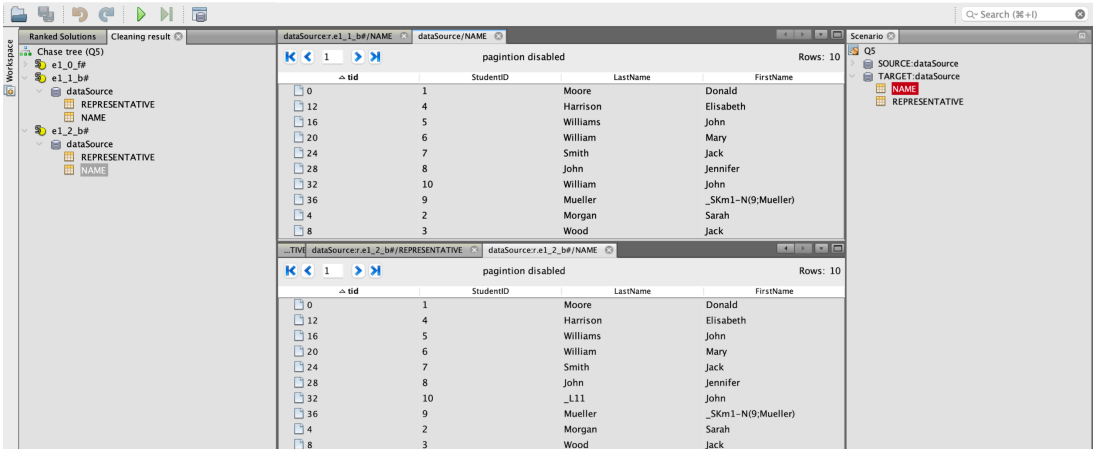
[3] Pegasus-Projektwebsite: `https://sourceforge.net/projects/chase-backchase/`

PDQ Proof-Driven Query Answering oder kurz PDQ[4] ist ein Werkzeug zur Anfrage-Transformation und zum semantischen Optimieren von Anfragen [BLT14; BLT15]. Das System wurde an der Universität Oxford entwickelt und 2014 veröffentlicht. Es kann über das Terminal angesprochen oder über eine Benutzeroberfläche bedient werden (siehe Abbildung 4.2h).

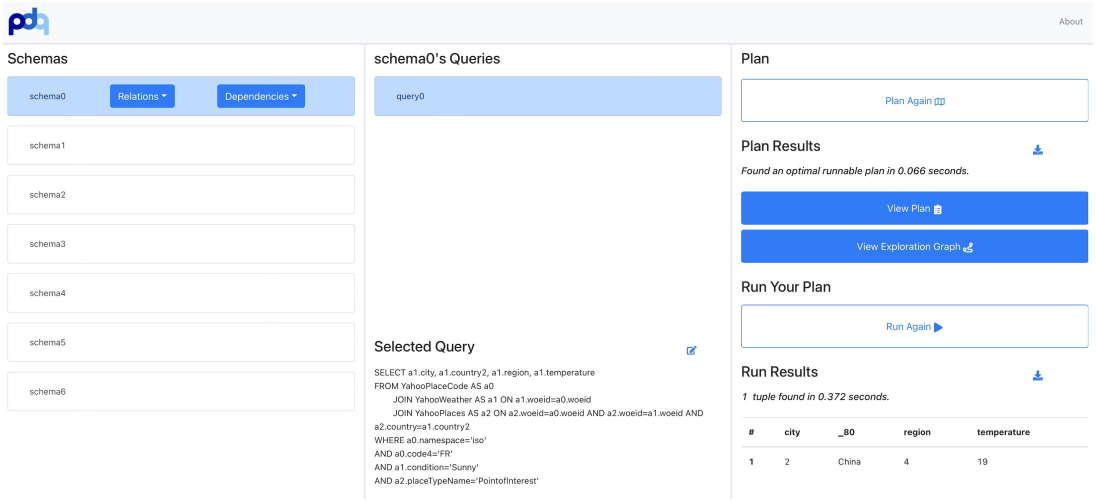
PDQ transformierte eine Anfrage so, dass gegebene Integritätsbedingungen erfüllt, Schnittstellenbeschreibungen wie beispielsweise Sichten eingehalten und ihre Kosten gemäß einer vorgewählten Kostenfunktion minimiert werden. Die Anzahl der möglichen Rewritings wird dabei durch die Verwendung eines Chase&Backchase-Algorithmus drastisch reduziert.

Praktische Tests: Wie in [Bre+20] beschrieben, liefert PDQ für die Benchmark-Anfragen QA_1 bis QB_3 stets eine optimierte äquivalente Anfrage. Das System verwendet allerdings mehr Projektionen als erwartet. Zu sehen ist dies beispielsweise im Anfrageergebnis zu QA_1 in Abbildung 4.2f. Die von PDQ bestimmten äquivalente Anfragen sind somit nicht immer minimal.

Lediglich Anfrage QB_4 kann von PDQ nicht gelöst werden. Stattdessen wird eine `IndexOutOfBoundsException` ausgelöst. Dies kann an einer fehlerhaften Implementierung liegen oder überschreitet schlicht die für PDQ geplante Anwendung. Insgesamt ist PDQ ein leistungsfähiges und leicht zu bedienendes Werkzeug, welches bis auf eine Anfrage alle unsere Testfälle lösen kann.



(g) Llunatic-GUI



(h) PDQ-GUI

Abbildung 4.2. Chase-Ergebnisse in verschiedenen Systemen.

[4] PDQ-Projektwebsite: `http://www.cs.ox.ac.uk/projects/pdq/home.html`

Prov$_{C\&B}$ Prov$_{C\&B}$[5] ist ein Algorithmus zum Query-Rewriting, der denselben universellen Plan wie C&B (siehe Definition 3.25) konstruiert [Haa99; DPT99], aber exponentiell weniger Kandidaten als C&B inspiziert. Der Algorithmus basiert auf dem sogenannten *provenance-aware* CHASE, welcher zusätzliche Provenance-Informationen zur Bestimmung der Anfragekandidaten verwendet. So kann Prov$_{C\&B}$ das Query-Rewriting direkt aus dem Ergebnis einer einzigen CHASE-Anwendung des universellen Plans ablesen und spart die exponentiell vielen Unteranfragen, welche beim klassischen C&B durchgeführt werden würden. Die so erzielte Reduzierung der CHASE-Anwendungen ist hierbei der entscheidende Grund für die Beschleunigung von Prov$_{C\&B}$ gegenüber C&B. Ein detailliertes Beispiel hierzu kann in [DH13] (Original) sowie [Aug17] (angewandt auf unser Studentenbeispiel) nachgelesen werden. Der formale Beweis zur Vollständigkeit von Prov$_{C\&B}$ ist in [Ile+14] zu finden.

Praktischer Test: Prov$_{C\&B}$ ließ sich problemlos installieren. Das Programm liefert bei Ausführung der aktuellsten Version jedoch lediglich zwei Zeitangaben. Ein konkretes CHASE-Ergebnis ist auf den ersten Blick nicht zu erkennen. Die in [Ile+14] vorstellten Ergebnisse können wir verifizieren, eigene Testanfragen führen jedoch zu keinem zufriedenstellenden Ergebnis.

Llunatic Llunatic[6][7] ist ein CHASE-basiertes Framework, das sich dem Data Exchange und Data Repairing bzw. Data Cleaning widmet. Das Tool interpretiert beides als zwei Seiten des selben Problems. Für eine gegebene Datenbankinstanz sowie eine Menge von Abhängigkeiten liefert Llunatic alle möglichen Lösungen, die während der CHASE-Anwendung entstehen, sowie eine minimale Lösung, welche alle einzuarbeitendem Abhängigkeiten erfüllt. Insgesamt unterscheidet Llunatic drei verschiedene Szenarien [Gee+14a]:

(a) traditionelle Data Cleaning- und Data Repairing-Szenarien, basierend auf speziellen *cleaning egds* [Gee+14a];

(b) traditionelle Data Exchange-Szenarien mit s-t tgds, target tgds und target egds;

(c) eine neue Form von Mapping- und Cleaning-Szenarien, in welchen Daten von einer Quelle in ein Ziel verschoben werden, welches entsprechend den Cleaning-Bedingungen repariert wird.

Die Anzahl der möglichen Lösungen kann hierbei durch die Auswahl zusätzlicher Kostenmanager beschränkt werden [Gee+14a].

Die Lösungen selbst werden im sogenannten CHASE-*Baum* zusammengefasst [Gee+14a]. Er besteht aus allen möglichen CHASE-Zwischenergebnissen, der Quell-Instanz als Wurzel sowie den CHASE-Lösungen als Blättern des Baumes. Die Offenlegung der CHASE-Zwischenschritte ermöglicht eine präzise Interaktion mit dem Nutzer. Dieser kann spezielle Zwischenschritte untersuchen und Ersetzungen im Sinne des Data Repairings bzw. Data Cleaning präferieren. Welche Attributwerte dabei für den Nutzer interessant sein können, wird durch sogenannte *Lluns* hervorgehoben [Gee+14a]. Dies sind spezielle Llunatic-interne Platzhalter, welche „reparatur-bedürftige" Attributwerte festlegt.

Alle Informationen zu Llunatic sowie die zugehörige Literatur wie etwa [Gee+13; Gee+14b; Gee+14a] sind auf der Projektwebsite zu finden. Die dort verfügbare Programmversion bildet zudem die Grundlage für unsere praktischen Tests. Llunatic wurde einige Jahre später noch einmal aufgearbeitet [Gee+20]. Eine neue Version des Systems ist dabei jedoch nicht entstanden.

Praktischer Test: Llunatic implementiert einen Test auf schwache Azyklizität (siehe Definition 3.32). Dieser besagt, dass die Ausführung des CHASE nur dann erlaubt ist, wenn der Algorithmus auch tatsächlich terminiert. Den größten Mehrwert liefert Llunatic jedoch in seiner granularen Behandlung von Nutzereingaben, welche es uns ermöglichen, zu jedem Zeitpunkt über den CHASE-Baum in den Ablauf des Algorithmus einzugreifen. Ein Screenshot der Llunatic-GUI ist in Abbildung 4.2g zu sehen. Llunatic hat sich auf die Anwendungsfälle Data Repairing bzw. Data Cleaning und Data Mapping spezialisiert. Die im Llunatic-Repository mitgelieferten Beispiele können fehlerfrei rekonstruiert und auf unsere eigenen Beispiel-Datenbank übertragen.

Llunatic führt zudem unsere Benchmark-Anfragen für die semantische Optimierung beinahe fehlerfrei aus. Eine Optimierung findet jedoch nicht statt. Lediglich bei Anfrage QA_2 werden die Nullwerte nicht richtig behandelt. Insbesondere im Bereich Data Repairing bzw. Data Cleaning und Data Mapping ist Llunatic viel umfangreicher, als

[5] Prov$_{C\&B}$-Projektwebiste: `https://github.com/IoanaIleana/ProvCB`
[6] Llunatic-Projektwebsite: `http://www.db.unibas.it/projects/llunatic/`
[7] Llunatic-GitHub: `https://github.com/donatellosantoro/Llunatic`

wir mit unseren Benchmark-Anfragen zeigen können. Die im Llunatic-Repository mitgelieferten Beispiele können wir fehlerfrei rekonstruieren und auf unsere eigenen Beispiel-Datenbank übertragen.

Graal Graal[8] widmet sich der Ontologie-vermittelten Beantwortung von Anfragen, welches zwischen 2012 und 2015 vom GraphIK-Team in Frankreich entwickelt wurde. Graal basiert auf praktischen Anwendungserfahrungen wie in [Bag+15] beschrieben und besitzt eine ausführliche Projekt-Website mit zusätzlichen Experimenten, App-Anwendungen sowie einer Liste von Veröffentlichungen und Dokumentationen. Ein Ergebnis unserer eigenen Tests ist in Abbildung 4.2d zu sehen.

Graal ist „modular aufgebaut, um die Wiederverwendung und Erweiterung der Software zu erleichtern“ (Quelle: Graal-Website). Die Entwickler erleichtern so dem Nutzer die Tests neuer Szenarien und Techniken. Neben verschiedenen Schichten wie einer Datenebene, einer onthologischen Ebene, einer Wissensbasis, einem Regelanalysator sowie verschiedenen Hilfswerkzeugen implementiert Graal zwei Klassen von Algorithmen zur Verarbeitung von Ontologie-gestützten Anfragen. Hierzu gehören Algorithmen zur Umschreibung von Anfragen, welche den Kern der Graal-Techniken bilden, und das *forward chaining*, eine Version des restricted (oder Standard-)Chase sowie eine Version des skolem Chase. Details können in den auf der Projekt-Website hinterlegten Publikationen nachgelesen werden.

Praktischer Test: Graal ist ein multifunktionales Werkzeug mit vielen Anwendungsbereichen. So implementiert Graal neben grundlegenden Data Exchange-Funktionalitäten auch ein Query Rewriting, welches Ähnlichkeiten mit dem Backchase aufweist. Die Benchmark-Anfragen QA_1 sowie QB_1 bis QB_3 werden fehlerfrei optimiert. In einigen Fällen stellt Graal somit eine Alternative zum Backchase dar. Die Benchmark-Abfragen QA_2 und QB_4 werden aufgrund der fehlenden EGD-Unterstützung nicht optimiert. Auch Benchmark-Anfrage QA_3 kann Graal nicht optimieren. In diesem Fall scheint der Chase&Backchase unverzichtbar und das in Graal implementierte forward und backward chaining nicht ausreichend zu sein.

ChaseFUN ChaseFUN[9] wurde 2016 von Ioana Ileana entwickelt. Es ist damit eines der neueren Systeme mit dem Schwerpunkt des Datenaustausches. Das System verarbeitet funktionale Abhängigkeiten sowie s-t tgds und implementiert eine Variante des skolem oblivious Chase. Durch Umordung der Chase-Schritte ermöglicht ChaseFUN eine Schritt-für-Schritt-Verarbeitung, eine Parallelisierung sowie eine Beschleunigung des Chase-Algorithmus durch Reduzierung der Größe der Zwischenergebnisse. Details zu ChaseFUN können in [BIL16; BIL17] nachgelesen werden.

Praktischer Test: Aufgrund technischer Schwierigkeiten war es uns nicht möglich, ChaseFUN selber zu testen. Wir erwarten wegen der Parallelität jedoch eine schnellere Durchführung des Data Exchange-Problems als etwa bei Llunatic. Zudem vermuten wir, dass die Anfragen QA_1 bis QA_3, analog zu Llunatic, ausgeführt, aber nicht optimiert werden.

4.1.4. Fazit:

Wir halten fest, dass keines der Systeme alle sieben Benchmark-Anfragen optimiert. Jedes System ist für einen Spezialfall konzipiert, welchen er jedoch sehr zuverlässig und wie in den Literatur „beworben“ erfüllt. So dominiert ChaseTEQ bei der Überprüfung der verschiedenen Terminierungskriterien. Llunatic überzeugt bei der Bearbeitung von Data Exchange- und Data Cleaning- bzw. Data Repairing-Szenarien. Die Verarbeitung von Anfragen ist hingegen Schwerpunkt von Graal und PDQ.

Auch wenn es Überschneidungen in den jeweiligen Anwendungsgebieten gibt, vereint keines der getesteten Systeme die drei Hauptaufgaben (1) Überprüfung der Terminierung sowie die Ausführung des Chase-Algorithmus auf (2) Anfragen und (3) Instanzen. Dies wollen wir durch eine eigene Implementierung des CHASE ändern. So verarbeitet ChaTEAU sowohl Anfragen — ebenso wie PDQ und Graal — als auch Instanzen — analog Llunatic — und implementiert zudem fünf grundlegende Terminierungstests — wie ChaseTEQ. Einen konkreten Anwendungsfall vertritt ChaTEAU nicht. Näheres hierzu folgt in Kapitel 9.

[8] Graal-Projektwebsite: `https://graphik-team.github.io/graal/`
[9] ChaeFUN-Projektwebiste: `https://github.com/dbunibas/chasebench/tree/master/tools/chasefun`

Eigene sowie betreute Arbeiten

Die Auswahl der zu testenden Systeme basiert auf dem renommierten Benchmark-Artikel [Ben+17] sowie Gesprächen und Reviews mit und von Chase-Experten auf Konferenzen und Tagungen. Alle getesteten Systeme liegen in ihrer aktuellsten Version als virtuelle Maschine oder zip-Datei vor. Die Durchführung der Benchmark-Anfragen wurde insbesondere in drei studentischen Projektarbeiten durchgeführt. Die dort vorgestellten Ergebnisse wurden anschließend geprüft, gegebenenfalls erweitert und korrigiert. Insbesondere bei den Ergebnisse der Literaturanalyse waren einige Korrekturen notwendig:

- [RR17] Renn, Röger: Chase-Tools (Teil I), KSWS[10], WS17/18
- [Bre+19] Brekenfelder, Kronenberg, Rutofski, Schimanski, Zimmer: Chase-Tools (Teil II), Projekt[11], SS19
- [Bre+20] Brekenfelder, Kronenberg, Lamster, Manthey, Zimmer: Chase-Tools (Teil III), NEidI[12], SS20

4.2. Analyse verschiedener Provenance-Systeme *(nach [AH22b])*

In dem Maße, in welchem das Themen-Gebiet *Reproduzierbarkeit und Provenance* an Bedeutung gewinnt, wächst auch die Anzahl der entwickelten Provenance-Systeme. So konnten Pérez et al. 2017 bereits 251 in der Literatur bekannte Systeme identifizieren [PRS18]. Hierfür untersuchten sie in einer mehrstufigen Literatur-Analyse Surveys sowie publizierte Artikel verschiedener Konferenzen und Journale. Hierzu gehören unter Anderem der International Provenance and Annotation Workshop (IPAW)[13], der Workshop on Theory and Practice of Provenance (TaPP)[14], die Future Generation Computer Systems (FGCS)[15] sowie verschiedene ACM SIGMOD-Formate wie das SIGMOD Record Journal[16], die ACM SIGMOD International Conference on Management of Data (SIGMOD)[17] und das ACM SIGMOD Symposium on Principles of Database Systems (PODS)[18]. In [PRS18] nicht genannt, aber unserer Meinung nach erwähnenswert, ist zudem die ProvenanceWeek[19]. Sie vereint die IPAW und TaPP mit kleineren Workshops zum Thema Provenance in einer gemeinsamen, alle zwei Jahre stattfindenden Veranstaltung.

Basierend auf den Ergebnissen ihrer Literaturstudie definieren die Autoren von [PRS18] eine sechs-dimensionale Taxonomie von Provenance-Merkmalen. Diese besteht aus den folgenden Aspekten:

- allgemeine Aspekte, welche den allgemeinen Hintergrund der Provenance-Systeme beschreiben;
- Datenerfassung, welche die Erfassung der Provenance-Daten mit Hilfe der bestehenden Systemen beschreibt;
- Datenzugriff, welcher den Zugriff eines Nutzers auf die Provenance-Daten-Repositories beschreibt;
- Subjekt, welches unter Berücksichtigung der Interoperabilität die Provenance-Darstellung beschreibt;
- Speicherung der Provenance-Informationen;
- nicht-funktionale Anforderungen an Provenance-Systeme, wie z. B. Sicherheit.

Die Studie ergab zudem, dass 25 Systeme im Provenance-Kontext besonders häufig referenziert werden. Dabei werden nach Aussage der Autoren in der Literatur hauptsächlich zwei Ebenen unterschieden: innerhalb wissenschaftlicher Datenverarbeitung und *Work-Flow-Management-Systeme* (engl. Scalable Scientific Workflows Management Systems (SWfMS) sowie innerhalb von *Datenbank-Management-Systemen* (DBMS). Die meisten Systeme arbeiten auf einer der beiden Ebenen. Es gibt jedoch auch Ansätze, welche es schaffen, Provenance auf mehreren Ebenen zu erfassen. So erfasst beispielsweise LipStick [Ams+11] neben dem internen Zustand, d.h. der Data Provenance, auch fein granulären Abhängigkeiten der Workflow-Provenance.

[10] KSWS: *Komplexe Software-Systeme*, Mudol an der Universität Rostock im Bachelorstudiengang Informatik
[11] Projekt: Modul an der Universität Rostock im Bachelorstudiengang Informatik
[12] NEidI: *Neueste Entwicklungen in der Informatik*, Modul an der Universität Rostock im Masterstudiengang Informatik
[13] IPAW: `http://ipaw.info`
[14] TaPP: `https://www.usenix.org/conferences/byname/186`
[15] FGCS: `https://www.sciencedirect.com/journal/future-generation-computer-systems`
[16] SIGMOD Record Journal: `https://dl.acm.org/newsletter/sigmod`
[17] SIGMOD: `https://dl.acm.org/conference/mod`
[18] PODS: `https://sigmod.org`
[19] ProvenanceWeek: `http://provenanceweek.org`

Tool	Level	Provenance-Typ	Verfügbarkeit
Buneman	Datenbank	***where***, ***why***	—
Cui 2000	Data Warehouse	Lineage	—
DBNotes	Datenbank	***where***	—
Perm/GProM	Datenbank	***how***, ***why***, ***where***, Lineage	Open Source
Lipstick	Workflow oder Datenbank	—	Open Source
Orchestra	Data Warehouse	***how***	Open Source
Tioga/Tioga-2	Datenbank	***how***	Open Source
Trio	Datenbank	***how***	Open Source
$\text{Prov}_{\text{C\&B}}$	—	***why***	Open Source
ProvSQL	Datenbank	***how***	Open Source

Tabelle 4.3. Klassifikation der untersuchten Provenance-Systeme nach Literatur, basierend auf [PRS18; Sen+18; Ile14].

Als Systeme, welche rein auf dem Datenbank-Level agieren, identifiziert [PRS18] Trio, Tioga, Buneman, Perm bzw. GProM sowie DBNotes. Neben diesen arbeiten auch Cui 2000 und Orchestra anfrage-basiert, jedoch auf Data Warehouses (siehe Tabelle 4.3). Da Cui 2000 ebenso wie Buneman nicht Open Source-zugänglich sind, ist uns eine praktische Analyse hier nicht möglich. Gleiches gilt für Lipstick. Obwohl der Code von Lipstick laut [PRS18] öffentlich zugänglich sein soll, konnten wir ihn leider nicht auffinden. Auch Workflow-basierte Systeme wie beispielsweise Kepler [ABJ06] werden an dieser Stelle vernachlässigt, da hier eine Klassifikation des in Abschnitt 3.4.1 vorgestellten Provenance-Typs nach ***where***, ***why***, ***how*** oder ***why not*** nicht zielführend ist. Einen Überblick über die theoretischen Hintergründe der Workflow-Provenance sowie deren wichtigste Systeme können in [HDL17] und [PRS18] nachgelesen werden. Wir konzentrieren uns im Folgenden auf die Provenance-Systeme der DBMS-Ebene.

4.2.1. Auswahl der Provenance-Systeme

Basierend auf [PRS18] ergeben sich so fünf Systeme, welche wir praktisch analysieren wollen: Perm [GA09], GProM [Ara+18], Orchestra [Ive+08], Tioga [Sto+93] und Trio [Wid04]. Zusätzlich wählen wir noch ProvSQL [Sen17] und $\text{Prov}_{\text{C\&B}}$ [Ile14]. Ersteres ist das aktuell neueste Provenance-System, welches 2018 auf der ProvenanceWeek erstmalig vorgestellt wurde. $\text{Prov}_{\text{C\&B}}$ implementiert den sogenannten *provenance-aware* Chase&Backchase. Es ist somit eines der wenigen Systeme, welches Provenance mit dem Chase-Algorithmus kombiniert. Ziel ist hier das Query Rewriting wie etwa bei AQuV. Beide Systeme sind zudem Open Source (siehe Tabelle 4.3). Insgesamt erhalten wir so die in Abbildung 4.3 zusammengefasste Auswahl an Provenance-Systemen, welche wir im weiteren Verlauf dieses Abschnitts in einer ausführlichen Literaturrecherche sowie einigen praktischen Tests (rot hervorgehoben) untersuchen werden. Da das Gebiet der Provenance und somit auch seiner praktischen Umsetzung noch recht jung ist, konzentrieren uns auf die bekanntesten und somit relevantesten Provenance-Systeme der letzten 30 Jahre.

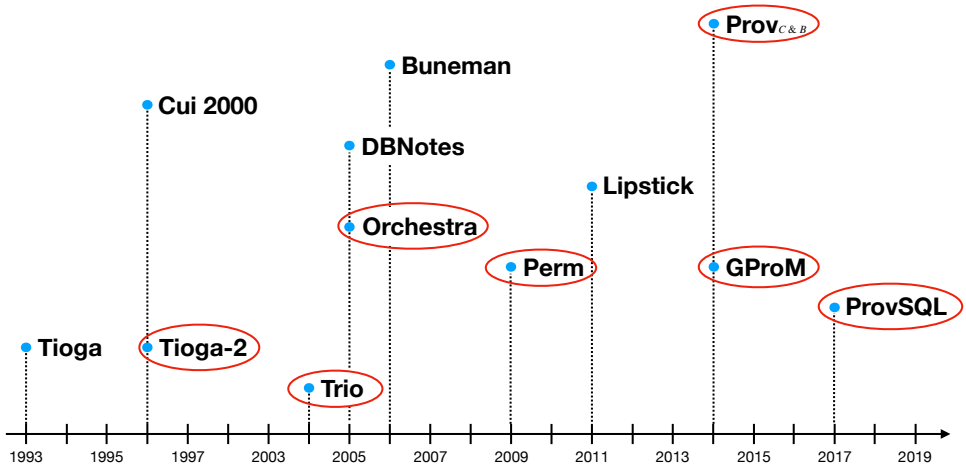


Abbildung 4.3. Auswahl der untersuchten Provenance-Systeme.

Fassen wir unsere Auswahl noch einmal zusammen: Tioga ist das älteste Provenance-System, welches wir bestimmen konnten. Trio ist auf Data Lineage, einem Vorgänger der heutigen ***why***-Provenance, und Uncertainty spezialisiert. Orchestra ist das erste System, welches die sogenannten Provenance-Polynome verarbeitet. Es ist somit ein Grundstein der ***how***-Provenance. Perm verwendet Annotation Propagation und Query Rewriting zur Bestimmung der Provenance und ist zudem Vorgänger von GProM, welches einen Ansatz zur Erzeugung von Provenance Game-Graphen zur Beantwortung der ***why***- und ***why not***-Provenance realisiert. ProvSQL ist das neueste von uns getestete System. Es verwendet Semi-Ringe zur Berechnung von Provenance-Polynomen, welche durch benutzerdefinierte Aliase besonders anschaulich dargestellt werden. $\text{Prov}_{\text{C\&B}}$ kombiniert eine Variante des Chase&Backchase mit der ***why***-Provenance. Es ist daher sowohl als Chase- als auch als Provenance-System interessant. Da wir Cui 2000, DBNotes, Buneman und Lipstick nicht praktisch testen können, beschränken wir uns bei diesen vier Systemen auf einen kurzen Literaturreview.

4.2.2. Ergebnisse der praktischen Tests

Wir konnten insgesamt sieben Systeme praktisch testen (siehe Abbildung 4.3, rot hervorgehoben). Sie stehen in ihrer aktuellen Version als virtuelle Maschinen unter `https://nextcloud.informatik.uni-rostock.de/index.php/s/SNXPC8BaxR9i9z5` zur Verfügung. Ziel der praktischen Tests ist die Überprüfung der in der Literatur angegebenen Funktionalitäten. Hierfür definieren wir vier Benchmark-Anfragen QP_1 bis QP_4 basierend auf dem Studierenden-Beispiel aus Anhang A. Diese Anfrage bilden die Grundlage für unsere anschließenden Provenance-Anfragen ***where***, ***why*** und ***how*** sowie die Data Lineage.

Anfrage QP_1 liefert eine Liste aller besuchten Studiengänge. Sie testet die Duplikat-Eliminierung. Anfrage QP_2 verarbeitet eine Selektion sowie einen natürlichen Verbund, indem die Namen aller Teilnehmer eines konkreten Moduls ausgegeben werden. Anschließend erweitern wir diese Anfrage um einen weiteren Verbund. Durch Vernachlässigung der Selektion entsteht eine Verbund-Kette, deren mittlere Relation lediglich als „Verbundrelation“ fungiert. Sie taucht daher in der ***where***-Provenance nicht mehr auf. In der ***why***- und ***how***-Provenance ist sie jedoch weiter zu finden. Wir erhalten QP_3, welche in Anfrage 4.3 exemplarisch dargestellt ist. Die Anfragen QP_1, QP_2 und QP_4 können im Anhang B nachgelesen werden. Anfrage QP_4 bestimmt die Notenverteilung eines konkreten Moduls. Sie dient somit als Vertreter der einfachen Aggregat-Funktionen aus Abschnitt 3.2.

Anfrage 4.3 Verbund dreier Relationen (QP_3)

```
SELECT s.firstname, l.fullname
FROM students s, participants p, lecturers l
WHERE s.student_id = p.student_id
AND p.course_nr = l.course_nr;
```

Hinweis: Da Tioga-2/DataSplash nicht fehlerfrei aufgesetzt werden konnte und die Nutzung von $\text{Prov}_{\text{C\&B}}$ nicht abschließend geklärt werden konnte, entfallen beide Systeme bei den folgenden Betrachtungen. Wir kennzeichnen dies durch (—).

Benchmark-Anfrage	Tioga	Trio	Orchestra	Perm	GProM	$\text{Prov}_{C\&B}$	ProvSQL
QP_1	—	(✓)	✓	✓	✓	—	✓
QP_2	—	✓	✓	✓	✓	—	✓
QP_3	—	✓	✓	✓	✓	—	✓
QP_4	—	(✓)	✗	✓	✓	—	✗

Tabelle 4.4. Ergebnisse der Benchmark-Anfragen im Vergleich zu den erwarteten Ergebnissen aus Anhang B, basierend unter anderem auf [Fla+20a]. Legende: Benchmark-Ergebnis erhalten (✓), Benchmark-Ergebnis nicht erhalten (✗), praktischer Test nicht möglich (—).

Wie erwartet, können die Benchmark-Anfragen QP_1, QP_2 und QP_3 von allen getesteten Provenance-Systemen (Trio, Orchestra, Perm, GProM, ProvSQL) verarbeitet werden (siehe Tabelle 4.4). Einfache SPJ-Anfragen sind in den verschiedenen Systemen somit problemlos umsetzbar. Für eine Verarbeitung in Trio muss Anfrage QP_1 jedoch ein wenig umgeschrieben werden, da der Aufruf von `DISTINCT` sonst die Fehlermeldung *can only concatenate tuple (not "list") to tuple* verursacht. Da Aggregatfunktionen in Orchestra nicht unterstützt werden, kann Anfrage QP_4 dort nicht umgesetzt werden. Gleiches gilt für ProvSQL. Nach Aussage der Entwickler von ProvSQL ist eine

solche Erweiterung in der Zukunft jedoch noch geplant. Auch eine Verarbeitung in Trio ist nicht direkt möglich. Hierfür muss die Aggregation zunächst als horizontale Subquery aufgefasst werden. Entscheidend ist hier das Schlüsselwort `HCOUNT` statt `COUNT`, siehe Anfrage 4.4.

Anfrage 4.4 Aggregation und Gruppierung (QP_4), umgeschrieben für Verarbeitung in Trio

```
SELECT grade, hcount(*) AS gcount
FROM grades
WHERE courseID = '002'
GROUP BY grade
ORDER BY grade
```

Um die Ergebnisse der Literaturanalyse von [PRS18] verifizieren zu können, bestimmen wir für unsere Benchmark-Anfragen auch jeweils den umgesetzten Provenance-Typ. Wie in Abbildung 4.4 zu sehen, bieten bis auf Tioga-2/ DataSplash alle Systeme eine graphische Darstellung der Provenance. GProM liefert einen Provenance Game-Graphen, welcher je nach Färbung eine Antwort auf die Frage ***why*** bzw. ***why not*** gibt. Perm erweitert die Ergebnistupel um zusätzlichen Provenance-Attribute, welche wiederum der Zeugenbasen der ***why***-Provenance entsprechen. ProvSQL und Orchestra beantworten die ***how***-Provenance durch Angabe eines Provenance-Graphen. In ProvSQL kann das zugrunde liegende Provenance-Polynom zudem als Formel über einem beliebigen beschreibendem Alias ausgegeben werden.

Wie in Tabelle 4.5 zu sehen, wird die ***where***-Provenance von allen Systemen indirekt durch Angabe einer höherwertigen Provenance mit ausgegeben, dargestellt durch einen Hacken (✓). Aufgrund der Provenance-Reduktion aus Abschnitt 3.4 kann die Zeugenbasis der ***why***-Provenance immer aus dem Provenance-Polynom oder dem Provenance-Graphen der ***how***-Provenance abgeleitet werden. Gleiches gilt für die ***where***-Provenance, welche aus der ***why***- oder ***how***-Provenance abgeleitet werden kann. ProvSQL und Orchestra beispielsweise erfüllen die ***why***- und ***where***-Provenance, auch wenn dies nicht explizit mit einem ✓gekennzeichnet ist.

Provenance	Tioga	Trio	ORCHESTRA	Perm	GProM	$\text{Prov}_{C\&B}$	ProvSQL
where	—	✓	✓	✓	✓	—	✓
why	—	✗	✗	✓	✓ (Game-Graph)	—	✗
why not	—	✗	✗	✗	✓	—	✗
how	—	✗	✓ (Graph)	✗	✗	—	✓ (Formel & Graph)
Lineage	—	✓	✗	✗	✗	—	✗

Tabelle 4.5. Bestimmung des unterstützten Provenance-Typs. Legende: für die Provenance-Anfrage geeignet (✓), für die Provenance-Anfrage nicht geeignet (✗), praktischer Test nicht möglich (—).

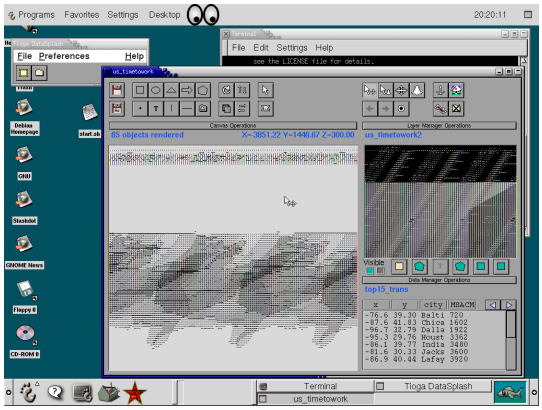
Da die in Trio implementierte Lineage der ***how***-Provenance ähnelt, definieren die Autoren von [PRS18] den zugehörigen Provenance-Typ als ***how***, wir hingegen als Lineage. Wir halten uns dabei an die von den Entwicklern genutzte Begrifflichkeit, unabhängig von der Entwicklung des Begriffs Lineage. So ist auch der Unterschied der Funktionalität von Perm und GProM bzgl. Lineage zu erklären (vgl. Tabelle 4.3 und 4.5).

4.2.3. Die Systeme im Detail

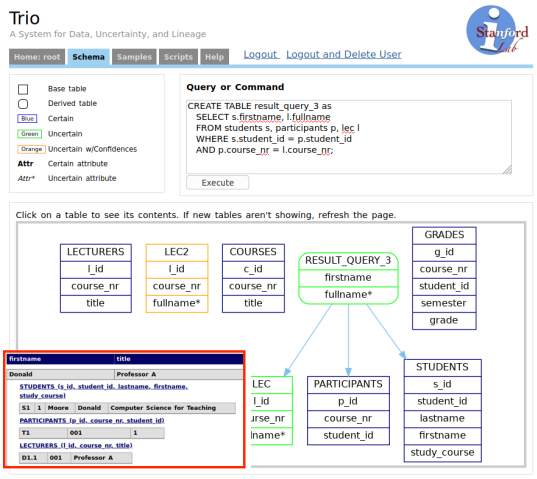
Betrachten wir die ausgewählten Systeme nun etwas genauer. Die Reihenfolge, in der wir die Systeme vorstellen, basiert auf ihrem in Abbildung 4.3 dargestellten Veröffentlichungsdatum. Die hier vorgestellten Ergebnisse sind zudem im technischen Bericht [AH22b] zu finden.

Tioga Das *Tioga-System*[20] — das älteste der von uns untersuchten Systeme — hat zwei Versionen: Tioga und Tioga-2. Die erste Version von Tioga wurde 1993 als eines der ersten Provenance-Systeme von Michael Stonebraker et al. in [Sto+93] vorgestellt. Zur „Unterstützung der Datenverwaltung von wissenschaftlichen Visualisierungen“ entwickelt, wurde es zunächst für die Visualisierung der Ausbreitung von Waldbränden verwendet, welche in Form eines gerichteten Graphen dargestellt wird. Die Darstellung erinnert uns an die heutige *Workflow-Provenance* (siehe Abschnitt 3.4).

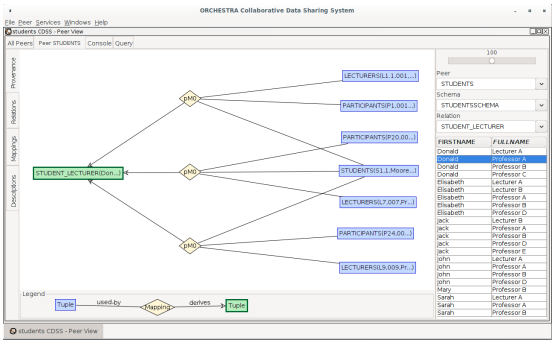
[20] Tioga-Projektwebsite: `http://datasplash.cs.berkeley.edu`



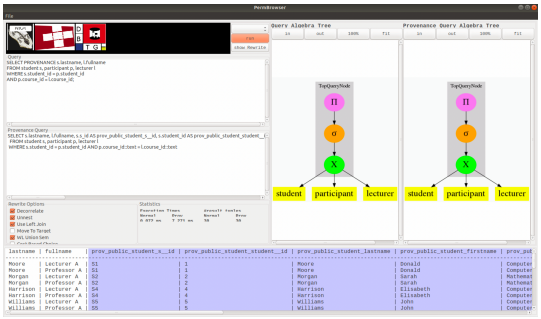
(a) Fehlerhafte Visualisierung in DataSplash [Fla+20b]



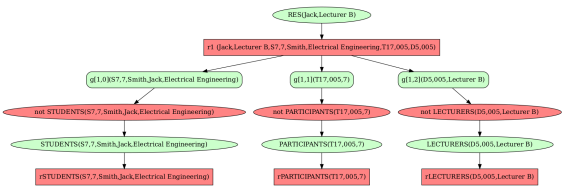
(b) TrioExplorer



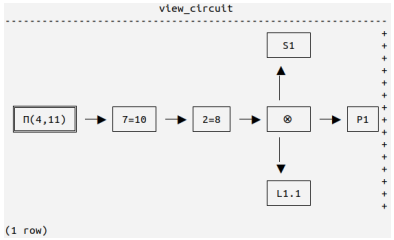
(c) Provenance-Graph in Orchestra



(d) Perm-Broswer



(e) Provenance Game-Graph in GProM



(f) Provenance-Graph in ProvSQL

Abbildung 4.4. Provenance-Darstellung der Anfrage QP_3 in verschiedenen Systemen.

Das 1996 entwickelte Re-Design *Tioga-2* bzw. *DataSplash* [Aik+96; WS97], basiert auf früheren Nutzererfahrungen von Tioga. Er ersetzt beispielsweise die „Boxen-Pfeil"-Notation durch ein Paradigma der direkten Manipulationsprogrammierung, sodass ein benutzerfreundlicheres, flexibleres und leistungsfähigeres Visualisierungssystem entsteht. Dabei gilt DataSplash als erstes System, welches die Provenance von benutzerdefinierten Funktionen mit Hilfe inverser Funktionen darstellen kann. Laut [PRS18] sollen Tioga und seine Nachfolger zudem die ***how***-Provenance darstellen können.

Praktischer Test: Da die Projektwebseite von Tioga nicht mehr existiert und der Quellcode für uns nicht verfügbar war, konnten wir Tioga nicht selbst testen. Die Installation des Nachfolgers DataSplash war möglich. Wir konnten das System, wie in Abbildung 4.4a zu sehen, jedoch nicht fehlerfrei zum Laufen bringen [Fla+20b].

Cui 2000 *Cui 2000* ermöglicht eine effiziente und konsistente Rückverfolgung von Datenreihen insbesondere in Bezug auf Data Warehouses. Es wurde um die Jahrtausendwende von Cui und Widom entwickelt. Basierend auf [Wie+96], widmet sich Cui 2000 dem *view data lineage*-Problem. Es ist in der Lage, Daten aus heterogenen Quellen zu sammeln, zu transformieren und gemäß der Warehouse-Spezifikation zusammenzufassen sowie in das Warehouse zu integrieren. Details hierzu können in [CWW00; CW00] nachgelesen werden. Ein praktischer Test war uns aufgrund des fehlenden Quellcodes leider nicht möglich.

Trio *Trio*[21] wurde zwischen 2004 und 2010 von Widom et al. entwickelt. Es erweitert PostgresSQL um *Data Lineage* sowie *Uncertainty* und vereinte diese in einer graphischen Benutzeroberfläche, dem *TrioExplorer*. Eine weitere Interaktionsmöglichkeit bietet TrioPlus eine Terminal-Schnittstelle. Auch das Einlesen kurzer SQL-Skripte ist möglich.

Die in Trio umgesetzte Lineage wird in einer separaten Tabelle gespeichert und kann für jedes Ergebnistupel einzeln ausgegeben werden. Da sich die Definitionen der ***why***- und ***how***-Provenance sowie der Data Lineage im Laufe der Jahre weiterentwickelt haben, ist die *Trio-Lineage* aus heutiger Sicht nicht einfach zu definieren. Sie ist informativer als die ***why***-Provenance, aber weniger informativ als die ***how***-Provenance, wie in Abbildung 4.4b zu sehen ist.

Die zweite Besonderheit von Trio ist die Implementierung der Uncertainty. Diese erlaubt die Definition von alternativen Werte für ein, mehrere oder alle Attribute eines Datenbanktupels sowie die Angabe von Vertrauenswerten, welche die Wahrscheinlichkeit für das Vorhandensein des Attributs beschreiben (siehe Abbildung 4.4b, grün oder orange hervorgehoben). Beide Informationen werden durch zusätzliche Attribute angeben.

Das Trio Data Model (TDM) wird erstmals in [Wid04] vorgestellt und dann ein neues Datenmodell namens *Uncertainty Lineage Database Modell (ULDB)* sowie eine separate Anfragesprache namens *TriQL* erweitert [Agr+06; Ben+06; Wid08]. Den Abschluss der Trio-Ära bilden die beiden neuen Varianten *Trio-ER* und *LIVE*, welche ab 2009 entstanden. Details hierzu können in [Agr+09] und [STW09] nachgelesen werden. Die bekannteste und meistgenutzte Trio-Version ist jedoch immer noch das Original von 2004.

Praktischer Test: Der Trio-Quellcode ist Open Source und auf der Trio-Projektseite der Stanford University verfügbar. Bei Ausführung einer Anfrage wird die Lineage automatisch mitberechnet. Ein zusätzlicher Befehl ist an dieser Stelle daher nicht notwendig. Die Ausgabe eines konkreten Ergebnistupels ist in Abbildung 4.4b zu sehen (Provenance-Schema rechts und Provenance-Ergebnis eines konkreten Tupels links). Für das Ergebnistupel (Donald, Professor A) der Anfrage Q_3 gibt die Data Lineage an, dass das Tupel aus den drei Quelltupeln S_1, T_1 und $D_{1.1}$ entstanden ist.

Trio verarbeitet einfache SQL-Anfragen — ohne jedoch das Schlüsselwort `JOIN` zu erlauben — inklusive der Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM` und `AVG`. Letztere können jedoch nicht mit Uncertainty kombiniert werden. Anders als in der Dokumentation beschrieben, war es uns nicht möglich, `DELETE`- und `UPDATE`-Anweisungen durchzuführen. Auch die Angabe von Fremdschlüsseln ist in Trio nicht möglich.

Insgesamt hat Trio eine sehr übersichtliche Benutzeroberfläche. Die Handhabung von Trio ist im Hinblick auf die Provenance-Informationen sehr intuitiv. Die Nutzung der Uncertainty ist etwas komplizierter, aber ebenfalls einfach. Details zu unseren Uncertainty-Tests können in [Fla+20b] nachgelesen werden.

Orchestra Um einen direkten Datenaustausch zwischen zwei (ursprünglich biowissenschaftlichen) Datenbanken zu ermöglichen, verfolgt *Orchestra*[22] die ausgetauschten Provenance-Informationen. Zu diesem Zweck erzeugt es *Provenance-Graphen* mit Tupel- und Mapping-Knoten sowie passenden Verbindungskanten. Intern arbeitet Orchestra mit den in [GKT07] definierten Provenance Semi-Ringen und ist damit das erste System, welches die ***how***-Provenance implementiert. Die Navigation durch den Provenance-Graphen wird durch die eigens hierfür entwickelte Anfragesprache *ProQL* ermöglicht [Ive+05; Gre+07; GKT07; GT17; Ive+08]. Darüber hinaus eliminiert Orchestra Duplikate und ermöglicht die Auswertung einfacher SPJU-Anfragen. Aggregatfunktionen werden jedoch nicht unterstützt. Die notwendige Polynomtheorie wurde erst 2011 entwickelt [ADT11].

Praktischer Test: Wir haben Orchestra in der Version 0.1-SNAPSHOT getestet. Da dies nicht die aktuellste Version ist, können einige unserer Testergebnisse von denen in der Literatur abweichen. Die aktuellere Version 0.2-SNAPSHOT konnten wir leider nicht fehlerfrei zum Laufen bringen.

Orchestra erstellt für jedes Anfrageergebnis (siehe Abbildung 4.4c) einen passenden Provenance-Graphen. Das Ergebnistupel ist grün und die zugehörigen Quell-Tupel blau hervorgehoben. Die gelben Rauten symbolisieren den Verbund der Quell-Relationen. Jeder Verbund entspricht dabei einer möglichen Konstruktion des Ergebnistuples. Duplikate werden durch Pfeile dargestellt. Für die Anfrage QP_3 bedeutet dies: Das Ergebnistupel entsteht auf drei

[21] Trio-Projektwebiste: `http://infolab.stanford.edu/trio/`
[22] Orchestra-Projektwebsite: `https://www.cis.upenn.edu/~zives/orchestra/`

verschiedene Arten (Pfeile) aus den drei Quell-Relationen `Student`, `Participant` und `Lecturer` (Rauten). Der zugehörige Provenance-Graph entspricht dem Provenance-Polynom $(S_1 \cdot P_1 \cdot L_{1.1}) + (S_1 \cdot P_{20} \cdot L_7) + (S_1 \cdot P_{24} \cdot L_9)$. Die Darstellung weicht jedoch von der in der Literatur ab, da wir die Operatoren + und · anstelle der Pfeile und Rauten nicht rekonstruieren können.

Da Orchestra keine Aggregatfunktionen unterstützt, konnten wir Anfrage QP_4 nicht ausführen. Auch die Auswertung der Selektion auf Nicht-Schlüsselattributen hat nicht immer fehlerfrei funktioniert. Diese Probleme können ebenso wie die „falsche“ Darstellung der Provenance-Graphen jedoch auf die unterschiedlichen Versionen von Orchestra zurückzuführen sein.

DBNotes *DBNotes* [Bha+05; CTV05] ist ein Annotations-Managementsystem für relationale Datenbanken. Es verknüpft jedes Datenelement mit einer oder mehreren Annotationen, welche im weiteren Verlauf nicht weiter verändert werden. DBNotes beantwortet zwar keine der obigen Provenance-Anfrage direkt, die Annotationen können aber für die Interpretation eines Anfrage-Ergebnisses genutzt werden. Sie helfen so die Herkunft der Daten zu verstehen, Sichten zu definieren oder die Qualität der Daten zu bestimmen.

Die Annotationen werden ermöglicht durch die SQL-Erweiterung *pSQL*. Die Autoren stellen insgesamt drei verschiedene Arten von Annotationsverteilungsschemata vor. Neben einem default-Schema sind auch ein default-all-Schema sowie ein benutzerdefiniertes Schema möglich. Während das default-Schema Annotationen lediglich anhand ihrer Herkunft propagiert, berücksichtigt das default-all-Schema zudem äquivalente Anfragetransformationen. Die Wahl des Schemas ist dabei stets vom Anwendungsfall abhängig.

Wie ein Benutzer pSQL zusammen mit DBNotes verwenden kann, um Annotationen auf verschiedene Weise zu verbreiten und abzufragen, wird in [CTV05] vorgestellt. Ein praktischer Test war uns aufgrund des fehlenden Codes an dieser Stelle leider nicht möglich.

Buneman Der Ansatz von Buneman et al. [BCC06] bestimmt und speichert seine Provenance nicht nutzer- bzw. fallabhängig. *Buneman* speichert stattdessen alle möglichen Informationen. Der Artikel zeigt verschiedene Strategien, wie fein granuläre Provenance-Informationen effizient verfolgt, abgespeichert und abgefragt werden können. Hierbei handelt es sich um einen naiven Ansatz sowie zwei Optimierungsmöglichkeiten, im Sinne eines transaktionalen und eines hierarchischen Provenance-Managements.

Mit einem Mehraufwand von 28% pro Aktualisierung ist der Verarbeitungsaufwand des naiven Ansatzes ziemlich hoch. Auch die Menge der gespeicherten Provenance-Informationen steigt proportional zur Größe der geänderten Daten. Die Optimierung reduziert die zusätzlichen Provenance-Kosten auf weniger als 5-10% pro Vorgang. Auch die Speicherkosten sind um den Faktor 5-7 geringer und zudem begrenzt. Details hierzu sowie eine Beschreibung der Implementierung können in [BCC06] nachgelesen werden. Weitere Arbeiten zu diesem Ansatz konnten wir ebenso wie die Autoren von [PRS18] nicht finden. Auch ein praktischer Test war uns aufgrund des fehlenden Codes an dieser Stelle leider nicht möglich.

Perm 2009 als modifizierter PostgreSQL-Server implementiert, handelt es sich bei *Perm*[23] — **P**rovenance **E**xtension of the **R**elational **M**odel — um ein Provenance-Management-System, das Provenance-Anfragen unterschiedlicher Art unterstützt. Dazu wird eine SQL-Anfrage in eine *SQL-PLE*-Anfrage umgeformt. Aufgerufen wird diese transformierte Anfrage durch das Schlüsselwort `PROVENANCE` in der `SELECT` Select-Klausel. Perm kann so den in PostgreSQL existierenden Optimierer zur Anfrageverarbeitung nutzen und generiert zudem alle notwendigen Provenance-Informationen, welche in zusätzlichen Provenance-Spalten gespeichert werden. Die entsprechenden Spaltennamen setzen sich dabei aus dem Präfix *prov*, dem Schemanamen, dem Namen der Quellrelation sowie dem Attributnamen zusammen. Für Anfrage QP_3 ergibt sich beispielsweise `PROV_PUBLIC_GRADES_ID`. Es sind also bereits einige Provenance-Informationen im Spaltennamen selbst enthalten. Neben den zusätzlichen Provenance-Spalten liefert die Perm-GUI noch einen Anfragebaum sowie einen Provenance-Anfragebaum wie in Abbildung 4.4d zu sehen. Insgesamt realisiert Perm somit die ***where***- und die ***why***-Provenance [GA09].

[23] Perm-Projektwebsite: `https://github.com/IITDBGroup/perm`

Praktischer Test: Perm kann sowohl einfache SPJU-Anfragen als auch die klassischen Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM` und `AVG` verarbeiten. Auch Unteranfragen sind möglich. Darüber hinaus verfügt Perm über eine benutzerfreundliche und übersichtliche GUI, wie in Abbildung 4.4d zu sehen ist. Aufgrund der zusätzlichen Provenance-Spalten können Duplikate nicht direkt erkannt werden. Dementsprechend werden sie nicht eliminiert, und das in Perm angegebene Anfrageergebnis ist keineswegs „minimal".

Lipstick Lipstick [Ams+11] ist eines der wenigen Provenance-Systeme, welches auf verschiedenen Provenance-Ebenen arbeitet. Konkret verbindet es datenbank- und workflow-ähnliche Provenance miteinander, indem es die Funktionalität einzelner Komponenten offen legt und so den internen Zustand sowie deren Abhängigkeiten erfasst. Hierfür haben die Autoren einen neuen Provenance-Graphen entwickelt, welcher die Darstellung einer fein granulären Workflow-Provenance ermöglicht. Die Graph-Transformationen *ZoomIn* und *ZoomOut* ermöglichen es zudem, die Granularität der Provenance-Anfrage festzulegen. Details hierzu können in [Ams+11] nachgelesen werden. Es scheint zudem, als wäre das Projekt anschließend nicht weiter verfolgt worden. Es war weder den Autoren von [PRS17; PRS18] noch uns möglich, weitere Arbeiten zu finden, welche neuere Merkmale, Funktionen oder neue Schwerpunkte von Lipstick aufzeigen.

GProM Die *Generic Provenance Middleware* [24] wurde ab 2014 als Nachfolger von Perm entwickelt. Bei GProM handelt es sich um eine Middleware, die verschiedene Datenbankmanagementsysteme und -sprachen wie SQL und Datalog, PostgresSQL, SQLite, Oracle und MonetDB unterstützt. Das System kombiniert deklarative Anfragen mit Provenance-Anfragen und führt den resultierenden SQL-Code auf einem Backend-Datenbanksystem aus. Nach Aussage der Autoren ist GProM das einzige System, welches Provenance für SQL-Anfragen und Transaktionen berechnen kann [Ara+18], indem es für eine Datalog-Anfrage sogenannte *Provenance Games* erstellt und interpretiert. Dazu wird ein passender Provenance-Graph [KLZ13] erstellt und in Abhängigkeit der Fragestellung ***why*** oder ***why not*** eingefärbt. Details können in [Lee+17] nachgelesen werden.

Praktischer Test: Der Code von GProM ist Open Source auf GitHub verfügbar. Als Middleware hat GProM selbst kein User-Interface. Alle erstellten Graphen werden daher automatisch in separaten Dateien abgespeichert. Zudem sind ein zusätzliches Front- sowie ein Backend notwendig. GProM selbst ist einfach zu installieren und gut dokumentiert. Anfragen können sowohl in SQL als auch in Datalog formuliert werden. Dies funktioniert bei uns einwandfrei. GProM hat zudem die einzigartige Funktion *Provenance Game Graphs* zu erstellen (siehe Abbildung 4.4e). Es bietet somit eine Implementierung der ***why***- sowie der ***why not***-Provenance (je nach Färbung). Insgesamt konnten wir alle in der Literatur beschriebenen Funktionen abbilden.

Prov$_{C\&B}$ Das System *Prov*$_{C\&B}$ [25] wurde 2014 von Ileana entwickelt [Ile14]. Es implementiert den provenance-directed Chase&Backchase aus [DH13] zur Lösung des *Answering Queries using Views* (AQuV)-Problems. Ziel von AQuV ist die Bestimmung aller Rewritings einer Anfrage Q, welche nicht auf den Basisrelationen, sondern auf den dem Nutzer zur Verfügung stehenden Sichten arbeiten. Der Algorithmus arbeitet in zwei Schritten: Die Chase-Phase liefert einen universellen Plan $U = Q^{\mathcal{V} \cup I} \mid_{\mathcal{V}}$. Hierfür wird Q zunächst um seine Sichten und Integritätsbedingungen erweitert und anschließend auf die Sichten eingeschränkt. In der Backchase-Phase werden alle Teilanfragen bestimmt, welche ein Rewriting zu Q bilden. Diese kann durch Verwendung der ***why***-Provenance optimiert werden (siehe Abschnitt 3.4.1). So müssen nicht mehr alle möglichen Teilanfragen von U überprüft werden. Die passenden Rewritings können stattdessen direkt aus U abgelesen werden. Provenance reduziert somit den Aufwand der Backchase-Phase drastisch.

Praktischer Test: Der Code von Prov$_{C\&B}$ ist Open Source auf GitHub verfügbar. Das System ließ sich problemlos installieren. Die Bedienung erwies sich jedoch als etwas komplizierter. Es scheint, als wäre Prov$_{C\&B}$ speziell für die in [Ile+14] durchgeführten Experimente entwickelt worden, da die Eingabe lediglich aus drei Zahlen besteht. Da das System die Lösung des Answering Queries usinv Views-Problem zum Ziel hat und Provenance nur zur Effizienzverbesserung intern genutzt wird, sparen wir uns an dieser Stelle die Benchmark-Tests.

[24] GProM-Projektwebiste: `https://github.com/IITDBGroup/gprom`
[25] Prov$_{C\&B}$-Projektwebsite: `https://github.com/IoanaIleana/ProvCB`

ProvSQL Seit 2018 entwickeln Pierre Senellart et al. eine PostgreSQL-Erweiterung namens *ProvSQL*[26], welche ebenso wie Orchestra die Theorie der Provenance Semi-Ringe umsetzt [Sen+18]. Die mittels Annotationen bestimmten Provenance-Polynome werden sowohl als Formel als auch als Provenance-Graph ausgegeben, wobei jede Provenance-ID durch ein selbst gewähltes Alias ersetzt wird. Dies erhöht insbesondere die Lesbarkeit der Polynome und Graphen und macht die ProvSQL-Ausgaben somit für den Nutzer leicht lesbar.

ProvSQL ist das jüngste von uns untersuchte System, dessen Entwicklung bis heute noch nicht abgeschlossen wurde (letzte größere Änderung im Dezember 2021). Es unterstützt verschiedene Semi-Ringe wie den *counting semiring*, den *tropical semiring* oder *positive Boolean functions* (siehe [Sen17; GT17]). Darüber hinaus ist die Spezifikation von benutzerdefinierten Semi-Ringen [Sen+18] möglich. Weitere Funktionalitäten wie die Verarbeitung von Aggregatfunktionen sind laut Aussage der Autoren zu erwarten.

Praktischer Test: Die von ProvSQL vorgenommenen PostgresSQL-Erweiterungen werden als zusätzliche Features bereitgestellt. Hierfür wird zunächst mittels `ADD_PROVENANCE` eine interne Provenance-ID vergeben. Die Besonderheit von ProvSQL liegt in der Darstellung der ***how***-Provenance. So kann der Nutzer selbst entscheiden, über welches Attribut das Provenance-Polynom definiert werden soll. Hierfür wird zunächst ein Mapping definiert, welches anschließend in der Berechnung des Provenance-Polynoms aufgerufen wird. Für unser Studierenden-Beispiel könnte dies wie folgt aussehen:

```
SELECT add_provenance('students');
SELECT create_provenance_mapping('studentID_mapping','students','studentID');
SELECT formula('39371d93-587a-5b89-8e8a-89f58ef6','studentID_mapping');
```

Das zugehörige Ergebnis ist in Abbildung 4.4f zu sehen. Als Alias verwenden wir hier die bereits bekannten Tupel-IDs. Zusammengefasst verarbeitet ProvSQL einfache SQL-Anfragen ohne Probleme. Aggregatfunktionen wie `MAX`, `MIN`, `COUNT`, `SUM` oder `AVG` können aktuell noch nicht verarbeitet werden (Stand Sommer 2022).

4.2.4. Fazit

Mittlerweile existieren in der Literatur mehr als 250 Systeme, welche Provenance-Informationen im engeren oder weiteren Sinne verarbeiten. In diesem Abschnitt haben wir die Funktionalität der sechs bekanntesten Data Provenance-Systeme Tioga, Trio, Orchestra, Perm, GProM und ProvSQL getestet. Wie in Abbildung 4.4 zu sehen ist, bieten alle Systeme außer Tioga eine grafische Provenance-Darstellung. Alle Systeme können einfache SPJ-Anfragen wie QP_1, QP_2 und QP_3 verarbeiten (siehe Tabelle 4.4). Die Verarbeitung von Anfragen mit Aggregatfunktionen wie beispielsweise QP_4 ist hingegen nur in den Systemen möglich, welche nicht auf Provenance-Polynomen basieren. Dies sind Perm, GProM sowie (mit Einschränkungen) Trio.

Trio ermöglicht die Angabe der Data Lineage, eine der ältesten Provenance-Arten. GProM bietet einen Provenance Game-Graphen, der je nach Färbung des Graphen eine Antwort auf die Frage ***why*** oder ***why not*** bietet. Perm gibt alle Provenance-Tupel in zusätzlichen Provenance-Spalten aus. Dies entspricht den Zeugenbasen der ***why***-Provenance. ProvSQL und Orchestra hingegen beantworten die Frage, wie ein Ergebnis berechnet wird, indem sie einen Provenance-Graphen spezifizieren. In ProvSQL wird das zugrundeliegende Provenance-Polynom als eine Formel definiert, die durch einen selbst gewählten Alias repräsentiert wird. Die hieraus resultierende Klassifizierung der verschiedenen Provenance-Typen ist in Tabelle 4.5 zusammengefasst.

Die einzigen Systeme, welch alle vier Benchmark-Anfragen fehlerfrei beantworten sind (mit Einschränkungen) Trio, Perm und GProM. Trio ist aufgrund seines Alters und seiner Fokussierung auf die Data Lineage für unsere weiteren Überlegungen jedoch nicht relevant. Perm und GProM hingegen schon. Beide Systeme benötigen für die Beantwortung der vier Benchmark-Anfragen lediglich die ***where***- bzw. die ***why***-Provenance. Dies nehmen wir uns für die Entwicklung unseres eigenen Systems ProSA zum Vorbild (siehe Abschnitt 10.9) und konzentrieren uns ebenfalls hauptsächlich auf die ***why***-Provenance. Diese hat zudem den Vorteil, dass sie „weitestgehend“ Privacy-konform ist, wie wir in Abschnitt 8.2 zeigen werden.

[26] ProvSQL-Projektwebsite: https://github.com/PierreSenellart/provsql/

Eigene sowie betreute Arbeiten

Die Auswahl der zu testenden Systeme basiert auf den Ergebnissen der Studie von [PRS18], einer Keynote der ProvenanceWeek 2018 sowie verschiedenen Gesprächen mit Provenance-Experten wie V. Tannen und B. Glavic. Alle Systeme liegen in ihrer aktuellsten Version als eigene virtuelle Maschine vor. Die Erstellung der virtuellen Maschinen sowie die Durchführung der Benchmark-Anfragen wurde insbesondere in zwei studentischen Projektarbeiten durchgeführt. Alle in

- [Fla+20a] Flach, Herrmann, Röhrs, Strelnikov: Data Provenance-Tools (Teil I), KSWS[27], SS20
- [FRS20] Flach, Röhrs, Scharlau: Data Provenance-Tools (Teil II), Projekt/NEidI[28], WS20/21

vorgestellten Ergebnisse wurden anschließend geprüft, gegebenenfalls erweitert und korrigiert. Insbesondere bei den Ergebnisse der Literaturanalyse sowie der Aktualisierung der virtuellen Maschinen waren einige Korrekturen notwendig. Die hier vorgestellten Ergebnisse sind zudem im technischen Bericht [AH22b] zu finden.

[27] KSWS: *Komplexe Software Systeme*, Modul an der Universität Rostock im Bachelorstudiengang Informatik
[28] Projekt: Modul an der Universität Rostock im Bachelorstudiengang Informatik
NEidI: *Neueste Entwicklungen in der Informatik*, Modul an der Universität Rostock im Masterstudiengang Informatik

Teil III.

Ergebnisse der Dissertation

5. Formalisierung der Problemstellung

Schauen wir uns nun, nachdem wir in Kapitel 3 die für das Verständnis der Arbeit wichtigsten Begriffe kennengelernt haben, die Problemstellung noch einmal formal an (siehe Abschnitt 5.1) und fassen auch die Ergebnisse der Dissertation noch einmal kurz formal zusammen (siehe Abschnitt 5.2). Auf den ersten Blick mag die Wahl des Chase für die Auswertung von Anfragen, deren Invertierung, die Integration von Provenance-Informationen sowie die Verarbeitung der Evolution etwas ungewöhnlich erscheinen. Sie hat allerdings einige bedeutende Vorteile, welche wir abschließend in Abschnitt 5.3 kurz diskutieren werden.

5.1. Modifizierte technische Problemstellung *(basierend auf [AH18a; Aug18])*

Wie in Abschnitt 2.2 beschrieben, liegt das Ziel der vorliegenden Dissertation in der Bestimmung einer (minimalen) Teil-Datenbank mit deren Hilfe ein gegebenes Anfrageergebnis reproduziert, repliziert oder zumindest auf Plausibilität überprüft werden kann. Die (minimale) Teil-Datenbank soll dabei durch Data Provenance und zusätzliche Annotationen maximiert und bzgl. Privacy-Aspekten minimiert werden. Zudem soll die (minimale) Teil-Datenbank unter (Schema-)Änderungen konsistent bleiben. Insgesamt ergeben sich hieraus die drei bekannten Fragen:

(I.) Wie lässt sich der minimale Teil der ursprünglichen Forschungsdatenbank berechnen, der für die Replizierbarkeit bzw. Reproduzierbarkeit der Auswertungsergebnisse dauerhaft gespeichert werden muss?

(II.) Wie lassen sich die Theorien zu Data Provenance und Schema-Entwicklung vereinheitlichen?

(III.) Wie lassen sich Data Provenance und Privacy-Aspekte,d.h. Aspekte des Datenschutzes, im Falle der Anfrage-Invertierung kombinieren?

Abbildung 5.2 beschreibt den Zusammenhang unserer drei Forschungsfragen, welche die Basis von ProSA bilden. Besonders hervorzuheben sind hier die Beziehungen zwischen I und K zur Berechnung der (minimalen) Teil-Datenbank I^* (Frage (I.)), der Zusammenhang zwischen J und K zur Verarbeitung der Evolution (Frage (II.)) sowie die Integration von Privacy-Aspekten symbolisiert durch das kleine Schloss in I^*_{anon} (Frage (III.)).

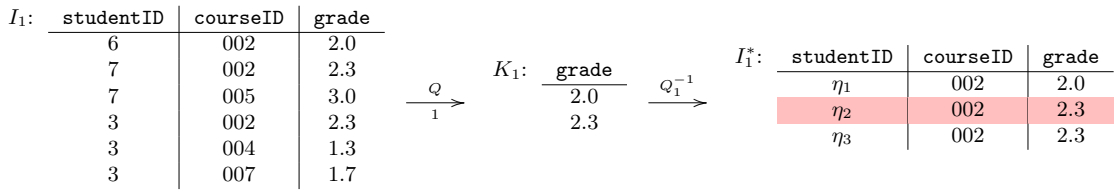
Die Repräsentation der Auswertungsanfrage Q, der Evolution E sowie ihrer Inversen als Menge von (s-t) tgds erlaubt die Verarbeitung beider Anfragen über den Chase (siehe Kapitel 6 und 7). Für die Integration von Data Provenance erweitern wir die Formalisierungen der Inversen durch zusätzliche Provenance-Attribute. Schauen wir uns dies basierend auf [AH18a; Aug18] noch einmal im Detail an.

Berechnung einer (minimalen) Teildatenbank Die rekonstruierte (minimale) Teildatenbank soll die Ergebnisse der Auswertungsanfrage unter den folgenden (alternativen) Randbedingungen rekonstruieren können:

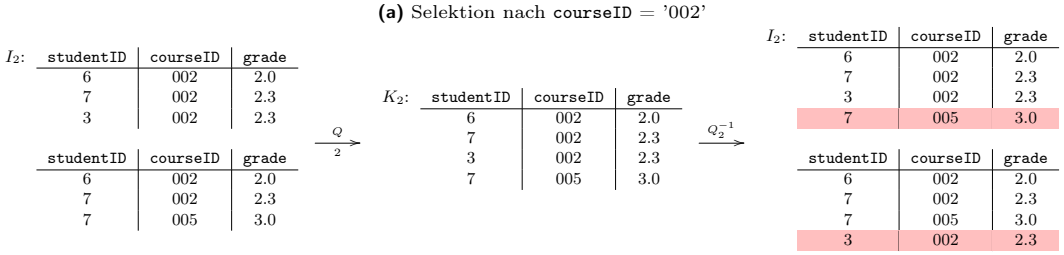
- die Anzahl der relevanten Quell-Tupel bleibt bzgl. der gegebenen Anfrage erhalten[1];
- die (minimale) Teil-Datenbank kann homomorph auf die Original-Datenbank abgebildet werden;
- die Teil-Datenbank bildet eine intensionale Beschreibung (nach Abschnitt 10.7) der Original-Datenbank.

[1] Wir gehen davon aus, dass I bereits auf die für die Auswertung der Anfrage Q relevanten Tupel eingeschränkt ist. Enthält I hingegen Tupel, welche für die Auswertung von Q absehbar nicht benötigt werden, kann $|I^*| = |I|$ niemals garantiert werden.

Konkret stellt sich die Frage, welche zusätzlichen Informationen bei archiviertem Ergebnis K und gemerkter Auswertungsanfrage Q für die Rekonstruktion der (minimalen) Teil-Datenbank I^* benötigt werden. Welchen Teil der (minimalen) Zeugenbasis (***why***-Provenance, [BKT01]) und/oder welche zugehörigen Provenance-Polynome müssen wir uns für die Rekonstruktion der Teil-Datenbank aufheben (***how***-Provenance, [GKT07])? Oder benötigen wir zusätzliche Annotationen — ganze Tupel bzw. Datenbankausschnitte —, welche wir in sogenannten *Side Tables* direkt abspeichern?



(a) Selektion nach courseID = '002'



(b) Vereinigung zweier Tabellen

Abbildung 5.1. Rekonstruktion der (minimalen) Teil-Datenbanken I_i^* für unterschiedliche Anfragen Q.

Doch was verstehen wir überhaupt unter dem Begriff der *(minimalen) Teil-Datenbank*. Sei hierfür erneut die Universitätsdatenbank mit den beiden Relationen `Student` und `Grade` gegeben. Sei weiter I eine der in Abbildung 5.1 gegebenen Quell-Instanzen I_1 bzw. I_2 über dem Verbund der beiden Relationen. Dann entspricht die zu Q_1 rekonstruierte (minimale) Teil-Datenbank einem *Ausschnitt* (siehe Definition 6.2) der originalen Instanz I_1, d.h. es existiert ein Homomorphismus von I_1^* nach I_1 mit $\eta_1 \mapsto 6$, $\eta_2 \mapsto 7$ und $\eta_3 \mapsto 3$ (siehe Kapitel 1). Die zu Q_2 rekonstruierte (minimale) Teil-Datenbank I_2^* hingegen ist sogar „größer“ als die originale Instanz I_2, sodass in diesem Fall kein Homomorphismus von I_2^* nach I_2 existiert. Die (minimale) Teil-Datenbank kann, muss aber keine Teilmenge von I sein. Sie ergibt sich schlicht aus der Anwendung des Chase&Backchase auf eine gegebene Quell-Instanz I sowie eine Auswertungsanfrage Q (siehe Definition 5.1).

Definition 5.1 ((minimale) Teil-Datenbank)**.** Die *(minimale) Teil-Datenbank* ist das Ergebnis der Anwendung des Chase&Backchase auf eine gegebene Quell-Instanz I sowie eine Auswertungsanfrage Q, d.h.

$$I^* = \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) = \textsc{Chase}_{\mathcal{M}^*}(K),$$

wobei $\mathcal{M}$ und $\mathcal{M}^*$ den Schemaabbildungen zu Q und Q^{-1} entsprechen.

Mit der Invertierung der Anfrage Q soll die (minimale) Teil-Datenbank bestimmt werden (siehe Abbildung 5.2, rot hervorgehoben). Maximiert wird diese durch zusätzliche Provenance-Informationen wie Zeugenbasen und Side Tables (Annotationen). Die Inverse Q^{-1} muss dabei nicht zwangsläufig einer Inversen im klassischen Sinne $Q \circ Q^{-1} = \mathrm{Id}$ entsprechen. Grund hierfür ist, dass eine Chase-Inverse nicht in allen Fällen angegeben werden kann [Fag07; Fag+11b]. In den meisten Fällen kann jedoch eine quasi-inverse Chase-Funktion, welche nach Anwendung des Chase auf die Original-Instanz I sowie die mittels Backchase berechnete (minimale) Teil-Datenbank I^* dasselbe Ergebnis liefert, d.h. $\textsc{Chase}_{\mathcal{M}_Q}(I) = \textsc{Chase}_{\mathcal{M}_Q}(I^*)$. So kann für die Projektion etwa eine relaxte Chase-Inverse (siehe Definition 6.5), für die Projektion ohne Duplikateliminierung eine tp-relaxte Chase-Inverse (siehe Definition 6.6) und für die Vereinigung eine ergebnisäquivalente Chase-Inverse (siehe Definition 6.7) angegeben werden [Aug17; AH18b; AH18c]. Der Chase-Inversentyp entscheidet nun, wie viele zusätzliche Informationen für eine möglichst maximale Rekonstruktion von I^* notwendig sind. Wird Q^{-1} durch zusätzliche Provenance erweitert, nennen wir diese auch Q^{-1}_{prov}. Auf die Definition sowie die Bestimmung der verschiedenen Chase-Inversentypen werden wir in Kapitel 5 genauer eingehen.

Ausgewertet werden die Anfrage Q sowie ihre Inverse Q^{-1} bzw. Q^{-1}_{prov} durch Anwendung des Chase. Dafür werden beide Anfragen als Schemaabbildungen der Form $\mathcal{M}_Q = (S_t, T, Q)$ und $\mathcal{M}^*_Q = (T, S_t, Q^{-1})$ interpretiert,

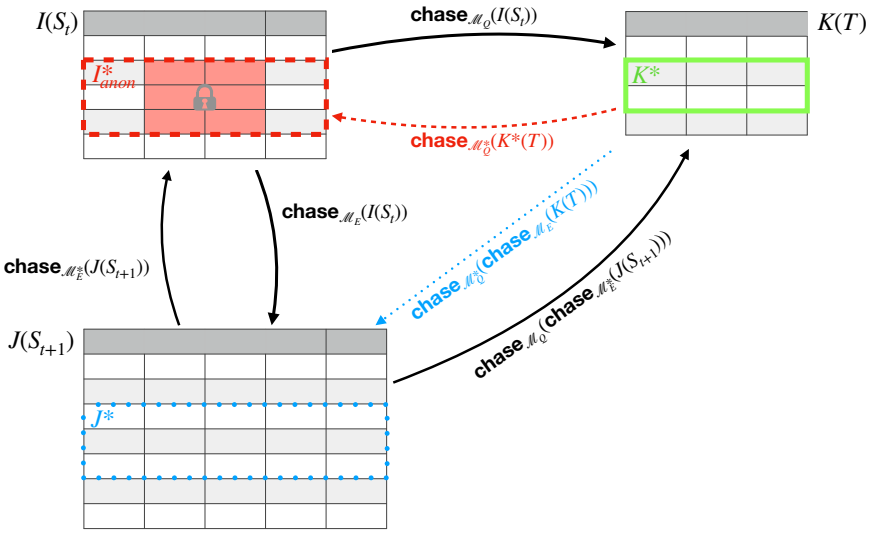


Abbildung 5.2. Chase-Anwendungen bei der Berechnung der (minimalen) Teil-Datenbanken $I^* \preccurlyeq I$ und $J^* \preccurlyeq J$ basierend auf der Ergebnisinstanz $K^* \preccurlyeq K$, der Auswertungsanfrage Q, der Evolution E sowie ihren Inversen Q^{-1} und E^{-1}. Verarbeitet werden die Schemaabbildungen $\mathcal{M}_Q = (I(S_t), K(T), Q)$ und $\mathcal{M}^*_Q = (K(T), I(S_t), Q^{-1})$ zur Berechnung von I^* sowie $\mathcal{M}_E = (I(S_t), J(S_{t+1}), E)$ und $\mathcal{M}^{-1}_E = (J(S_{t+1}, I(S_t), E^{-1}))$ zur Berechnung von J^*.

wobei S_t dem Quell-Schema von I zum Zeitpunkt t und T dem Ziel-Schema zum Anfrageergebnis $Q(I) = K$ entspricht. $\mathcal{M}_Q$ beschreibt die Schemaabbildung zu Q und $\mathcal{M}^*_Q$ die Schemaabbildung zu Q^{-1}. Insgesamt ergibt sich so:

$$\begin{aligned} K(T) &= \textsc{Chase}_{\mathcal{M}_Q}(I(S_t)) \\ I^*(S_t) &= \textsc{Chase}_{\mathcal{M}^*_Q}(K^*(T)) \end{aligned}$$

Für die Hintereinanderausführung beider Anfragen definieren wir in Kapitel 6 einen eigenen (provenance-aware) Chase&Backchase. Dabei soll für alle Anfragen mindestens eine ergebnis-äquivalente Chase-Inverse garantiert werden können. Sollte dies nicht der Fall sein, werden zwangsläufig zusätzliche Provenance-Informationen benötigt. Insgesamt muss also gelten:

$$\textsc{Chase}_{\mathcal{M}_Q}(I) = K = \textsc{Chase}_{\mathcal{M}_Q}(I^*).$$

Vereinheitlichung von Provenance und Evolution Bisherige Provenance-Anfragen Q^{-1}_{prov} (***why***-, ***where***- und ***how***-Provenance) erfolgen stets auf einer gegebenen festen Datenbank sowie einer Auswertungsanfrage Q. Die Kombination von Provenance mit Schema-Evolution soll die Auswertung von Provenance-Anfragen bei sich ändernden Daten und Schemata ermöglichen (siehe Abbildung 5.2). Hierfür muss zunächst die alte Instanz I mittels E^{-1} rekonstruiert, mittels Q ausgewertet (grün hervorgehoben), mittels Q^{-1} oder Q^{-1}_{prov} invertiert (rot hervorgehoben) und anschließend mittels E wieder evolutioniert werden (blau hervorgehoben). Zur Einbeziehung von Provenance nutzten wir Q^{-1}_{prov} in Kombination mit der ***where***-, ***why***- oder ***how***-Provenance.

Zusammengefasst definieren wir eine neue Anfrage Q' als Komposition der ursprünglichen Auswertungsanfrage Q und der inversen Evolution E^{-1}, sodass gilt:

$$Q'(J(S_{t+1})) = (Q \circ E^{-1})(I(S_t)) = E^{-1}(Q(I(S_t))).$$

Die neue Provenance-Anfrage Q'_{prov} ergibt sich analog als Komposition der Evolution E und der alten Provenance-Anfrage Q^{-1}_{prov} mittels $Q'_{\text{prov}}(K^*(T)) = (E \circ Q_{\text{prov}})(K^*(T))$. Es genügt daher, wenn wir uns eine der beiden (minimalen) Teil-Datenbanken $I^*(S_t)$ (rot) oder $J^*(S_{t+1})$ (blau) abspeichern. Die jeweils andere lässt sich mit Hilfe der Inversen leicht berechnen. Im Forschungsdatenmanagement entspricht K^* stets der gesamten Ergebnis-Datenbank K, d.h. $K^* = K$, da das gesamte Ergebnis der wissenschaftlichen Auswertung reproduzierbar bzw. replizierbar sein muss. Allgemeine Provenance-Anfragen können aber auch auf Teilmengen dieses Ergebnisses oder auf einzelne Tupel im Ergebnis angewendet werden.

Ausgewertet werden die neuen Anfragen Q' und Q'_{prov} erneut durch den Chase&Backchase aus Kapitel 6. Hierfür definieren wir zwei weitere Schemaabbildungen der Form $\mathcal{M}_E = (S_t, S_{t+1}, E)$ und $\mathcal{M}^*_E = (S_{t+1}, S_t, E^{-1})$, welche die Evolution E und ihre Inverse E^{-1} beschreiben. Hierbei entspricht S_t dem Schema über I zum Zeitpunkt t und S_{t+1} dem Schema über J zum Zeitpunkt $t+1$. Dann ergibt sich:

$$\begin{aligned} K(T) &= \text{Chase}_{\mathcal{M}_Q}(\text{Chase}_{\mathcal{M}^*_E}(J(S_{t+1}))) \\ J^*(S_{t+1}) &= \text{Chase}_{\mathcal{M}^*_Q}(\text{Chase}_{\mathcal{M}_E}(K(T))). \end{aligned}$$

Provenance und Privacy Der Privacy-Begriff geht für uns über den Schutz personenbezogener Daten hinaus. Wir sehen alle Arten von Forschungsdaten als schützenswert, insbesondere, wenn diese bestimmten Privacy-Policies unterliegen. Gesucht ist daher eine Anonymisierung I^*_{anon} der (minimalen) Teil-Datenbank I^*, welche wiederum nach Anwendung der Auswertungsanfrage dasselbe bereits bekannte Ergebnis K oder manchmal auch nur ein anonymisiertes Ergebnis, das in einem noch zu definierenden Sinn "kompatibelßum publizierten Ergebnis ist, liefert. Wir werden für die Anonymisierung unterschiedliche Methoden und Maße untersuchen. Unseren Schwerpunkt legen wir aber auf der Generalisierung durch den Einsatz von Generalisierungshierarchien als Anonymisierungsmethode sowie k-Anonymität und l-Diversität als Anonymisierungsmaße. Die Integration von Privacy erfolgt nach Durchführung des Chase&Backchase. Den Chase nutzten wir an dieser Stelle lediglich zur Überprüfung der Korrektheit unserer Anonymisierung, d.h.:

$$\text{Chase}_{\mathcal{M}_Q}(I) = K = \text{Chase}_{\mathcal{M}_Q}(I^*_{\text{anon}}).$$

ProSA Die Kombination der Ergebnisse zu (I.), (II.) und (III.) ist Hauptaufgabe der praktischen Umsetzung, d.h. der Implementierung, von ProSA. Hierfür rekonstruieren wir zunächst die Original-Instanz I aus einer gegebenen Sicht J und berechnen das Anfrageergebnis $K = Q(I)$ sowie die (minimale) Teil-Datenbank I^*. Diese wird anschließend zu I^*_{anon} anonymisiert, bevor sie abschließend zu J^* evolutioniert wird. Dabei muss gelten:

$$\begin{aligned} \text{Chase}_{\mathcal{M}_{Q'}}(J^*) &= \text{Chase}_{\mathcal{M}_{Q'}}(J) \\ &= \text{Chase}_{\mathcal{M}_Q}(\text{Chase}_{\mathcal{M}^*_E}(J)) \\ &= \text{Chase}_{\mathcal{M}_Q}(I) \\ &= K \end{aligned} \qquad \begin{aligned} \text{Chase}_{\mathcal{M}_Q}(I) &= \text{Chase}_{\mathcal{M}_Q}(I^*) \\ &= \text{Chase}_{\mathcal{M}_Q}(I^*_{\text{anon}}) \\ &= K \end{aligned}$$

Eine ausführliche Beschreibung des ProSA-Konzeptes sowie des Systems selbst erfolgt in Kapitel 10.

5.2. Zusammenfassung der Ergebnisse *(basierend auf [Aug20])*

Um die verschiedenen Theorien zu vereinheitlichen, stellen wir die Auswertungsanfrage, die zugehörigen Provenance-Anfragen sowie die Evolution als Menge von (s-t) tgds dar (siehe Definition 3.13) — einer Verallgemeinerung der klassischen Verbundabhängigkeiten — und verarbeiten diese mit dem Chase (siehe Definition 3.22). Der Chase verarbeitet somit sowohl die Auswertung (Evaluation) als auch die Evolution der Anfrage. In vielen Fällen können Evaluation und Evolution zudem komponiert und in einem einzigen Chase-Schritt ausgeführt werden. Details hierzu können in [Kas22] nachgelesen werden.

In einem zweiten Schritt, dem so genannten Backchase, verwenden wir den Chase erneut. Zur Verarbeitung der invertierten Anfrage — in Kombination mit zusätzlichen Provenance-Informationen auch *Provenance-Anfrage* genannt — und generieren eine Instanz auf Grundlage des zuvor berechneten Anfrageergebnisses. Diese Instanz ist Teilmenge der originalen Instanz, welche bzgl. Provenance maximiert und Privacy-Aspekten minimiert wird. Es entsteht die sogenannte *(minimale) Teil-Datenbank*.

Grundlage für unsere Theorien sind unter anderem die Arbeiten von Fagin et al. Zaniolo et al. sowie Green et al. So basiert unsere Inversen-Theorie auf [Fag+11b] und [Fag+09]. Für die Bestimmung der relevanten Schema-Modifikationsoperatoren berufen wir uns auf [Cur+08]. Grundlage für unsere Provenance-Theorie sind schließlich [BKT01; CCT09] und [GKT07]. Die Wahl unseres Anonymisierungsmaßes sowie der passenden Methode basiert

auf den Arbeiten [GLS14; PS17; Sch22]. Für die konkret gewählten Maße und Verfahren sei insbesondere auf [Sam01; Swe01] verwiesen.

In diesem Abschnitt wollen wir die Ergebnisse der Dissertation kurz zusammenfassen. Grundlage hierfür sind im Rahmen der Promotion entstandene Artikel (siehe Anhang F), betreute studentische Projekte und Abschlussarbeiten (siehe Anhang F.2). Eine detaillierte Beschreibung der Ergebnisse folgt in den Kapiteln 6 bis 10. Die wesentlichen Inhalte der oben erwähnten Artikel werden in den Grundlagen sowie dem State of the Art in Kapitel 3 vorgestellt.

Berechnung der (minimalen) Teil-Datenbank Sei I eine Datenbankinstanz, $\mathcal{M} = (S_t, T, \Sigma_Q)$ eine durch eine Menge Σ_Q von (s-t) tgds definierte Schemaabbildung und $I^* = \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I))$ die durch zweimalige Anwendung des Chase berechnete (minimale) Teil-Datenbank. Während eine exakte Chase-Inverse immer die ursprüngliche Datenbankinstanz I rekonstruiert, erfordern schwächere Varianten nur Datenaustauschäquivalenz sowie die Existenz eines Homomorphismus von I^* nach I. Die (minimale) Teil-Datenbank I^* muss also nicht zwangsläufig in I enthalten sein. Ein Homomorphismus, welcher die in I^* enthaltenen Nullwerte auf passende Konstanten in I abbildet, ist vollkommen ausreichend. So kann beispielsweise das Tupel $(1, 3, \eta_1) \in I^*$ auf das Tupel $(1, 3, 5) \in I$ abgebildet werden. Das heißt, wenn I^* keine (markierten) Nullwerte enthält und alle Tupel aus I^* auf Tupel aus I abgebildet werden können, ist I^* (minimale) Teil-Datenbank von I.

Neben der exakten, der klassischen sowie der relaxten Chase-Inversen, welche ab 2005 eingeführt wurden, benötigen wir zwei weitere Chase-Varianten. Um die Anzahl der Tupel aus I zu erhalten, definieren wir eine *tupelerhaltende relaxte (tp-relaxete)* Chase-*Inverse*. Und um auch für aggregierte Funktionen eine Chase-Inverse spezifizieren zu können, definieren wir eine *ergebnisäquivalente* Chase-*Inverse* [AH18c]. Beide Definitionen basieren auf der Theorie von Fagin et al. [Fag+11b]. Dabei gilt:

$$\text{ergebnisäquivalent} \preccurlyeq \text{relaxt} \preccurlyeq \text{tp-relaxt} \preccurlyeq \text{exakt}.$$

Die ergebnisäquivalente Chase-Inverse ist somit die schwächste und die exakte Chase-Inverse die stärkste Inverse. Die passenden notwendigen und hinreichenden Bedingungen folgen in Abschnitt 6.1 sowie 6.3.

Wir können für jede Grundoperation wie $\pi, \bowtie, \sigma$ oder `AVG` eine exakte ($=$), (tp-)relaxte ($\preccurlyeq_{(\text{tp})}$) oder ergebnisäquivalente ($\leftrightarrow$) Chase-Inverse angegeben werden. Durch zusätzliche Provenance-Informationen kann der Inversentyp in einigen Fällen verbessert werden. So kann für die Projektion statt einer relaxten Inversen eine tp-relaxte Chase-Inverse angegeben werden. Für andere Operationen, wie etwa der Selektion kann der Inversentyp trotz zusätzlicher Informationen jedoch nicht verbessert werden [AH18c]. Für die Bestimmung des Chase-Inversentyps müssen die (s-t) tgds jedoch zunächst invertiert werden. In Abschnitt 6.4 werden wir hierzu einen passenden Algorithmus vorstellen, den (provenance-aware) Chase&Backchase.

Um den Informationsverlust möglichst gering zu halten, ergänzen wir die Invertierung durch zusätzlichen Provenance-Informationen wie Zeugenbasen, Provenance-Polynome und/oder weitere Annotationen. Basierend auf den Ergebnissen definieren wir vier Klassen von Operatoren, klassifiziert nach der Existenz von Dangling Tuples und Duplikaten, Provenance-Invarianz sowie weiteren Operatoren. Die formalen Beweise können hierzu können im Anhang C nachgelesen werden.

Vereinheitlichung von Provenance und Evolution Langfristige, datengestützte Studien sind in vielen Bereichen der Wissenschaft unverzichtbar geworden. Oft ändern sich im Laufe der Zeit die Datenformate, Strukturen und die Semantik der Daten, die Datensätze entwickeln sich weiter. Daher müssen insbesondere Studien, die sich über mehrere Jahrzehnte erstrecken, wechselnde Datenbankschemata berücksichtigen. Die Evolution dieser Datenbanken führt irgendwann zu einer großen Anzahl von Schemata, die kosten- und zeitintensiv gespeichert und verwaltet werden müssen. Im Sinne der Reproduzierbarkeit von Forschungsdaten muss jedoch jede Datenbankversion mit geringem Aufwand rekonstruierbar sein. So kann ein bereits veröffentlichtes Ergebnis jederzeit validiert und reproduziert bzw. repliziert werden. Dennoch kann in vielen Fällen eine solche Entwicklung nicht vollständig rekonstruiert werden.

Gegeben seien zwei Instanzen I und $J = E(I)$, wobei J durch Evolution aus I entstanden ist. Die Schemaänderung kann formal durch sogenannte *Schema-Modifikationsoperatoren* beschrieben werden. Wir bestimmen

hierfür zunächst die Operatoren, welche im Forschungsdatenmanagement am häufigsten auftreten. Hierzu gehören unter anderem `DECOMPOSE Table`, `JOIN Table`, `MERGE Table` und `MERGE Column`. Dabei konnten wir in einer Bachelorarbeit in Zusammenarbeit mit dem Leibniz-Institut für Ostseeforschung Warnemünde (IOW) feststellen, dass viele Schemamodifikationen Merging- oder Splitting-Operationen enthalten [Man20]. Wir definieren die Modifikationsoperatoren `MERGE Column` und `SPLIT Column` als Hintereinanderausführung von `ADD Column` und `DROP Column`-Operatoren. Dies mag im ersten Moment unnötig erscheinen, macht im Falle des IOWs aber 17% der Operationen aus. Die Ergebnisse hierzu sind in [Aug+20] zusammengefasst. Wir werden zudem in Abschnitt 7.2 detailliert auf die Ergebnisse der Zusammenarbeit mit dem IOW eingehen.

Um eine Aussage über die Entwicklung der (minimalen) Teil-Datenbank treffen zu können, haben wir die Modifikationsoperatoren auf ihre Chase-Inversen-Typen untersucht. Auf diese Weise können wir prüfen, ob unsere bestimmte Teil-Datenbank realistisch, d.h. plausibel ist. Auch diese können in die gleichen vier Klassen wie die Auswertungsanfragen eingeteilt werden (siehe Abschnitt 7.3 und [AH22a]). Die zugehörigen Beweise können wiederum im Anhang C nachgelesen werden.

Die Invertierung einiger Operatoren bedarf dabei etwas mehr Fingerspitzengefühl. So müssen neben der Invertierung der Evolution selbst auch die (Quasi-)Invertierbarkeit potentieller Hilfsfunktionen sowie die Änderung der zugrundeliegenden Datentypen berücksichtigt werden. So kann beispielsweise die Zusammenführung der Attribute `tag`, `monat` und `jahr` zu einem gemeinsamen `datum` durch eine erneute Aufteilung des Datums invertiert werden. Hierbei muss sichergestellt werden, dass der ursprüngliche Attributwert korrekt wiederhergestellt wird und der zugehörige Datentyp durch die Invertierung nicht verloren geht. Dies passiert insbesondere dann, wenn die mathematische Invertierung und die implementierte Invertierung voneinander abweichen. In diesen Fällen kann es beispielsweise durch Runden zu unsauberen Daten kommen. Um dieses Problem zu lösen, definieren wir in Abschnitt 7.4 die sogenannte ***what***-*Provenance*.

Durch die Formalisierung von Evolution und Evaluation, d.h. der Auswertungsanfrage, sowie ihrer Invernsen als Schemaabbildungen sind wir in der Lage, Datenauswertungen mit Provenance in temporalen Datenbankstrukturen zu kombinieren. Dies ermöglicht die Reproduzierbarkeit, Replizierbarkeit oder zumindest den Plausibilitätsnachweis wissenschaftlicher Ergebnisse über längere Zeiträume. Alle Ergebnisse hierzu werden in Kapitel 7 ausführlich diskutiert.

Privacy im Falle von Anfrage-Invertierung Für uns geht der Begriff *Privacy* über die klassische Definition des (meist personenbezogenen) Datenschutzes hinaus. Vielmehr beziehen wir uns auf den Schutz von Daten im Allgemeinen. Wir unterscheiden somit zwei Varianten der Privacy: (1) Sensible, evtl. personenbezogene (Forschungs-) Daten, welche anonymisiert werden müssen und (2) Daten, welche eine spezielle Intellectual Property Preservation erfüllen müssen. Die Gründe für den Schutz von Forschungsdaten sind vielfältig. Diese können wirtschaftlich, persönlich oder finanziell sein, da die Erstellung solcher Daten oft zeit- und kostenaufwendig ist.

Um ein Gefühl für die Relevanz von Provenance und Privacy im Alltag der Rostocker Wissenschaftler zu bekommen, haben wir im Rahmen einer Bachelorarbeit [Sch20] eine Interview-Studie mit 21 Wissenschaftlern und Nicht-Wissenschaftlern verschiedener Fachgebiete durchgeführt. Neben ihrem Verständnis von Provenance und Privacy waren wir zudem am Forschungsdatenmanagement unserer Teilnehmer interessiert. Die Ergebnisse der Interviews sind in [ASH21b] noch einmal neu interpretiert und zusammengefasst worden.

Wie in [Aug20] beschrieben, sollen bei der Bestimmung der (minimalen) Teil-Datenbank nur solche Tupel rekonstruiert werden, die nicht im Widerspruch zu Privacy-Aspekten stehen. Je nach gewählter Data Provenance sind unterschiedliche Privacy-Probleme zu beachten [ASH21a]. Sei hierfür I eine Datenbankinstanz und I^* die zugehörige (minimale) Teil-Datenbank. Dann gilt: (1) Bei der Verwendung von Relationennamen oder Provenance-IDs (***where***-Provenance) sind in der Regel nicht genügend schützenswerte Daten vorhanden. Solange mithilfe der IDs nicht wieder auf die originalen Daten aus I zugegriffen wird, kann aus Relationenname und Provenance-ID für I^* lediglich das Schema sowie die Anzahl der ursprünglichen Tupel rekonstruiert werden (siehe hierzu Kapitel 6). Privacy-Aspekte können in diesem Fall vernachlässigt werden. Wird hingegen das originale Tupel mit abgespeichert, kann dies zu Privacy-Problemen führen. (2) Auch bei der ***why***-Provenance kann es durch die Speicherung der Zeugenbasen zu Privacy-Problemen kommen. Dies gilt in den wenigen Sonderfällen, wenn die Varianz der Attributwert-Verteilung gleich Null ist. (3) Die ***how***-Provenance geht mit ihren Provenance-Polynomen noch ein

wenig weiter. Sie berechnet oft zu viele wiederherstellbare Informationen, so dass Privacy-Aspekte bei diesem Ansatz sehr häufig ein Problem darstellen. Wir beschränken uns in ProSA, wie wir in Kapitel 10 vorstellen werden, daher auf die ***why***-Provenance. Sollten weitere Informationen notwendig sein, behelfen wir uns mit zusätzlichen Side Tables.

Für die Lösung des Problems betrachten wir verschiedene Ansätze wie Generalisierung, Unterdrückung, Permutation von Attributwerten oder Differential Privacy. Auch intensionale, d.h. beschreibende Attributwerte sind möglich. Letztere werden in Kapitel 8 jedoch nur kurz diskutieren. Stattdessen betrachten wir die Anonymisierung von ***where***-, ***why***- und ***how***-Provenance bzgl. der eben genannten Anonymisierungsmethode. In Kapitel 10 wählen wir uns schließlich ein Anonymisierungsmaß, eine Anonymisierungsmethode sowie einen passenden Zeitpunkt, in der wir die Anonymisierung der (minimalen) Teil-Datenbank im ProSA-Ablauf durchführen können. Erste Ideen hierzu sind zudem in [Sch22] sowie [AHH22] zu finden.

Verallgemeinerung des Chase Anwendung findet der Chase, wie in Abschnitt 4.1 vorgestellten Systeme zeigen beispielsweise in der semantischen Optimierung, Data Exchange und Data Integration, Query Rewriting, Answering Queries using Views oder dem Data Cleaning. Dafür werden stets Anfragen oder Instanzen verarbeitet. Eine Zusammenfassung der bekanntesten Chase-Varianten ist beispielsweise in [Ben+17] oder [GMS12] zu finden. Wir haben eine Verallgemeinerung des Chase auf (beliebige) Objekte $\bigcirc$ und Parameter $*$ entwickelt, welche in Kapitel 9 ausführlich diskutiert wird. Erste Ansätze hierzu sind in [AH19] zu finden.

Die konkrete Implementierung von ChaTEAU ist Teil verschiedener studentischer Abschlussarbeiten [Jur18; Ren19; Zim20; Gör20] und Projekte [Alb+21] sowie durch unsere Hilfskräfte Jakob Zimmer und Nic Scharlau. Zusammengefasst werden die Theorie sowie der aktuelle Stand der Implementierung im Demo-Paper [Aug+22]. Nennenswerte Arbeiten mit Bezug zu ChaTEAU sind zudem [Ros20; Zim21].

Der verallgemeinerte Chase arbeitet auf einem (beliebigen) Objekt sowie einem (beliebigen) Parameter. Dabei umfasst das Chase-Objekt $\bigcirc$ sowohl Instanzen I als auch Anfragen Q. Mehr noch, ChaTEAU verarbeitet allgemeine (s-t) tgds, wobei eine Anfrage Q als Menge spezieller (s-t) tgds aufgefasst werden kann. In beiden Fällen — Anfragen oder Instanzen — können Nullwerte/Variablen durch andere Nullwerte/Variablen sowie Konstanten ersetzt werden. Die Variablensubstitution hängt dabei von bestimmten Bedingungen ab, welche unter anderem in [AH19; Aug+22] beschrieben werden.

Der Chase-Parameter $*$ entspricht einer Datenbank-internen oder -übergreifenden Abhängigkeit, dargestellt als Menge von (s-t) tgds und egds. Hierzu gehören neben klassischen Verbund- (JD) und Inklusionsabhängigkeiten (IND) auch funktionale Abhängigkeiten (FD). Die vordefinierte Hierarchie der Abhängigkeiten JD $\preccurlyeq$ tgd $\preccurlyeq$ s-t tgd und FD $\preccurlyeq$ egd erlaubt es, alle Abhängigkeiten als (s-t) tgds bzw. egds aufzufassen. Auch Sichten können als solche Abhängigkeiten dargestellt werden. Abhängigkeiten zur Verarbeitung von Data Cleaning sowie Anfragen können ebenfalls als Menge von (s-t) tgds und egds formalisiert werden. Dies ermöglicht die Verwendung eines allgemeinen Parameters für Anwendungen wie die semantische Optimierung, Answering Queries using Views, Data Exchange und Data Integration sowie Data Cleaning, Query Rewriting und mehr.

Ziel dieses Kapitels ist die Entwicklung eines verallgemeinerten Chase-Algorithmus zur Verarbeitung eines allgemeinen Chase-Objekts $\bigcirc$ sowie einem Parameter $*$, um so verschiedene Anwendungsfälle abdecken zu können. Neben der theoretischen Entwicklung soll dieser zudem praktisch umgesetzt werden. Beides ist im Demo-Artikel [Aug+22] kurz zusammengefasst.

ProSA — Provenance management using Schema mappings with Annotations ProSA vereint die bisherigen Ergebnisse zu einem gemeinschaftlichen Konzept, sodass die (minimale) Teil-Datenbank bzgl. Provenance maximiert und bzgl. Privacy minimiert auch für sich ändernde Datenbanken berechnet werden kann. Hierfür wird in Kapitel 10 zunächst der konzeptuelle Ablauf von ProSA vorgestellt. Dieser sieht vor, dass die Anfragen zwar in SQL gestellt, intern aber als Menge von (s-t) tgds und egds verarbeitet werden. Hierfür werden zwei Algorithmen zur Transformation sowie zur Invertierung der Anfrage vorgestellt. Anschließend erweitern wir unseren Ansatz zur Einbindung von einfachen Aggregatfunktionen von `MAX`, `MIN`, `COUNT`, `SUM` und `AVG` sowie Gruppierung. Für die Einbindung von Privacy-Aspekten wählen wir zunächst eine passende Anonymisierungmethode sowie ein passendes Anonymisierungsmaß. Unsere Wahl fällt hierbei auf die Generalisierung in Kombination mit der

k-Anonymität. Alle Erweiterungen werden abschließend in einem verallgemeinerten (Provenance-aware) Chase&Backchasezusammengefasst.

Die Implementierung von ProSA erfolgt in Java als eigenständiges Programm, welches ChaTEAU zur Ausführung des Chase zweimal aufruft. Die Einbindung von Data Provenance gelingt durch eine Erweiterung von ChaTEAU. Auch wenn für ProSA entwickelt, wird die Provenance selbst nicht in ProSA direkt eingearbeitet, da sonst eine Verarbeitung über den Chase nicht möglich wäre. Die Interpretation der Provenance-Ergebnisse ist hingegen Teil von ProSA und nicht von ChaTEAU.

Die Konzeptentwicklung von ProSA sowie die Entwicklung der verschiedenen Algorithmen ist Schwerpunkt der vorliegenden Arbeit und wird in den Kapiteln 5 bis 8 sowie 10 ausführlich vorgestellt. Die Implementierung hingegen ist großteils in verschiedenen studentischen Abschlussarbeiten [Kav22; Sch22; Han22; Spo22] und Projekten [För+21; RSZ21] sowie durch unsere Hilfskräfte Nic Scharlau und Jakob Zimmer entstanden.

5.3. Abgrenzung zu anderen Ansätzen

Auf den ersten Blick mag die Wahl des Chase für die Auswertung von Anfragen, deren Invertierung sowie die Verarbeitung von Provenance-Anfragen und der Evolution etwas ungewöhnlich erscheinen. Sie hat allerdings einige bedeutende Vorteile:

- Sowohl die Schema-Evolution als auch einfache Auswertungsanfragen können als Schemaabbildungen interpretiert und mit dem Chase-Algorithmus [Fag+11b] verarbeitet werden. Somit können beide Abbildungen mit der gleichen Theorie behandelt werden.
- Auch die zugehörigen Inversen können als Menge von (s-t) tgds geschrieben und mit Hilfe des Chase verarbeitet werden.
- Die Invertierung von Schemaabbildungen wurde bereits in den 2010er Jahren von Fagin et al. ausführlich untersucht. Ihre Theorie über Quasi-Inverse von Schemaabbildungen sowie deren Komposition kann in [Fag+08; Fag+11b] nachgelesen werden.
- Eine Analyse der Inversentypen verschiedener Auswertungsoperatoren findet sich beispielsweise in [AH18c]. Sie kann durch zusätzliche Provenance-Informationen wie Zeugenbasen und Annotationen verbessert werden. Dies ermöglicht die Rekonstruktion von verlorenen Null-Tupeln und Duplikaten sowie die Ersetzung von Nullwerten durch konkrete Werte.

Die Interpretation von Evolution und Evaluation, d.h. der Auswertungsanfrage, als Schemaabbildung ermöglicht die einheitliche Verarbeitung beider Schritte über den Chase. Die Anwendung des Chase&Backchase (siehe Kapitel 6, 7 und 10) liefert so auch bei sich ändernden Datenbanken eine (minimale) Teil-Datenbank (siehe Abbildung 1.4). Die Komposition von Evolution und Auswertungsanfrage ermöglicht die Berechnung der (minimalen) Teil-Datenbank aus jeder beliebigen Sicht. Eine Versionierung im klassischen Sinne ist bei unserem Ansatz somit nicht notwendig. Hierfür hat die *Research Data Alliance*[2] (RDA) Working Group on Dynamic Data Citation[3] 14 Empfehlungen veröffentlicht, welche sich auf „die Zeitstempelung und Versionierung sich entwickelnder Datenquellen und die dynamische Identifizierung von Teilmengen über dauerhafte Identifikatoren konzentrieren, die den Anfragen zur Auswahl der jeweiligen Teilmengen zugewiesen werden“ [Rau+21]. Stattdessen können wir die (minimale) Teil-Datenbank im alten Datenbank-Zustand (rot) mittels Evolution in den neuen Zustand (blau) und umgekehrt überführen. Solange eine der beiden (minimalen) Teil-Datenbanken bekannt ist, kann die jeweils andere Datenbankinstanz ohne großen Aufwand rekonstruiert werden und eine aufwendige Versionierung entfällt.

Eine Versionierung via Git ist ebenfalls denkbar, aufgrund der regelmäßigen, aber verhältnismäßig seltenen Schemaänderungen in einigen Anwendungsfällen aber nicht notwendig bzw. zielführend. Da alle Änderungen gleichzeitig und nur alle paar Jahre stattfinden — zumindest im Fall der Sauerstoff-Daten am IOW — kann die Original-Instanz auch über mehrere Versionen hinweg problemlos rekonstruiert werden, sofern die Evolutionsschritte E_i bekannt sind. Der hiermit einhergehende Informationsverlust ist verhältnismäßig klein und wird von

[2] RDA: https://rd-alliance.org
[3] Working Group on Dynamic Data Citation: https://www.rd-alliance.org/groups/data-citation-wg.html

uns in Kauf genommen. Grund hierfür ist, dass archivierenswerte Daten im Laufe der Zeit immer nur mehr, aber niemals weniger werden [Aug+20]. Existierende Daten werden — im Kontext des Forschungsdatenmangements — also niemals weggeschnitten oder gelöscht, sodass jeglicher Verlust an Tupeln oder Attributen, welcher durch die Invertierung der Evolution entsteht, kein Verlust im eigentlichen Sinne ist. Die „zusätzlichen“ Daten aus S_{t+1} existierten in S_t überhaupt nicht.

Auch den Privacy-Begriff nutzten wir anders als üblich. Privacy, zu Deutsch Privatsphäre, beschreibt „das Recht einer Person, ihre persönlichen Angelegenheiten und Beziehungen geheim zu halten[4].“ Wir fassen den Begriff Privacy jedoch etwas weiter. Neben persönlichen Daten gelten für uns Forschungsdaten im Allgemeinen als schützenswert. Wir stützen uns hierbei unter Anderem auf die Ergebnisse einer Interview-Studie sowie verschiedenen Policies unterschiedlicher Forschungseinrichtungen, welche wir in Kapitel 8 vorstellen werden. An dieser Stelle grenzen wir uns bewusst von den existierenden Definitionen ab.

Eigene sowie betreute Arbeiten

Die in diesem Kapitel vorgestellte Problemstellung basiert auf den folgenden Veröffentlichungen:

- [AH18a] **Tanja Auge**, Andreas Heuer: Combining Provenance Management and Schema Evolution. *ProvenanceWeek*, 2018
- [Aug18] **Tanja Auge**: Exposé eines Promotionsprojektes: Provenance Management für Data-Science-Anwendungen unter Berücksichtigung von Daten- und Schema-Evolution. *Exposé*, 2018
- [Aug20] **Tanja Auge**: Extended Provenance Management for Data Science Applications. *PhD@VLDB*, 2020

[4] Cambirdge Dictionary: https://dictionary.cambridge.org/dictionary/english/privacy

6. Berechnung der (minimalen) Teil-Datenbank

Für eine gegebene Datenbankinstanz I sowie eine Menge von Abhängigkeiten Σ liefert der Chase eine Ergebnisinstanz K, welche Σ genügt. Im Falle von ProSA entspricht Σ dabei der Formalisierung einer Auswertungsanfrage Q — meist angegeben als relationenalgebraische Anfrage oder als SQL-Anfrage — und K dem zugehörigen Anfrageergebnis. Dabei kann die Auswertung insbesondere bei Operationen, welche die Daten aggregieren wie beispielsweise der Projektion oder der Summenbildung `SUM`, schwierig sein. In vielen Fällen ist eine vollständige Invertierung daher gar nicht möglich. Stattdessen beschränken wir uns auf die sogenannten *Quasi-Inversen*.

In ProSA unterscheiden wir fünf verschiedene (quasi-)inverse Schemaabbildungen, welche durch den Chase erzeugt werden, die sogenannten Chase-*Inversen* (siehe Abschnitt 6.1). Wie eine Menge von (s-t) tgds invertiert werden kann, stellen wir anschließend in Abschnitt 6.2 vor. In Abschnitt 6.3 bestimmen wir schließlich die Inversentypen der wichtigsten Auswertungsoperationen π, σ, $\bowtie$, `SUM`, usw. Bis auf wenige Ausnahmen kann dabei für alle Operationen ein Chase-Inversentyp angegeben werden, welcher durch zusätzliche Provenance-Informationen beeinflusst werden kann, wie in Tabelle 6.4 zu sehen ist. Abschließend stellen wir in Abschnitt 6.4 den kompletten Ablauf zur Bestimmung der (minimalen) Teil-Datenbank mit Hilfe eines Chase&Backchase-Verfahrens vor. Hierbei wird eine als Menge von (s-t) tgds formulierte Anfrage Q mit Hilfe von Chase auf einer gegebenen Instanz I ausgewertet und anschließend in einem zweiten Schritt, dem Backchase, invertiert. Insgesamt erhalten wir so eine (minimale) Teil-Datenbank I^*, welche es uns ermöglicht das Anfrageergebnis $Q(I)$ zu rekonstruieren [AH18a; AH18c]. Zu beachten ist, dass die (minimale) Teil-Datenbank I^* keine echte Teilmenge von I sein muss und kein Homomorphismus von I^* nach I existieren muss. Die Teil-Datenbank I^* entspricht lediglich einer (möglicherweise falsch) rekonstruierten Version von I, für welche jedcoh gilt: $Q(I^*) = Q(I)$.

6.1. Ergebnisäquivalente und tp-relaxte Chase-Inverse *(basierend auf [Aug17; AH18c])*

Die für die Rekonstruktion der (minimalen) Teil-Datenbank nötigen inversen Schemaabbildungen sind bereits bzgl. ihrer Invertierungseigenschaften untersucht und im Bereich der Schemaevolution erfolgreich angewendet worden [Fag07]. Wir wollen diese nun auch zur Auswertung von Anfragen sowie zur Verarbeitung von Provenance-Anfragen nutzen. Hierfür wird die inverse Abbildung durch zusätzliche Annotationen erweitert, welche während der Anfrageauswertung erhoben werden (siehe Abschnitt 6.3).

Sei hierzu $\mathcal{M} = (S, T, \Sigma)$ eine Schemaabbildung von einem Schema S in ein zweites Schema T bzgl. einer Menge von Abbhängigkeiten Σ (siehe Definition 3.16). Die zugehörige inverse Schemaabbildung $\mathcal{M}^* = (T, S, \Sigma^{-1})$ entspricht dann der Rückabbildung des Ziel-Schemas T auf das Quell-Schema S bzgl. einer gegebenen Menge von Abhängigkeiten Σ^{-1}. Gilt $\mathcal{M} \circ \mathcal{M}^* \neq \mathrm{Id}$, so sprechen wir von einer Quasi-Inversen [Fag+08], welche lediglich einen Teil der originalen Daten „ordnungsgemäß“ invertiert. Vergleichen lässt sich dies beispielsweise mit der Quadratfunktion $g : \mathbb{Z} \to \mathbb{N}$ mit $x \mapsto x^2$, welche alle ganzen Zahlen quadriert. Da negative Zahlen im klassischen Sinne keine Wurzel besitzen, liefert ihre Quasi-Inverse $g^{-1} : \mathbb{N} \to \mathbb{N}$ mit $x \mapsto \sqrt{x}$ keine negativen Werte mehr. Eine solche Quasi-Inverse (siehe Definition 6.1) reicht für unsere Zwecke jedoch völlig aus.

Definition 6.1 (Quasi-Inverse, [SS83])**.** Sei $f : X \to Y$ eine Funktion von X nach Y. Dann ist g *Quasi-Inverse* von f, wenn gilt:

- $g : Z \to X$ mit $ran(f) \subseteq Z \subseteq Y$ für den Bildbereich $ran(f)$ von f, und
- $f \circ g \circ f = f$, wobei $\circ$ die funktionale Komposition[1] bezeichnet. ■

[1] Ausführung von links nach rechts: $(f \circ g \circ f)(x) = f(g(f(x)))$

Wir konzentrieren uns im Folgenden auf eine spezielle Form von Inversen, welche in [Fag+09] erstmals eingeführt wurde und eine „klare operationelle Semantik hat, die auf dem Begriff Chase basiert und sie aus praktischer Sicht attraktiv macht“ [Fag+11b]. Die sogenannten Chase-*Inversen* können verwendet werden, um die ursprünglichen Daten so gut wie möglich und mit geringstmöglichen Informationsverlust wiederherzustellen. Dabei ist sowohl eine exakte als auch eine teilweise Rekonstruktion der Quell-Daten möglich. Neben den von Fagin et al. eingeführten Chase-inversen Funktionen — *exakte*, *klassische* und *relaxte* Chase-*Inverse* — definieren wir noch zwei weitere Typen von Chase-Inverse, welche für die Berechnung der (minimalen) Teil-Datenbank relevant sind, die *tp-relaxte* und *ergebnisäquivalente* Chase-*Inverse*.

Chase-Inverse auf Datenbanken Wie bereits beschrieben, beschränken wir uns in ProSA auf die Chase auf Instanzen. Anwendungsfälle für den Chase auf Anfragen können beispielsweise bei unseren Systemtests in Abschnitt 4.1 nachgelesen werden. Sei also $\mathcal{M} = (S, T, \Sigma)$ eine Schemaabbildung mit Quell-Schema S, Ziel-Schema T sowie einer Menge von Abhängigkeiten Σ. Sei weiter I eine gegebene Quell-Instanz über S. Dann liefert die Anwendung des Chase eine Ergebnisinstanz $K = \text{Chase}_{\mathcal{M}}(I)$. Eine erneute Anwendung des Chase auf K und $\mathcal{M}^* = (T, S, \Sigma^{-1})$ liefert eine (minimale) Teil-Datenbank $I^* = \text{Chase}_{\mathcal{M}^*}(K) = \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I))$. Diese doppelte Anwendung des Chase ist eine spezielle Form des Chase&Backchase-Verfahren (siehe Abbildung 6.1). In Abschnitt 6.4 werden wir das Verfahren noch einmal genauer beleuchten.

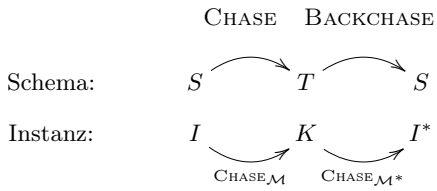


Abbildung 6.1. Chase&Backchase auf Datenbanken.

Enthält die rekonstruierte (minimale) Teil-Datenbank I^* ein Tupel mit (markierten) Nullwerten, deren verbleibende Attributwerte mit den Attributwerten eines Tupels der Quell-Instanz I übereinstimmen (z.B. $(1, 3, 5) \in I$ und $(1, 3, \mu_1) \in I^*$), so nennen wir I^* einen *Ausschnitt* der Instanz I (kurz: $I^* \preccurlyeq I$). In diesem Fall existiert einen Homomorphismus h, welcher die Tupel von I^* auf die Tupel von I abbildet. Für unser Beispiel-Tupel gilt etwa: $1 \mapsto 1, 3 \mapsto 3$ und $\mu_1 \mapsto 5$. Wenn I^* keine (markierten) Nullwerte enthält und alle Tupel aus I^* auch Tupel in I sind, schreiben wir $I^* \subseteq I$.

Definition 6.2 (Teil-Instanz)**.** Seien I und J zwei Datenbankinstanzen. Seien weiter $t_1 \in I$ und $t_2 \in J$ zwei gegebene Tupel, wobei t_2 Nullwerte enthalten darf. Existiert ein Homomorphismus $h : J \to I$ mit $h(t_2) = t_1$, dann heißt J *Ausschnitt* bzw. *Teil-Instanz* von I. Wir schreiben $J \preccurlyeq I$. ■

Mit dieser Definition können wir uns nun den verschiedenen Chase-Inversentypen widmen. Seien auch hierfür I und I^* zwei Instanzen über dem Schema S. Fortan bezeichnen wir mit I^* das Ergebnis des Chase&Backchase, d.h. $I^* = \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I))$.

Exakte, klassische und relaxte Chase-Inverse Während eine *exakte* Chase-*Inverse* (siehe Definition 6.3) stets die Original-Datenbank rekonstruiert, liefert die *klassische* Chase-*Inverse* (siehe Definition 6.4) nur ein zur Original-Datenbank äquivalentes Ergebnis [Fag+11b]. Kann keine klassische oder exakte Chase-Inverse angegeben werden, kann auf die sogenannte *relaxte* Chase-*Inverse* (siehe Definition 6.5) zurückgegriffen werden [Fag+08]. Diese liefert eine zur Original-Instanz homomorphe Teil-Instanz [Fag+11b].

Definition 6.3 (exakte Chase-Inverse, [Fag+11b])**.** Sei $\mathcal{M}$ eine s-t tgd von einem Schema S auf ein zweites Schema T. Dann heißt $\mathcal{M}^*$ *exakte* Chase-*Inverse* von $\mathcal{M}$ (kurz: $=$), wenn $\mathcal{M}^*$ eine s-t tgd von T auf S ist, sodass für jede Instanz I über S gilt:

$$I = \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)).$$

■

Definition 6.4 (klassische CHASE-Inverse, [Fag+11b]). Sei $\mathcal{M}$ eine s-t tgd von einem Schema S auf ein zweites Schema T. Dann heißt $\mathcal{M}^*$ *klassische* CHASE-*Inverse* von $\mathcal{M}$ (kurz: $\equiv$), wenn $\mathcal{M}^*$ eine s-t tgd von T auf S ist, sodass für jede Instanz I über S gilt:

$$I \equiv \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)).$$

■

Definition 6.5 (relaxte CHASE-Inverse, [Fag+11b]). Sei $\mathcal{M}$ eine s-t tgd von einem Schema S auf ein zweites Schema T. Sei weiter I eine beliebige gegebene Instanz und $I^* = \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I))$. Dann heißt $\mathcal{M}^*$ *relaxte* CHASE-*Inverse* von $\mathcal{M}$ (kurz: $\preccurlyeq$), wenn gilt:

- $\mathcal{M}^*$ ist s-t tgd von T nach S;
- $\forall I : I^* \leftrightarrow_{\mathcal{M}} I$, d.h. I^* und I sind ergebnisäquivalent bzgl. $\mathcal{M}$;
- $\forall I : \exists$ Homomorphismus $I^* \to I$, d.h. I^* ist Ausschnitt von I (kurz: $I^* \preccurlyeq I$). ■

Schauen wir uns abschließend die Definitionen der exakten sowie der relaxten CHASE-Inversen noch einmal am konkreten Beispiel an. Zu beachten ist, dass erneut alle $\exists$-quantifizierten Variablen groß und alle $\forall$-quantifizierten Variablen klein geschrieben werden.

Beispiel 6.1. Seien `Student` und `Grade` die bereits bekannten Relationen aus dem Studierenden-Beispiel eingeschränkt auf `Student`(`studentID`,`name`,`studies`) und `Grade`(`courseID`,`studentID`,`grade`). Sei weiter `Join`(`name`, `courseID`,`grade`) ein möglicher Verbund von `Student` und `Grade` und $I_1 = \{\texttt{Join}(\text{Donald}, 001, 2.0)\}$ sowie $I_2 = \{\texttt{Student}(1, \text{Donald}, \text{Teaching}), \texttt{Grade}\ (001, 1, 2.0)\}$ zwei beliebige Instanzen. Seien Σ sowie Σ^{-1} wie folgt gegeben:

$$\begin{aligned}
\Sigma \quad &= \quad \{\texttt{Join}(\texttt{name}, \texttt{courseID}, \texttt{grade}) \\
&\qquad \to \exists\ \texttt{StudentID}, \texttt{Studies} : \texttt{Student}(\texttt{StudentID}, \texttt{name}, \texttt{Studies}) \wedge \texttt{Grade}(\texttt{courseID}, \texttt{StudentID}, \texttt{grade})\} \\
\Sigma^{-1} \quad &= \quad \{\texttt{Student}(\texttt{studentID}, \texttt{name}, \texttt{studies}) \wedge \texttt{Grade}(\texttt{courseID}, \texttt{studentID}, \texttt{grade}) \\
&\qquad \to \texttt{Join}(\texttt{grade}, \texttt{courseID}, \texttt{grade})\}
\end{aligned}$$

Dann ist Σ^{-1} exakte CHASE-Inverse zu Σ und Σ relaxte CHASE-Inverse zu Σ^{-1}. Denn:

$$\begin{aligned}
I_1^* &= \text{CHASE}_{\Sigma^{-1}}(\text{CHASE}_{\Sigma}(I_1)) \\
&= \text{CHASE}_{\Sigma^{-1}}(\text{CHASE}_{\Sigma}(\{\texttt{Join}(\text{Donald}, 001, 2.0)\}) \\
&= \text{CHASE}_{\Sigma^{-1}}(\{\texttt{Student}(\eta_1, \text{Donald}, \eta_2), \texttt{Grade}(001, \eta_1, 2.0)\}) \\
&= \{\texttt{Join}(\text{Donald}, 001, 2.0)\} \\
&= I_1 \\
I_2^* &= \text{CHASE}_{\Sigma}(\text{CHASE}_{\Sigma^{-1}}(I_2)) \\
&= \text{CHASE}_{\Sigma}(\text{CHASE}_{\Sigma^{-1}}(\{\texttt{Student}(1, \text{Donald}, \text{Teaching}), \texttt{Grade}(001, 1, 2.0))\})) \\
&= \text{CHASE}_{\Sigma}(\{\texttt{Join}(\text{Donald}, 001, 2.0)\}) \\
&= \{\texttt{Student}(\eta_1, \text{Donald}, \eta_2), \texttt{Grade}(001, \eta_1, 2.0)\} \\
&\preccurlyeq I_2
\end{aligned}$$

□

Tp-relaxte und ergebnisäquivalente Chase-Inverse Wie in [Fag+11b] beschrieben, ist die relaxte CHASE-Inverse eine Abschwächung der klassischen CHASE-Inversen. Sie fordert keine Äquivalenzbeziehung zwischen Quellinstanz und (minimaler) Teil-Datenbank I^*, dafür aber Ergebnisäquivalenz [Fag+11b] sowie die Existenz eines Homomorphismus von I^* nach I. Um die Tupelzahl der Original-Datenbank beizubehalten, erweitern wir die Definition der relaxten CHASE-Inversen zur sogenannten *tp-relaxte* CHASE-*Inversen* (siehe Definition 6.6). Ein Beispiel hierfür sind etwa Σ und Σ^{-1} aus Beispiel 6.1.

Definition 6.6 (tp-relaxte CHASE-Inverse, [AH18c])**.** Seien S und T zwei Schemata und $\mathcal{M}$ eine s-t tgd von S nach T. Sei weiter $I^* = \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I))$ die rekonstruierte (minimale) Teil-Datenbank einer beliebige gegebenen Instanz I. Dann heißt $\mathcal{M}^*$ *tp-relaxte* CHASE-*Inverse* von $\mathcal{M}$ (kurz: $\preccurlyeq_{\text{tp}}$), wenn gilt:

- Tupelanzahl der Quell- und der rekonstruierten (minimalen) Teil-Datenbank stimmen überein, d.h. $|\,I^*\,| = |\,I\,|$.
- $\mathcal{M}^*$ ist s-t tgd von T nach S;
- $\forall I : I^* \leftrightarrow_{\mathcal{M}} I$, d.h. I^* und I sind ergebnisäquivalent bzgl. $\mathcal{M}$
- $\forall I : \exists$ homomorphe Abbildung $I^* \to I$, d.h. I^* ist Abschnitt von I (kurz: $I^* \preccurlyeq I$) ■

Eine Abschwächung der relaxten CHASE-Inversen ist hingegen die *ergebnisäquivalente* CHASE-*Inverse* (siehe Definition 6.7). Seien hierfür I und J zwei Instanzen über S und $\mathcal{M} = (S, T, \Sigma)$ eine gegebene Schemaabbildung. Dann heißt I *ergebnisäquivalent bzgl.* J (kurz: $I \leftrightarrow_{\mathcal{M}} J$), wenn gilt:

$$\text{CHASE}_{\mathcal{M}}(I) \leftrightarrow \text{CHASE}_{\mathcal{M}}(J).$$

Dies ist beispielsweise der Fall, wenn I und I^* beide Nullwerte enthalten. So liefert die Anwendung des CHASE für $I_3 = \{\texttt{Student}(1, \text{Donald}, \eta_1), \texttt{Grade}(001, 1, 2.0)\}$ im obigen Beispiel

$$\begin{aligned} I_3^* &= \text{CHASE}_{\Sigma}(\text{CHASE}_{\Sigma^{-1}}(I_3)) \\ &= \text{CHASE}_{\Sigma}(\text{CHASE}_{\Sigma^{-1}}(\{\texttt{Student}(1, \text{Donald}, \eta_1), \texttt{Grade}(001, 1, 2.0))\})) \\ &= \text{CHASE}_{\Sigma}(\{\texttt{Join}(\text{Donald}, 001, 2.0)\}) \\ &= \{\texttt{Student}(\eta_2, \text{Donald}, \eta_3), \texttt{Grade}(001, \eta_2, 2.0)\} \\ &\neq I_3. \end{aligned}$$

sowie

$$\text{CHASE}_{\Sigma^{-1}}(I_3) = \texttt{Join}(\text{Donald}, 001, 2.0) = \text{CHASE}_{\Sigma^{-1}}(I_3^*).$$

Definition 6.7 (ergebnisäquivalente CHASE-Inverse)**.** Seien S und T zwei Schemata und $\mathcal{M}$ eine s-t tgd von S nach T. Sei weiter $I^* = \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I))$ die rekonstruierte (minimale) Teil-Datenbank einer beliebige gegebenen Instanz I. Dann heißt $\mathcal{M}^*$ *ergebnisäquivalente* CHASE-*Inverse* von $\mathcal{M}$ (kurz: $\leftarrow$), wenn gilt:

- $\mathcal{M}^*$ ist s-t tgd von T und S;
- $\forall I : I^* \leftrightarrow_{\mathcal{M}} I$, d.h. I^* und I sind ergebnisäquivalent bzgl. $\mathcal{M}$ (kurz: $I^* \leftrightarrow I$). ■

Reduktion der Chase-Inversen Die relaxte CHASE-Inverse ist eine Abschwächung der klassischen CHASE-Inversen. Sie fordert keine Äquivalenzbeziehung zwischen der Quell-Instanz I und der rekonstruierten Teil-Datenbank I^*, sondern lediglich die Datenaustauschäquivalenz sowie die Existenz eines Homomorphismus von I^* nach I. Die tp-relaxte CHASE-Inverse erhält in I^* zudem die Tupelanzahl von I bei. Sie ist somit eine Erweiterung der relaxten CHASE-Inversen. Schränken wir die relaxte CHASE-Inverse hingegen auf die Datenaustauschäquivalenz ein, erhalten wir eine ergebnisäquivalente CHASE-Inverse. Insgesamt ergibt sich folgende Reduktion:

$$\text{ergebnisäquivalent} \preccurlyeq \text{relaxt} \preccurlyeq \text{tp-relaxt} \preccurlyeq \text{exakt}.$$

Diese Reduktion bildet die hinreichenden Bedingungen der verschiedenen CHASE-Inversentypen. Die notwendigen Bedingungen ergeben sich aus der jeweiligen Definition der CHASE-inversen Funktion. Tabelle 6.1 fasst die verschiedenen Bedingungen noch einmal zusammen. Die Existenz einer exakten CHASE-Inversen ist somit hinreichende Bedingung für die Existenz einer klassischen CHASE-Inversen. Notwendige Bedingungen ist die Existenz zweier Homomorphismen $h : I^* \to I$ sowie $h' : I \to I^*$ usw.

Die klassische CHASE-Inverse wird in unseren weiteren Untersuchungen weitestgehend vernachlässigt. Grund hierfür ist, dass beim Auftreten einer klassischen CHASE-Inversen in der Regeln auch eine exakte CHASE-Inverse angegeben werden kann (siehe Abschnitt 6.3 und Abschnitt 7.3). Für die klassische CHASE-Inverse existieren daher (nach unseren Bewertungskriterien) kaum Anwendungsfälle.

CHASE-Inverse	hinr. Bedingung	notw. Bedingung
exakt ($=$)	$I^* = I$	$I^* = I$
klassisch ($\equiv$)	$=$	$\exists$ Homomorphismus $h : I^* \to I$ und $h' : I \to I^*$
tp-relaxte ($\preccurlyeq_{\text{tp}}$)	$=$	$\exists$ Homomorphismus $h : I^* \to I, \lvert I^* \rvert = \lvert I \rvert, I^* \leftrightarrow_{\mathcal{M}} I$
relaxt ($\preccurlyeq$)	$\equiv$ oder $\preccurlyeq_{\text{tp}}$	$\exists$ Homomorphismus $h : I^* \to I, I^* \leftrightarrow_{\mathcal{M}} I$
ergebnisäquivalent ($\leftrightarrow$)	$\preccurlyeq$	$I^* \leftrightarrow_{\mathcal{M}} I$

Tabelle 6.1. Hinreichende und notwendige Bedingungen für die Existenz von CHASE-Inversen.

6.2. Invertierung von (s-t) tgds

Für die Invertierung einer gegebenen Anfrage Q wird diese zunächst als Menge von (s-t) tgds interpretiert, einzeln invertiert und anschließend zu einer gemeinsamen inversen s-t tgd zusammengefasst. Sei also Q eine Anfrage bestehend aus verschiedenen (s-t) tgds $\forall\ x : (\phi(x) \to \exists y : \psi(x, y))$. Bei der Invertierung vertauschen wir jeweils Body und Head der einzelnen (s-t) tgds und ergänzen $\exists$-Quantoren für alle freien Variablen im Head. Gleichheitsprädikate der Form $x = c$ mit x Variable und c Konstante werden aufgelöst, indem x in allen Relationen des Heads der invertierten s-t tgd durch c ersetzt wird. Die so ersetzten Variablen werden nicht $\exists$-quantifiziert. Insgesamt entsteht so für jede Abhängigkeit eine neue, inverse (s-t) tgd $\forall\ x, y : (\psi(x, y) \to \exists z : \gamma(x|_\forall, z, y))$ mit $z = x|_\exists$, welche abschließend durch Komposition zu einer Gesamt-Inversen zusammengefasst werden. Dabei beschreibt $x|_\forall$ alle x-Variablen, welche sowohl in ϕ als auch in ψ $\forall$-quantifiziert sind. Alle weiteren x-Variablen sind in γ $\exists$-quantifiziert. Wir bezeichnen diese mit $x|_\exists$. Die Inversen der wichtigsten Basis-Operatoren wie die Projektion, der natürliche Verbund, die Selektion oder die Mengenoperatoren sind in Tabelle 6.2 aufgelistet. Eine vollständige Liste sowie weitere Spezialfälle wie beispielsweise die Bereichsselektion sowie ein `Full Outer Join` können in [Spo22] nachgelesen werden.

Operation	s-t tgd	Inverse
Projektion	$R(a_1, ..., a_n) \to \texttt{Result}(a_i)$	$\texttt{Result}(a_i) \to \exists A_1, ..., A_{i-1}, A_{i+1}, ..., A_n : R(A_1, ..., A_{i-1}, a_i, A_{i+1}, ..., A_n)$
Selektion	$R(a_1, ..., a_n) \wedge a_k \theta c \to \texttt{Result}(a_1, ..., a_k, ..., a_n)$ $R(a_1, ..., a_n) \wedge a_k \theta a_l \to \texttt{Result}(a_1, ..., a_k, ..., a_l, ..., a_n)$	$\texttt{Result}(a_1, ..., a_k, ..., a_n) \to R(a_1, ..., a_k, ..., a_n)$ $\texttt{Result}(a_1, ..., a_k, ..., a_l, ..., a_n) \to R(a_1, ..., a_k, ..., a_n)$
Verbund	$R(a_1, ..., a_n, c) \wedge S(b_1, ..., b_m, c) \to \texttt{Result}(a_1, ..., a_n, c, b_1, ..., b_m)$	$\texttt{Result}(a_1, ..., a_n, c, b_1, ..., b_m) \to R(a_1, ..., a_n, c) \wedge S(b_1, ..., b_m, c)$
Vereinigung	$R(a_1, ..., a_n) \to \texttt{Result}(a_1, ..., a_n)$ $S(a_1, ..., a_n) \to \texttt{Result}(a_1, ..., a_n)$	$\texttt{Result}(a_1, ..., a_n) \to R(a_1, ..., a_n)$ $\texttt{Result}(a_1, ..., a_n) \to S(a_1, ..., a_n)$
Schnitt	$R(a_1, ..., a_n) \wedge S(a_1, ..., a_n) \to \texttt{Result}(a_1, ..., a_n)$	$\texttt{Result}(a_1, ..., a_n) \to R(a_1, ..., a_n) \wedge S(a_1, ..., a_n)$
Differenz	$R(a_1, ..., a_n) \wedge \neq S(a_1, ..., a_n) \to \texttt{Result}(a_1, ..., a_n)$	$\texttt{Result}(a_1, ..., a_n) \to R(a_1, ..., a_n)$
Umbenennung	$R(a_1, ..., a_j, ..., a_n) \wedge a_j = x \to \texttt{Result}(a_1, ..., x, ..., a_n)$	$\texttt{Result}(a_1, ..., x, ..., a_n) \to R(a_1, ..., a_j, ..., a_n) \wedge a_j = x$

Tabelle 6.2. Basis-Auswertungsoperatoren und ihre Inversen.

Schauen wir uns die Invertierung zunächst am Beispiel an. Anschließend erweitern wir den Algorithmus 2 zu Algorithmus 3 und bestimmen die Inverse erneut.

Beispiel 6.2. Sei für die Veranschaulichung die SQL-Anfrage `SELECT lastname FROM Student NATURAL JOIN Grade WHERE grade = '1.3'` gegeben. Diese kann beispielsweise als Menge der drei (s-t) tgds

$$\texttt{Student}(\texttt{studentID}, \texttt{lastname}) \wedge\ \texttt{Grade}(\texttt{studentID}, \texttt{grade}) \to \texttt{Interim}_1(\texttt{studentID}, \texttt{lastname}, \texttt{grade})$$
$$\texttt{Interim}_1(\texttt{studentID}, \texttt{lastname}, \texttt{grade}) \wedge\ \texttt{grade} = 1.3 \to \texttt{Interim}_2(\texttt{studentID}, \texttt{lastname}, 1.3)$$
$$\texttt{Interim}_2(\texttt{studentID}, \texttt{lastname}, 1.3) \to \texttt{Result}(\texttt{lastname})$$

bestehend aus den Basis-Operationen Verbund, Selektion und Projektion interpretiert werden.

Diese werden nun in umgekehrter Reihenfolge einzeln invertiert:

$$\begin{aligned} \texttt{Result}(\texttt{lastname}) &\rightarrow \exists \texttt{StudentID}, \texttt{Grade} : \texttt{Interim}_2(\texttt{StudentID}, \texttt{lastname}, \texttt{Grade}) \\ \texttt{Interim}_2(\texttt{studentID}, \texttt{lastname}, 1.3) &\rightarrow \texttt{Interim}_1(\texttt{studentID}, \texttt{lastname}, 1.3) \\ \texttt{Interim}_1(\texttt{studentID}) &\rightarrow \texttt{Student}(\texttt{studentID}, \texttt{lastname}) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{grade}) \end{aligned}$$

und abschließend wie folgt komponiert:

$$\texttt{Result}(\texttt{lastname}) \rightarrow \exists \texttt{StudentID} : \texttt{Student}(\texttt{StudentId}, \texttt{lastname}) \wedge \texttt{Grade}(\texttt{StudentId}, 1.3).$$

Dabei ist zu beachten, dass einmal ∃-quantifizierte Variablen wie die `studentID` auch in der Gesamt-Inversen ∃-quantifiziert sein müssen. Auch Konstanten wie die 1.3 werden soweit möglich beibehalten. □

Diese vereinfacht dargestellte Methode basiert auf dem *Maximum Extended Recovery-Algorithmus* von Fagin et al. [Fag+11a], welcher die Anwendung von Nullwerten in der Quell-Datenbank erlaubt und daher ideal für die in ProSA vorliegende Invertierung geeignet ist, da diese durch Datenintegration sowie Evolution jederzeit entstehen können. Die durch Fagins Algorithmus entstehenden *second order tgds* (so-tgds) können dabei leicht vermieden und durch s-t tgds erster Ordnung repräsentiert werden, wie in [Zim21] gezeigt. Sei hierfür $\mathcal{M} = (S, T, \Sigma)$ eine gegebene Schemaabbildung mit Quell-Schema S, Ziel-Schema T sowie einer Menge von (s-t) tgds Σ. Dann liefert der *Direct Adapted Max Extended Recovery-Algorithmus*, siehe Algorithmus 2, eine invertierte Schemaabbildung $\mathcal{M}^* = (T, S, \Omega)$, die sogenannte *Adapted Strong Maximum Extended Recovery*, wobei Ω eine Menge von Abhängigkeiten erster Ordnung ist. Die hierfür benötigten *disjunktiven Mengen* definiert Zimmer wie in Definition 6.8 beschrieben. Details hierzu können in [Zim21] nachgelesen werden.

Algorithm 2 Direct Adapted Max Extended Recovery($\mathcal{M}$), [Zim21]

Require: Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ mit Σ endliche Menge von s-t tgds
Ensure: Adapted Strong Maximum Extended Recovery $\mathcal{M}^* = (T, S, \Omega)$ von $\mathcal{M}$ mit Ω Menge von disjunktiven Mengen in Logik erster Ordnung

1. Normalisierung der Konjunktionen in den s-t tgd-Heads:
   Erstelle Σ' aus Σ, indem jede s-t tgd $\alpha \rightarrow (\beta_1 \wedge ... \wedge \beta_r)$ in Σ durch s-t tgds der Form $\alpha \rightarrow \beta_1, ..., \alpha \rightarrow \beta_r$ ersetzt wird, bei denen β_i im Head atomar ist.

2. Invertierung der s-t tgds:
   Füge für jede normalisierte s-t tgd $\alpha \rightarrow \beta$ in Σ' die s-t tgd $\beta \rightarrow \alpha$ zur Menge Σ'' hinzu. In Σ'' wird jede Variable in β zu einer ∀-quantifizierten Variable, während jede Variable in α, die nicht in β vorkommt, zu einer ∃-quantifizierten Variable wird.

3. Sammlung der s-t tgds in disjunktiven Mengen:
   Sei $\Omega = \{\omega_1, ..., \omega_q\}$ eine Menge von disjunktiven Mengen (q Anzahl der Relationsschemata in R), über denen Ausdrücke in den Bodies der s-t tgds in Σ'' existieren. Für jedes R in den Ausdrücken der Bodies der s-t tgd entsteht ein $\omega_i \in \Omega$, indem die s-t tgds in Σ' im Body-Ausdruck zusammengefasst werden.

Definition 6.8 (Disjunktive Menge, [Zim21])**.** Eine *disjunktive Menge* ist eine Menge von s-t tgds, deren Bodies aus je einem Ausdruck bestehen, wobei jeder dieser Body-Ausdrücke über demselben Relationenschema definiert ist. Die Variablen in dem Bodies der einzelnen s-t tgds müssen dabei nicht miteinander übereinstimmen. ■

Der Direct Adapted Max Extended Recovery-Algorithmus liefert als Inverse stets eine Menge von disjunktiven Mengen, welche die invertierten Abhängigkeiten bzw. s-t tgds als mehrere (Teil-)Abhängigkeiten in unterschiedlichen Mengen darstellt. Für die Verarbeitung der disjunktiven Mengen ist jedoch eine Anpassung des Chase notwendig [Zim21]. Da ProSA auf dem Standard-Chase basiert, ist somit eine weitere Anpassung des Invertierungs-Algorithmus notwendig, welche die disjunktiven Mengen vermeidet. Algorithmus 3 zeigt eine leicht modifizierte Version des hierfür von Spolwind entwickelten Algorithmus [Spo22]. Hauptunterschied ist, dass wir neben s-t tgds auch tgds zulassen. Aber schauen wir uns dies zunächst am Beispiel an.

Algorithm 3 Invertierung in ProSA, nach [Spo22]

Require: Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ mit Σ Menge von (s-t) tgds
Ensure: Invertierte Schemaabbildung $\mathcal{M}^* = (T, S, \Omega)$ von $\mathcal{M}$ mit Ω Menge von (invertierten) (s-t) tgds

1. Normalisierung der Konjunktion in den Heads der Abhängigkeiten:
   Normalisiere alle Abhängigkeiten in Σ', indem alle Formeln der Form $\alpha \rightarrow (\beta_1 \wedge ... \wedge \beta_n)$ durch eine Menge von Formeln $\alpha \rightarrow \beta_1, ..., \alpha \rightarrow \beta_n$ ersetzt werden.

2. Invertierung der Abhängigkeiten:
   Invertiere die normalisierten (s-t) tgds indem $\alpha \rightarrow \beta$ durch $\beta \rightarrow \alpha$ ersetzt wird. Sei Ω' die resultierende Menge, welche alle Inversen enthält.

3. Ersetzen der s-t tgd-Heads:
   Seien zwei (s-t) tgds der Form $\beta \rightarrow (\alpha_1 \wedge ... \wedge \alpha_i \wedge ... \wedge \alpha_n)$ und $\alpha_i \rightarrow \gamma$ in Ω' gegeben. Dann ersetze die beiden Formeln durch $\beta \rightarrow (\alpha_1 \wedge ... \wedge \gamma \wedge ... \wedge \alpha_n)$ sowie alle α_i durch γ.

4. Entfernen von Duplikaten im s-t tgd-Head:
   Sei eine (s-t) tgd der Form $\beta \rightarrow (\alpha_1, ..., \alpha_i, \alpha_i, ..., \alpha_n)$ mit Duplikat α_i gegeben, dann entferne alle bis auf ein α_i.

5. Entfernen der negierten Relationen:
   Sei $\beta \rightarrow (\alpha_1 \wedge ... \wedge \neg\alpha_i \wedge ... \wedge \alpha_n) \in \Omega'$ eine gegebene Abhängigkeit, dann entferne die negierte Relation $\neg\alpha_i$ aus dem Head der (s-t) tgd.

6. Komposition der einzelnen Inversen:
   Seien $\beta \rightarrow \alpha$ und $\gamma \rightarrow \alpha$ zwei s-t tgds mit gleichem Head, dann ersetze beide durch $(\beta \wedge \gamma) \rightarrow \alpha$.

7. Anwendung der Gleichheitsprädikate:
   Gegeben sei eine Abhängigkeit der Form $\beta \rightarrow (\alpha \wedge x = c)$ oder $\beta \rightarrow y$. Ersetze alle Variablen x durch die Konstante c bzw. durch die Variable y und entferne das Gleichheitsprädikat $x = c$ bzw. $x = y$.

8. Anpassung der Quantoren:
   Alle Variablen einer s-t tgd, die im Head, aber nicht im Body vorkommen, werden $\exists$-quantifiziert. Alle Variablen, welche durch einen konkreten Wert ersetzt wurden, werden nicht quantifiziert. Alle restlichen Variablen werden all-quantifiziert.

Beispiel 6.3 (Fortsetzung)**.** Betrachten wir erneut die Anfrage aus Beispiel 6.2, interpretiert als drei (s-t) tgds:

$$\begin{aligned}
\texttt{Student}(\texttt{studentID}, \texttt{lastname}) \wedge \texttt{Grade}(\texttt{studentID}, g) &\rightarrow \texttt{Interim}_1(\texttt{studentID}, \texttt{lastname}, \texttt{grade}) \\
\texttt{Interim}_1(\texttt{studentID}, \texttt{lastname}, \texttt{grade}) \wedge \texttt{grade} = 1.3 &\rightarrow \texttt{Interim}_2(\texttt{studentID}, \texttt{lastname}, 1.3) \\
\texttt{Interim}_2(\texttt{studentID}, \texttt{lastname}, 1.3) &\rightarrow \texttt{Result}(\texttt{lastname}).
\end{aligned}$$

Eine Normalisierung der Heads ist in unserem Beispiel nicht notwendig. Wir starten also direkt mit der Invertierung der (s-t) tgds. Das Vertauschen von Body und Head liefert dann:

$$\begin{aligned}
\texttt{Interim}_1(\texttt{studentID}, \texttt{lastname}, \texttt{grade}) &\rightarrow \texttt{Student}(\texttt{studentID}, \texttt{lastname}) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{grade}) \\
\texttt{Interim}_2(\texttt{studentID}, \texttt{lastname}, 1.3) &\rightarrow \texttt{Interim}_1(\texttt{studentID}, \texttt{lastname}, \texttt{grade}) \wedge \texttt{grade} = 1.3 \\
\texttt{Result}(\texttt{lastname}) &\rightarrow \texttt{Interim}_2(\texttt{studentID}, \texttt{lastname}, 1.3).
\end{aligned}$$

Anschließend werden Heads der s-t tgds ersetzt, sodass eine zusammengesetzte Gesamt-Inverse entsteht:

$$\texttt{Result}(\texttt{lastname}) \rightarrow \texttt{Student}(\texttt{studentID}, \texttt{lastname}) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{grade}) \wedge \texttt{grade} = 1.3.$$

Duplikate und negierte Relationen müssen nicht entfernt werden, sodass diese beiden Schritte übersprungen werden können. Auch Inversen mit gleichem Head existieren nicht. Abschließend wenden wir noch die Gleichheitsprädikate an und ergänzen fehlende Quantoren. Insgesamt ergibt sich so die Gesamt-Inverse:

$$\texttt{Result}(\texttt{lastname}) \rightarrow \exists \texttt{StudentID} : \texttt{Student}(\texttt{StudentID}, \texttt{lastname}) \wedge \texttt{Grade}(\texttt{StudentID}, 1.3).$$

Wir erhalten durch Anwendung von Algorithmus 3 somit die erwartete inverse s-t tgd (vgl. Beispiel 6.2). □

6.3. Auswertungsanfragen und ihre Chase-Inversen *(basierend auf [AH18c; AH22a])*

In ProSA unterscheiden wir drei verschiedene Anfragetypen. Hierzu gehören

- Auswertungsanfragen inklusive Selektion, Projektion, dem natürlichen Verbund, den Mengenoperationen sowie einfachen Aggregatfunktionen;
- Provenance-Anfragen als Invertierung der Auswertungsanfragen;
- Schema-Evolutionen wie der Addition von Attributen, dem Zusammenfassen und Aufspalten von Attributen sowie dem Erstellen neuer Tabellen.

Bis auf wenige Ausnahmen können für alle diese Anfrage exakte, (tp)-relaxte oder ergebnisäquivalente Chase-Inversen (siehe Abschnitt 6.1) angegeben werden.

Chase&Backchase Für die Berechnung der (minimalen) Teil-Datenbank I^* verwenden wir eine um Provenance erweiterte Version des Chase&Backchase, vorgestellt in Abschnitt 6.4. Seien hierfür S ein Quell-Schema, T ein Ziel-Schema und Σ sowie Σ^{-1} zwei Mengen von Abhängigkeiten, welche eine gegebene Anfrage Q und ihre Inverse Q^{-1} formalisieren. Seien $\mathcal{M} = (S, T, \Sigma)$ und $\mathcal{M}^* = (T, S, \Sigma^{-1})$ die zugehörigen Schemaabbildungen. Für zwei Instanzen I und K über S bzw. T berechnet der Chase&Backchase dann die (minimale) Teil-Datenbank mittels

$$I^* = \text{Chase}_{\mathcal{M}^*}(K) = \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)).$$

Chase-Inverse für die Auswertungsanfragen Für die Auswertungsanfragen bedeutet dies, dass die Existenz einer exakten Chase-Inversen (=) nur für einige relationale Operationen wie die Identitätsabbildung, das Kopieren oder univariate, skalare Rechenoperationen nachgewiesen werden kann. Die meisten Operationen haben hingegen eine tp-relaxete ($\preceq_{\text{tp}}$), eine relaxte ($\preceq$) oder eine ergebnisäquivalente Chase-Inverse ($\leftrightarrow$), wie in Tabelle 6.4 zusammengefasst.

Zusätzliche Provenance-Informationen wie die Verwendung von Zeugenliste, (minimale) Zeugenbasen, Provenance-Polynomen [ADT11; BKT01; GT17; Her15] sowie Side Tables (siehe Definition 6.10) können den Chase-Inversentyp beeinflussen. In der Regel kann dieser von relaxt zu tp-relaxt oder von ergebnisäquivalent auf exakt „verbessert“ werden. Grundlage hierfür ist die in Abschnitt 6.1 definierte Reduktion:

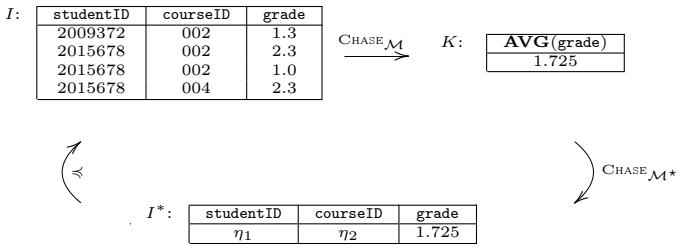
$$\text{ergebnisäquivalent} \preccurlyeq \text{relaxt} \preccurlyeq \text{tp-relaxt} \preccurlyeq \text{exakt}.$$

So kann im Falle der Aggregatfunktionen `AVG` der Inversentyp beispielsweise von einer ergebnisäquivalenten Chase-Inverse ($\leftrightarrow$) ohne weitere Hilfsmittel und in eine exakte Chase-Inverse (=) mit zusätzlichen Provenance-Informationen (vgl. Abschnitt 10.4) verbessert werden. Bei anderen Operationen wie etwa der Selektion ändert sich der Inversentyp durch zusätzliche Provenance-Informationen jedoch nicht (vgl. Tabelle 6.4, Spalte 3 und Spalte 5). Die zugehörige Chase-Inverse ist stets relaxt ($\preccurlyeq$). Operationen wie der natürliche Verbund mit Duplikaten sowie die univariaten arithmetischen Operationen $+$, $-$, $\cdot$ und $\div$ liefern auch ohne zusätzliche Provenance-Informationen eine exakte Chase-Inverse (=). Eine Verschärfung der Chase-Inversen ist hier somit nicht möglich. Doch schauen wir uns die verschiedenen Chase-Inversen am konkreten Beispiel einmal etwas genauer an.

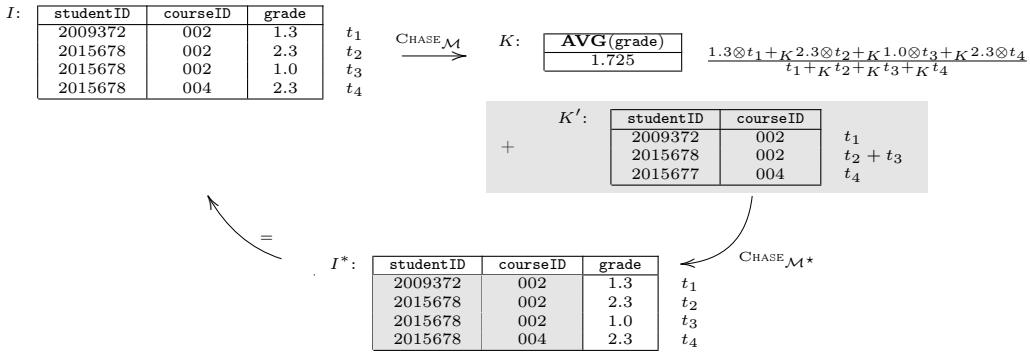
Beispiel 6.4 (basierend auf [AH18c])**.** Gegeben sei die Relation `Grade`, eingeschränkt auf die Attribute `studentID`, `courseID` und `grade`, sowie die Ergebnisrelation `Result`. Die Auswertungsanfrage Q führt die Aggregation des Attributes `grade` durch. Wir formalisieren sie und ihre Inverse als Schemaabbildungen $\mathcal{M} = (\texttt{Grade}, \texttt{Result}, \Sigma)$ und $\mathcal{M}^* = (\texttt{Result}, \texttt{Grade}, \Sigma^{-1})$ mit $\Sigma = \{\texttt{Grade}(\texttt{studentID}, \texttt{courseID}, \texttt{grade}) \rightarrow \texttt{Result}(\texttt{AVG}(\texttt{grade}))$ sowie $\Sigma^{-1} = \{\texttt{Result}(\texttt{AVG}(\texttt{grade})) \rightarrow \exists\ \texttt{StudentID}, \texttt{CourseID} : \texttt{Grade}(\texttt{StudendID}, \texttt{CourseID}, \texttt{grade})\}$.

Ohne Provenance erzeugt die Auswertung von Q und anschließende Invertierung das Tupel $(\eta_1, \eta_2, 1.725)$, welches „lediglich“ den aggregierten Wert von `AVG(grade)` speichert (siehe Abbildung 6.2a). Die zugehörige Chase-Inverse ist ergebnisäquivalent. Mit zusätzlicher Provenance kann sogar eine tp-relaxte Chase-Inverse angeben werden. Hierfür bestimmen wir das zugehörige Provenance-Polynom $t = \frac{1.3 \otimes t_1 +_K 2.3 \otimes t_2 +_K 1.0 \otimes t_3 +_K 2.3 \otimes t_4}{t_1 +_K t_1 +_K t_2 +_K t_2 +_K t_3 +_K t_4}$, welches neben

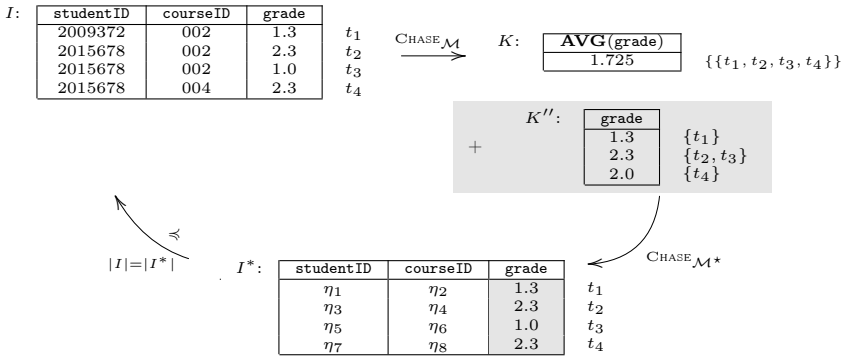
den Provenance-IDs und ihrer Berechnungsvorschrift auch konkrete Attributwerte abspeichert. Mit Hilfe dieser zusätzlichen Informationen erhalten wir die vier Tupel $(\eta_1, \eta_2, 1.3)$, $(\eta_3, \eta_4, 2.3)$, $(\eta_5, \eta_6, 1.0)$ und $(\eta_7, \eta_8, 2.3)$ mit Nullwerten für die Attribute studentID und courseID. Um in unserem Beispiel eine exakte Chase-Inverse zu gewährleisten, muss neben der Auswertungsanfrage $Q =$ `AVG(grade)`, der Ergebnisdatenbank K und dem Provenance-Polynom t auch die *Side Table* K' gespeichert werden (grauer Kasten in Abbildung 6.2b). Die grau eingefärbten Tupel der rekonstruierten Quell-Instanz I^* enthalten keine Nullwerte mehr, sie bestehen stattdessen aus den originalen Attributwerten für `studentID` und `courseID`.



(a) ohne zusätzliche Provenance



(b) unter Verwendung des Provenance-Polynoms $t = \frac{1.3 \otimes t_1 +_K 2.3 \otimes t_2 +_K 1.0 \otimes t_3 +_K 2.3 \otimes t_4}{t_1 +_K t_2 +_K t_3 +_K t_4}$ und der zusätzlichen Side Table K'



(c) unter Verwendung der Zeugenbasis $w = \{\{t_1, t_2, t_3, t_4\}\}$ sowie der zusätzlichen Side Table K''

Abbildung 6.2. Invertierung der Aggregatfunktion `AVG` mit und ohne zusätzlicher Provenance.

Die zusätzliche Side Table K' ist jedoch in vielen Fällen, z.B. aus datenschutzrechtlichen Gründen, nicht notwendig, da eine vollständige Rekonstruktion der Quell-Instanz nicht gewünscht ist (siehe Abschnitt 8.1). Auch die Verwendung von Provenance-Polynomen ist oftmals sehr aufwendig, liefert aber keinen nennenswerten Mehrwert, wie in Abschnitt 8.2 beschrieben. Wir beschränken uns daher im Folgenden auf die Verwendung von Zeugenbasen im Sinne der ***why***-Provenance. Sei hierfür $W_{Q,I} = \{\{t_1, t_2, t_3, t_4\}\}$ die Zeugenbasis des Ergebnistupels in K. Mit Hilfe der zusätzlichen Side Table K'', welche lediglich die Attributwerte von `grade` speichert, können so die vier Tupel $(\eta_1, \eta_2, 1.3)$, $(\eta_3, \eta_4, 2.3)$, $(\eta_5, \eta_6, 1.0)$ und $(\eta_7, \eta_8, 2.3)$ rekonstruiert werden (siehe Abbildung 6.2c). In den meisten Fällen ist diese tp-relaxte Chase-Inverse vollkommen ausreichend. □

Die inverse Schemaabbildung kann als Quasi-Inverse stets explizit angegeben werden (siehe Tabelle 6.4, Spalte 4 und Spalte 6). Sie entspricht in den meisten Fällen der Identitätsabbildung. Zusätzliche Provenance-Informationen wie Zeugenbasen oder Side Tables erlauben jedoch häufig die Rekonstruktion verlorener Tupel, wie sie durch Dangling Tuples entstehen können. Auch die Einschränkung des Anfrageergebnisses auf ihre Quell-Relationen wie bei der Vereinigung ist durch zusätzliche Informationen möglich. Für die Rekonstruktion verlorener Attributwerte, welche beispielsweise durch Duplikate entstehen, werden zusätzlich Side Tables benötigt. Welche zusätzlichen Informationen jeweils notwendig sind, ist in Tabelle 6.4, Spalte 7 zusammengefasst.

Schauen wir uns als nächstes an, wie die häufigsten Auswertungsoperationen anhand ihrer Chase-Inversentypen klassifiziert werden können. Hierzu unterscheiden wir insgesamt vier Klassen: Provenance-Invariant (Klasse I), Dangling Tuples (Klasse II), Duplikate (Klasse III) und Wiederhergestellt durch Provenance (Klasse IV). Eine Zuordnung der Auswertungsoperatoren zu ihren Klassen sowie ihre Inversen ist in den Tabellen 6.3 und 6.4 zusammengefasst. Die zugehörigen Beweise können im Anhang C nachgelesen werden.

Auswertungsanfrage	**Beschreibung**	**Inverse**	**Klasse**
$r(\mathcal{R})$	Identitätsabbildung	Identitätsabbildung	I
$\beta_{A_j \leftarrow A_i}(r(\mathcal{R}))$	Umbenennung eines Attributes A_i	inverse Umbenennung	I
$\pi_{A_i}(r(\mathcal{R}))$	Projektion auf ein Attribut A_i	Identitätsabbildung mit Nullwert-Erweiterung	III
$r_1(\mathcal{R}_1) \bowtie r_2(\mathcal{R}_2)$	Natürlicher Verbund zweier Relationen r_1 und r_2	Identitätsabbildung	II
$\sigma_{A_i \theta c}(r(\mathcal{R}))$ mit $\theta \in \{<, \leq, =, \neq, >, \geq\}$	Konstanten-Selektion	Identitätsabbildung	I
$\sigma_{A_i \theta A_j}(r(\mathcal{R}))$ mit $\theta \in \{<, \leq, =, \neq, >, \geq\}$	Attribut-Selektion	Identitätsabbildung	I
$r_1(\mathcal{R}_1) \cup r_2(\mathcal{R}_2)$	Vereinigung zweier Relationen r_1 und r_2	Identitätsabbildung	IV
$r_1(\mathcal{R}_1) \cap r_2(\mathcal{R}_2)$	Schnittmenge zweier Relationen r_1 und r_2	Identitätsabbildung	I
$r_1(\mathcal{R}_1) - r_2(\mathcal{R}_2)$	Differenz zweier Relationen r_1 und r_2	Identitätsabbildung	IV
$\texttt{MAX}_{A_i}(r(\mathcal{R}))/\texttt{MIN}_{A_i}(r(\mathcal{R}))$	Aggregatfunktionen `MAX` und `MIN`	Identitätsabbildung	$*$
$\texttt{COUNT}(*)$	Aggregatfunktion `COUNT`	Identitätsabbildung	$*$
$\texttt{SUM}_{A_i}(r(\mathcal{R}))$	Aggregatfunktion `SUM`	Identitätsabbildung	$*$
$\texttt{AVG}_{A_i}(r(\mathcal{R}))$	Aggregatfunktion `AVG`	Identitätsabbildung	$*$
$\gamma_{G_i;F_j(A_j)}(r(\mathcal{R}))$	Gruppierung	Identitätsabbildung	$*$
$r(\mathcal{R})\theta\alpha$ mit $\theta \in \{+, -, \cdot, /\}$	skalare Operationen	inverse skalare Anwendung	I

Tabelle 6.3. Auswertungsanfragen mit ihren zugehörigen Chase-Inversen und Provenance-Klassen I bis IV; Erläuterungen zu $*$ folgen in Abschnitt 10.4.

Komposition verschiedener Auswertungsoperationen Da die Operatoren innerhalb einer Sequenz unabhängig voneinander arbeiten, können wir auch ihre Inversen unabhängig voneinander bestimmen. Die Inverse einer Komposition von Schemaabbildungen $\mathcal{M}^* = (\mathcal{M}_1 \circ ... \circ \mathcal{M}_n)^{-1} = \mathcal{M}_n^* \circ ... \circ \mathcal{M}_1^*$ ergibt sich somit aus dem Produkt der einzelnen Teil-Inversen $\mathcal{M}_1^*, ..., \mathcal{M}_n^*$. Der Inversentyp von $\mathcal{M}^*$ entspricht dabei höchstens dem Typ des schwächsten Teil- Inversen $\mathcal{M}_i^*$. Weiterhin ist zu beachten: Existiert für eine der Teil-Operationen $\mathcal{M}_i$ keine Inverse, so auch nicht für ihre Zusammensetzung $\mathcal{M} = \mathcal{M}_1 \circ ... \circ \mathcal{M}_n$.

Korollar 6.1. *Seien $\mathcal{M}_1$ und $\mathcal{M}_2$ zwei Schemaabbildungen. Der Inversentyp ihrer Komposition $\mathcal{M}_1 \circ \mathcal{M}_2$ entspricht höchstens dem Typ der schwächsten Teil-Inversen $\mathcal{M}_1$ oder $\mathcal{M}_2$.*

Als Beispiel seien hier die Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM` und `AVG` genannt. Wie in Abschnitt 10.4 beschrieben, können diese als Menge von mehreren (s-t) tgds formalisiert werden, welche in einer bestimmten Reihenfolge hintereinander ausgeführt werden.

Klassifikation der Auswertungsanfragen

Seien S und T zwei Datenbankschemata und Q eine gegebene Anfrage, dargestellt als Menge von (s-t) tgds. Seien ferner $\mathcal{M} = (S, T, \Sigma)$ und $\mathcal{M}^* = (T, S, \Sigma^{-1})$ zwei Schemaabbildungen, welche Q und Q^{-1} formalisieren. Seien weiter R, V und T drei Relationen, die später konkretisiert werden. Sei r_i die Tupel-ID des i-ten Quell-Tupels und w_j die Zeugenbasis des j-ten Ziel-Tupels. Seien I, J und I^* drei Instanzen über S bzw. T, wobei J und I^* wie folgt berechnet werden: $J = \text{Chase}_{\mathcal{M}^*}(I)$ und $I^* = \text{Chase}_{\mathcal{M}^*}(J) = \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I))$. Dann definieren wir vier Klassen von Operatoren: Provenance-invariante Operationen (Klasse I), Operationen mit Dangling

Tuples (Klasse II) oder Duplikaten (Klasse III) sowie Operationen, deren Chase-Inversentyp ohne weitere Einschränkungen durch zusätzliche Provenance beeinflusst wird (Klasse IV). Die Ergebnisse der Klassifikation der Auswertungsoperatoren ist in Tabelle 6.3 zusammengefasst.

Klasse I: Provenance-Invariant Im Falle der Identitätsabbildung, der Umbenennung, der Selektion sowie der Anwendung skalarer Operationen ändert zusätzliche Provenance nichts am zugehörigen Inversentyp, da bereits ohne zusätzliche Informationen eine bestmögliche Chase-Inverse definiert werden kann. Lemma 6.1 fasst dies wie folgt zusammen:

Lemma 6.1. *Sei $\mathcal{M} = (S, T, \Sigma)$ eine Schemaabbildung der Klasse I. Dann ist die entsprechende inverse Funktion $\mathcal{M}^* = (T, S, \Sigma^{-1})$ immer eine exakte oder relaxte* Chase*-Inverse unabhängig von zusätzlichen Provenance-Informationen.*

Beweis. Sei $\mathcal{M} = (S, T, \Sigma)$ die Schemaabbildung, welche die Selektion beschreibt, und $\mathcal{M}^* = (T, S, \Sigma^{-1})$ die zugehörige inverse Funktion, formalisiert über

$$\Sigma = \{\exists c \in \mathbb{D}(a_{i_2}) : R(a_{i_1}, a_{i_2}, a_{i_3}) \wedge a_{i_2}\theta c \rightarrow T(a_{i_1}, a_{i_2}, a_{i_3})\} \quad \text{und} \quad \Sigma^{-1} = \{T(a_{i_1}, a_{i_2}, a_{i_3}) \rightarrow R(a_{i_1}, a_{i_2}, a_{i_3})\}.$$

O.B.d.A. sei R auf drei Attribute beschränkt. Ferner sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz mit verschiedenen drei-elementigen Tupeln. Dann liefert der Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge a_{i_2}\theta c\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge a_{i_2}\theta c\} \\ &\preccurlyeq I. \end{aligned}$$

Alle Tupel aus R, welche $a_{i_2}\theta c$ erfüllen, werden heraus selektiert und in T abgespeichert. Anschließend werden diese mit der Identitätsabbildung zurück nach R abgebildet, sodass gilt: $I^* \preccurlyeq I$. Wir können somit eine relaxte Chase-Inverse garantieren. Die Angabe einer exakten Chase-Inversen ist hingegen nicht möglich. Auch zusätzliche Provenance-Informationen wie Zeugenbasen ändern hieran nichts. Side Tables, welche die heraus selektierten Tupel speichern, sind ebenfalls nicht notwendig, da diese „zusätzlichen“ Tupel bei der erneuten Ausführung der Selektion wiederum verloren gehen würden. □

Die Aussagen über die anderen Klassenvertreter, d.h. für die Identitätsabbildung, die Umbenennung, die Schnittmengenbildung sowie die skalaren Operationen $+$, $-$, $\cdot$ und $\div$, können analog bewiesen werden. Hierfür sei auf den Anhang C verwiesen. Wir schließend außerdem:

Korollar 6.2. *Sei $\mathcal{M}$ eine Schemaabbildung der Klasse I, d.h. $\mathcal{M}$ hat eine exakte* Chase*-Inverse. Dann liefert zusätzliche Provenance keine weiteren Informationen.* □

Klasse II: Dangling Tuples Je nach Quell-Instanz können Dangling Tuples auftreten. So kann im Falle eines natürlichen Verbundes nicht ohne Weiteres eine exakte Chase-Inverse garantiert werden. Wie in Lemma 6.2 zusammengefasst, können zusätzliche Side Tables hierbei jedoch Abhilfe schaffen.

Lemma 6.2. *Sei $\mathcal{M} = (S, T, \Sigma)$ eine Schemaabbildung der Klasse II. Dann ist die entsprechende* Chase*-Inverse $\mathcal{M}^* = (T, S, \Sigma^{-1})$ exakt, sofern die Anwendung des* Chase *auf $\mathcal{M}$ keine Dangling Tuples liefert. Liegen Dangling Tuples vor, ist der* Chase*-Inversentyp relaxt. Zusätzliche Side Tables sowie Provenance-Annotationen ermöglichen jedoch eine exakte* Chase*-Inversen.*

Da die Auswertung eines natürlichen Verbundes mit der Operation `JOIN Table` zu vergleichen ist, verweisen wir an dieser Stelle auf die Erläuterungen in Abschnitt 7.3 sowie den Beweis von `JOIN Table` im Anhang C.

Klasse III: Duplikate Das Vorhandensein von Duplikaten wie etwa bei der Projektion auf ein konkretes Attribut erlaubt nur die Angabe einer relaxten Chase-Inversen. Der Inversentyp kann durch zusätzliche Provenance-Informationen wie Zeugenbasen und Side Tables jedoch zu einer exakten Chase-Inversen verbessert werden.

Lemma 6.3. *Sei $\mathcal{M} = (S, T, \Sigma)$ eine Schemaabbildung der Klasse III. Dann ist die entsprechende* Chase*-Inverse $\mathcal{M}^* = (T, S, \Sigma^{-1})$ relaxt. Existieren keine Duplikate, so ist der Inversentyp tp-relaxt. Zusätzliche Provenance-Informationen wie Zeugenbasen und Side Tables ermöglichen im Falle von Duplikaten zudem die Angabe einer tp-relaxten oder exakten* Chase*-Inverse.*

Beweis. Sei $\mathcal{M} = (S, T, \Sigma)$ die Schemaabbildung, welche die Projektion beschreibt und $\mathcal{M}^* = (T, S, \Sigma^{-1})$ die zugehörige inverse Funktion, formalisiert über

$$\Sigma = \{R(a_1, a_i, a_2) \to T(a_1, a_2)\} \quad \text{und} \quad \Sigma^{-1} = \{T(a_1, a_2) \to \exists A_i : R(a_1, A_i, a_2)\}.$$

O.b.d.A. sei R auf drei und T auf zwei Attribute beschränkt. Sei ferner $I = \{R(x_i, x_i, x_i)_{r_i} \mid i_1, i_2, i_3 \in \mathbb{N} \wedge r_i \text{ Tupelidentifikator}\}$ eine beliebige Quell-Instanz, wobei r_i Tupelidentifikator des i-ten Tupels ist. Seien weiter $R(x_{j_1}, x_{j_2}, x_{j_3})$ und $R(x_{k_1}, x_{k_2}, x_{k_3})$ zwei Tupel, welche nach der Projektion ein Duplikat liefern. Sei $w_j = \{\{r_j\}, \{r_k\}\}$ Zeugenbasis von r_j und $w_k = \{\{r_j\}, \{r_k\}\}$ Zeugenbasis von r_k (orange hervorgehoben). Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2})_{w_i} \mid i \in \mathbb{N} \wedge w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\})\} \\ &= \{R(x_{i_1}, \eta_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge \eta_{j_2} = \eta_{k_2}\} \\ &\preccurlyeq_{\text{tp}} I. \end{aligned}$$

Aufgrund der Existenzquantoren erzeugt die inverse s-t tgd immer einen neuen Nullwert η_{i_2}. Wegen $T(x_{j_1}, x_{j_2}) = T(x_{k_1}, x_{k_2})$ enthält die rekonstruierte Instanz I^* jedoch weniger Tupel als die ursprüngliche Instanz I. Die Inverse $\mathcal{M}^*$ ist tp-relaxt. Das Hinzufügen von Provenance garantiert zudem eine Chase-Inverse. □

Abbildung 6.3 zeigt die obige Situation an einem konkreten Beispiel. Sei hierfür R ein Ausschnitt der Relation `Lecturer` aus Anhang A. Das Duplikat (Professor) in T kann jedoch mit Hilfe zusätzlicher Provenance-Informationen wie etwa der Zeugenbasis (hervorgehoben in orange), rekonstruiert werden.

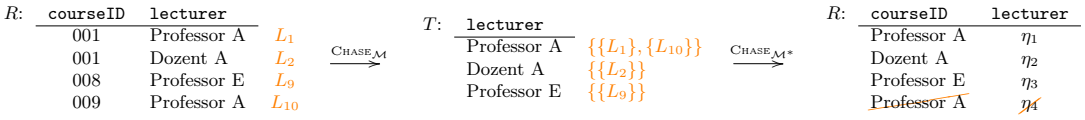


Abbildung 6.3. Chase-Inversentyp der Projektion erweitert um Provenance-Informationen (hervorgehoben in orange).

Klasse IV: Wiederhergestellt durch Provenance Klasse IV vereint alle Operationen, die einen Informationsverlust aufweisen, welcher jedoch nicht durch Dangling Tuples oder Duplikate zu begründen ist. Dies betrifft insbesondere die Vereinigung sowie die Differenz zweier Relationen.

Lemma 6.4. *Die Vereinigung zweier Relationen hat eine ergebnisäquivalente* Chase*-Inverse. Zusätzliche Provenance-Informationen ermöglichen alternativ die Angabe einer exakten* Chase*-Inversen.*

Sei hierfür Q : `Participant(courseID, studentID)` $\cup$ `ParticipantCopy(courseID, studentID)` eine gegebene Anfrage und I = {`Participant(001,1)`, `Participant(001,2)`, `ParticipantCopy(001,1)`} eine beispielhafte Instanz. Dann liefert die Anwendung des Chase auf

$$\texttt{Participant}(c, s) \to \texttt{Result}(c, s); \texttt{ParticipantCopy}(c, s) \to \texttt{Result}(c, s);$$
$$\texttt{Result}(c, s) \to \texttt{Participant}(c, s); \texttt{Result}(c, s) \to \texttt{ParticipantCopy}(c, s)$$

die rekonstruierte Instanz I^* mit

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(\{\underbrace{\texttt{Participant}(001,1)}_{p_1}, \underbrace{\texttt{Participant}(001,2)}_{p_2}, \underbrace{\texttt{ParticipantCopy}(001,1)}_{p_3}\})) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{\underbrace{\texttt{Result}(001,1)}_{\{\{p_1\},\{p_3\}\}}, \underbrace{\texttt{Result}(001,2)}_{\{\{p_2\}\}}\}) \\ &= \{\texttt{Participant}(001,1), \texttt{Participant}(001,2), \texttt{ParticipantCopy}(001,1), \texttt{ParticipantCopy}(001,2)\} \\ &\succcurlyeq I, \end{aligned}$$

wobei I^* mehr Tupel als I enthalten kann. Die fehlerhaft rekonstruierten Tupel in I^* können jedoch mit Hilfe zusätzlicher Provenance-Informationen (orange hervorgehoben) identifiziert und gelöscht werden, sodass gilt: $I^* = I$. Für die Vereinigung zweier Relationen existiert daher eine relaxte Chase-Inverse ohne sowie eine exakte Chase-Inverse mit Integration zusätzlicher Provenance-Informationen.

Lemma 6.5. *Die Differenz zweier Relationen hat eine tp-relaxte* Chase*-Inverse.*

Sei nun Q : `Participant(courseID,studentID)` − `Participant(courseID, studentID)` eine gegebene Anfrage und I die gleiche Instanz wie oben, d.h. $I = \{\texttt{Participant}(001,1), \texttt{Participant}(001,2), \texttt{ParticipantCopy}(001,1)\}$. Dann liefert die Anwendung des Chase auf

$$\begin{aligned} \texttt{Participant}(c,s) \wedge \neg\texttt{ParticipantCopy}(c,s) &\rightarrow \texttt{Result}(c,s) \\ \texttt{Participant}(c,s) \wedge \texttt{ParticipantCopy}(c,s) &\rightarrow H(c,s); \\ \texttt{Result}(c,s) &\rightarrow \texttt{Participant}(c,s) \\ H(c,s) &\rightarrow \texttt{Participant}(c,s) \wedge \texttt{ParticipantCopy}(c,s) \end{aligned}$$

die rekonstruierte Instanz I^* mit

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(\{\underbrace{\texttt{Participant}(001,1)}_{p_1}, \underbrace{\texttt{Participant}(001,2)}_{p_2}, \underbrace{\texttt{ParticipantCopy}(001,1)}_{p_3}\})) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{\underbrace{\texttt{Result}(001,2)}_{\{\{p_2\}\}}, \underbrace{H(001,1)}_{\{\{p_1\},\{p_3\}\}}\}) \\ &= \{\texttt{Participant}(001,1), \texttt{Participant}(001,2), \texttt{ParticipantCopy}(001,1)\} \\ &\preccurlyeq / = I. \end{aligned}$$

Ohne Provenance können wir somit eine relaxte Chase-Inverse angeben. Mit zusätzlichen Provenance-Informationen hingegen können die gelöschten Tupel jedoch in der Side Table $\texttt{H}(c,s)$ abgespeichert und über die erweiterten s-t tgds $\texttt{Participant}(c,s) \wedge \texttt{ParticipantCopy}(c,s) \rightarrow \texttt{H}(c,s)$ und $\texttt{H}(c,s) \rightarrow \texttt{ParticipantCopy}(c,s) \wedge \texttt{Participant}(c,s)$ (orange hervorgehoben) rekonstruiert werden. So kann im besten Falle eine exakte Chase-Inverse angegeben werden.

Für weitere Erläuterungen sowie die zugehörigen Beweise von Lemma 6.4 und Lemma 6.5 sei an dieser Stelle ebenfalls auf Abschnitt 7.3 sowie Anhang C verwiesen.

Auswertungsanfrage	ohne Provenance			mit Provenance		
	Σ	Type	Σ^{-1}	Type	Σ^{-1}	Provenance
$r(\mathcal{R})$	$R(a,b,c) \to T(a,b,c)$	$=$	$T(a,b,c) \to R(a,b,c)$	$=$	$T(a,b,c) \to R(a,b,c)$	
$\beta_{d \leftarrow b}(r(\mathcal{R}))$	$R(a,b,c) \to \exists D : T(a,D,c)$	$=$	$T(a,d,c) \to \exists D : R(a,D,c)$	$=$	$T(a,d,c) \to \exists B : R(a,B,c)$	
$\pi_b(r(\mathcal{R}))$	$R(a,b,c) \to T(a,c)$	$\preccurlyeq_{\text{tp}}$	$R(a,c) \to \exists B : R(a,B,c)$	$\preccurlyeq_{\text{tp}}$	$R(a,c) \to \exists B : R(a,B,c)$	
		$\preccurlyeq$	$R(a,c) \to \exists B : R(a,B,c)$	$\preccurlyeq_{\text{tp}}$	$R(a,c) \to \exists B : R(a,B,c)$ + Rekonstruktion von verlorenen Tupeln (Duplikate)	***why, how***
$r_1(\mathcal{R}_1) \bowtie r_2(\mathcal{R}_2)$	$R(a,b) \wedge V(b,c) \to T(a,b,c) \wedge H(a) \wedge H(c)$	$=$	$T(a,b,c) \to R(a,b) \wedge V(b,c)$	$=$	$T(a,b,c) \to R(a,b) \wedge V(b,c)$	
		$\preccurlyeq$	$T(a,b,c) \to R(a,b) \wedge V(b,c)$	$\preccurlyeq$	$T(a,b,c) \to R(a,b) \wedge V(b,c)$	
				$=$	$T(a,b,c) \to R(a,b) \wedge V(b,c)$ $H(a) \to \exists B : R(a,B)$ $H(c) \to \exists B : R(B,c)$ (Rekonstruktion von verlorenen Tupeln)	***why, how*** + Side Tables
$\sigma_{b \theta a}(r(\mathcal{R}))$ mit $\theta \in \{<, \leq, =, \neq, >, \geq\}$	$\exists a \in \mathbb{R} : (R(a,b,c) \wedge b \theta a \to T(a,b,c))$	$\preccurlyeq$	$T(a,b,c) \to R(a,b,c)$	$\preccurlyeq$	$T(a,b,c) \to R(a,b,c)$	
$\sigma_{b \theta c}(r(\mathcal{R}))$ mit $\theta \in \{<, \leq, =, \neq, >, \geq\}$	$R(a,b,c,d) \wedge b \theta c \to T(a,b,c,d)$	$\preccurlyeq$	$T(a,b,c,d) \to R(a,b,c,d)$	$\preccurlyeq$	$T(a,b,c,d) \to R(a,b,c,d)$	
$r_1(\mathcal{R}_1) \cup r_2(\mathcal{R}_2)$	$R(a,b) \to V(a,b)$; $V(a,b) \to T(a,b)$	$\updownarrow$	$T(a,b) \to V(a,b)$; $T(a,b) \wedge R(a,b)$	$=$	$T(a,b) \to R(a,b)$; $T(a,b) \wedge V(a,b)$ + Einschränkung von Tupeln aus R bzw. V	***why, how***
$r_1(\mathcal{R}_1) \cap r_2(\mathcal{R}_2)$	$R(a,b) \wedge V(a,b) \to T(a,b)$	$\preccurlyeq$	$T(a,b) \to R(a,b) \wedge V(a,b)$	$\preccurlyeq$	$R(a,b) \wedge V(a,b) \to T(a,b)$	
$r_1(\mathcal{R}_1) - r_2(\mathcal{R}_2)$	$R(a,b) \wedge \neg V(a,b) \to T(a,b) \wedge H(a,b)$	$\preccurlyeq$	$T(a,b) \to R(a,b)$	$=$	$T(a,b) \to R(a,b)$ $H(a,b) \to R(a,b) \wedge V(a,b)$ (Rekonstruktion von verlorenen Tupeln)	***why, how*** + Side Tables
$\texttt{MAX}_{A_i}(r(\mathcal{R}))/\texttt{MIN}_{A_i}(r(\mathcal{R}))$	siehe Abschnitt 10.4	$\preccurlyeq$	siehe Abschnitt 10.4	$\preccurlyeq_{\text{tp}}$	siehe Abschnitt 10.4	Side Tables
$\texttt{COUNT}(*)$ $\texttt{SUM}_{A_i}(r(\mathcal{R}))$ $\texttt{AVG}_{A_i}(r(\mathcal{R}))$	siehe Abschnitt 10.4	$\preccurlyeq_{\text{tp}}$ $\leftrightarrow$ $\leftrightarrow$	siehe Abschnitt 10.4	$\preccurlyeq_{\text{tp}}$ $=$ $=$	siehe Abschnitt 10.4	***why, how*** + Side Tables
$\gamma_{G_i;F_j(A_j)}(r(\mathcal{R}))$	siehe Abschnitt 10.4	$\preccurlyeq$ $\preccurlyeq_{\text{tp}}$ $\leftrightarrow$	siehe Abschnitt 10.4	$\preccurlyeq$ $\preccurlyeq_{\text{tp}}$ $=$	siehe Abschnitt 10.4	***why, how***
$r(\mathcal{R} \mid_{b \theta a})$ mit $\theta \in \{+, -, \cdot, /\}$	$\exists a \in \mathbb{R} : (R(a,b,c) \wedge d = b \theta a \to T(a,d,c))$	$=$	$\exists a \in \mathbb{R} : (T(a,b,c) \wedge b = d\theta^{-1}a)$	$=$	$\exists a \in \mathbb{R} : (T(a,b,c) \wedge b = d\theta^{-1}a)$	

Tabelle 6.4. Auswertungsanfragen mit exakter ($=$), tp-relaxter ($\preccurlyeq_{\text{tp}}$), relaxter ($\preccurlyeq$) oder ergebnisäquivalenter ($\leftrightarrow$) Chase-Inversen mit und ohne zusätzliche Provenance-Informationen (orange hervorgehoben); Dangling Tuples sind rot und Duplikate blau hervorgehoben; fehlerhaft rekonstruierte Tupel (grün hervorgehoben).

6.4. Chase&Backchase zur Bestimmung der (minimalen) Teil-Datenbank *(basierend auf [Aug17; AH18c])*

In diesem Abschnitt stellten wir unsere Version des Chase&Backchase zur Berechnung der (minimalen) Teil-Datenbank I^* vor und erweitern diesen anschließend um die Einbindung von zusätzlichen Provenance-Informationen. Seien hierfür $\mathcal{M} = (S, T, \Sigma)$ und $\mathcal{M}^* = (T, S, \Sigma^{-1})$ zwei Schemaabbildungen über einem Quell-Schema S sowie einem Ziel-Schema T. Wir unterscheiden drei Instanzen, welche durch einfache oder doppelte Anwendung des Chase entstehen:

- Quell-Instanz: I
- Ergebnisinstanz: $K = \text{Chase}_{\mathcal{M}}(I)$
- Rekonstruierte Instanz: $I^* = \text{Chase}_{\mathcal{M}^*}(K) = \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I))$

Der erste Schritt bestimmt K, der Zweite I^*. Diesen zweite Chase-Schritt, der für die Bestimmung bzw. den Nachweis dieser Chase-Inversen notwendig sind, können wir als eine Form des Backchase-Phase auffassen. Während der Chase-Schritt für Anfragen, Instanzen, Abhängigkeiten und Sichten auf ein allgemeines Chase-Objekt $\bigcirc$ sowie einem allgemeinen Chase-Parameter $*$ verallgemeinert werden kann (siehe Abschnitt 9.1), muss der Backchase-Schritt für jeden Anwendungsfall einzeln bestimmt werden. Wir unterscheiden daher verschiedene Backchase-Varianten für Anfragen (AQuV) [DH13], Ergebnisinstanzen [Aug17; AH18c] sowie die Anfrageoptimierung [DPT06]. Einen Überblick hierzu bietet auch [Ros20].

Wie in [Aug17] beschrieben, dient die zweite Chase-Anwendung in unserem Fall der Rückabbildung der Ergebnisinstanz K auf die ursprüngliche Quell-Instanz I. In der Regel gelingt dies aber nur teilweise, sodass zusätzliche Provenance-Informationen wie Zeugenbasen und Side Tables notwendig sind (siehe Abschnitt 6.3). Die rekonstruierte Instanz I^* kann anschließend mit der Quell-Instanz I verglichen werden. Die Berechnung von I^* kann als Backchase interpretiert werden, in der $\text{Chase}_{\mathcal{M}^*}(K)$ als eine Art inverse Schemaabbildung des Chase-Schrittes $\text{Chase}_{\mathcal{M}}(I)$ angesehen wird. Mit anderen Worten: Die Schemaabbildung $\mathcal{M}^*$ ist Chase-Inverse von $\mathcal{M}$. Das zugehörige Chase&Backchase-Verfahren definieren wir wie folgt:

Definition 6.9 (Chase&Backchase)**.** Seien zwei Schemaabbildungen $\mathcal{M} = (S, T, \Sigma)$ und $\mathcal{M}^* = (T, S, \Sigma^{-1})$ gegeben. Sei weiter I eine Quell-Instanz auf dem Quell-Schema S. Dann gilt:

- Chase: Berechne den Chase von I bzgl. $\mathcal{M}$ als Sequenz von tgd- und egd-Regeln der Menge Σ. Das Ergebnis dieser Chase-Anwendung werde als K bezeichnet.
- Backchase: Berechne den Chase von K bzgl. $\mathcal{M}^*$ als Sequenz von tgd- und egd-Regeln der Menge Σ^{-1}. Das Ergebnis dieser Chase-Anwendung werde als I^* bezeichnet. ■

In ProSA erweitern wir dieses „klassische“ Chase&Backchase-Verfahren um zusätzliche Provenance-Informationen durch Hinzufügen von Zeugenbasen und Side Tables. Wie in Abschnitt 8.2 beschrieben, können wir uns im Folgenden auf die ***why***-Provenance beschränken. Eine Version des *Provenance-aware* Chase&Backchase auf Basis der ***how***-Provenance kann jedoch in [Aug17] nachgelesen werden.

Definition 6.10 (Side Table)**.** Eine *Side Table* ist eine Relation R, welche konkrete Attributwerte der Quell-Instanz I speichert. Die zugehörigen Tupel können aus einem oder mehreren Attributen bestehen. Jedes Tupel ist zudem mit einer Liste der zugehörigen IDs aus I versehen. ■

Bei der erweiterten Variante des Chase&Backchase werden in der Chase-Phase zusätzlich für jedes Tupel der Ergebnisinstanz K die zugehörige Zeugenbasis w_t[2] sowie, falls notwendig, konkrete Attributwerte in separaten Side Tables (siehe Definition 6.10) gespeichert. Mit Hilfe der Zeugenbasen werden dann in der Backchase-Phase die Tupel der rekonstruierten Instanz I^* wie folgt erzeugt:

- Besteht die Zeugenbasis w aus nur einer Tupel-ID, d.h. $w = \{\{t_i\}\}$, ist das Tupel mit der ID t_i Teil der rekonstruierten Instanz I^*.

[2] Da keine konkrete Anfrage Q gegeben ist, schreiben wir für die Zeugenbasis kurz w.

- Enthält die Zeugenbasis w zwei Zeugenmengen, d.h. $w = \{\{t_i\}, \{t_j\}\}$, entsteht für jede Zeugenmenge $\{t_i\}$ und $\{t_j\}$ jeweils ein eigenes Tupel in I^* aufgefüllt um (markierte) Nullwerte.
- Enthält die Zeugenbasis w eine Zeugenmenge bestehend aus zwei Tupel-IDs, d.h. $w = \{\{t_i, t_j\}\}$, entsteht für jede Tupel-ID der Zeugenmenge $\{t_i, t_j\}$ jeweils ein Tupel in den beiden Quellrelationen.
- Existiert zu einem Tupel der Zeugenbasis w zudem ein Tupel in der Side Table K', werden die entsprechenden Attributwerte a_m in der rekonstruierten Instanz I^* ergänzt.

Auch diesen Abschnitt schließen wir mit einem Beispiel ab. Zuvor definieren wir noch den *Provenance-aware* Chase&Backchase für die Berechnung der (minimalen) Teil-Datenbank basierend auf der in [Aug17] vorgestellten Definition jedoch noch einmal formal:

Definition 6.11 (Provenance-aware Chase&Backchase)**.** Gegeben seien zwei Datenbankschemata S und T sowie zwei darauf definierte Schemaabbildungen $\mathcal{M} = (S, T, \Sigma)$ und $\mathcal{M}^* = (T, S, \Sigma^{-1})$. Sei weiter I eine Instanz über S. Dann gilt:

- Chase-Phase: Berechne K durch Anwendung des Chase auf I bzgl. $\mathcal{M}$:
  1. Berechne das Ergebnis der Chase-Anwendung als Sequenz von tgd- und egd-Regeln der Menge Σ. Das Ergebnis dieser Chase-Anwendung werde als K bezeichnet.
  2. Bestimme die zugehörige Provenance in Form einer
     - (minimalen) Zeugenbasis w_i für ein Tupel $k_i \in K$;
     - Side Table K' mit konkreten Attributwerten einzelner Quell-Tupel.
- Provenance-gestützte Backchase-Phase: Berechne die rekonstruierte Instanz I^* durch die erneute Anwendung des Chase auf K bzgl. $\mathcal{M}^*$ unter Berücksichtigung von Σ sowie der zusätzlichen Provenance-Informationen.

  Sei I eine gegebene Instanz über zwei Relationen R_1 und R_2. Seien $t_{1,j}$ und $t_{1,k}$ zwei Tupel in R_1 und $t_{2,l}$ ein Tupel in R_2 ($j, k, l \in \mathbb{N}$). Seien weiter $t'_{1,j}$, $t'_{1,k}$ und $t'_{2,l}$ die zugehörigen Tupel der rekonstruierten (minimale) Teil-Datenbank I^*, sofern vorhanden. Sei weiter w_i Zeugenbasen des Ergebnistupels $k_i \in K$ ($i \in \mathbb{N}$). Sei a_m mit $m \in \mathbb{N}$ Attributwert zum Ergebnistupel k_i, welcher in einer zusätzlichen Side Table gespeichert werden. Dann gilt für ein Ergebnistupel $k_i \in K$ mit zugehöriger Zeugenbasis w_i:
  - w_i besteht aus nur einem Tupel, d.h. $w_i = \{\{t_j\}\} \Rightarrow \text{Chase}_{\mathcal{M}^*}(k_i) = \{t'_j\}$
  - w_i enthält zwei Zeugenmengen, d.h. $w_i = \{\{t_{1,j}\}, \{t_{1,k}\}\} \Rightarrow \text{Chase}_{\mathcal{M}^*}(k_i) = \{t'_{1,j}, t'_{1,k}\}$
  - w_i enthält eine Zeugenmenge bestehend aus zwei Tupel-IDs, d.h. $w_i = \{\{t_{1,j}, t_{2,k}\}\}$ $\Rightarrow \text{Chase}_{\mathcal{M}^*}(k_i) = \{t'_{1,j}, t'_{2,k}\}$
  - Zu einem Tupel der Zeugenbasis w_i existiert ein Tupel t in der Side Table K', d.h. $t(a_m) \in K''$ $\Rightarrow$ Ersetze den Wert des Attributes A durch a_m in allen durch $\text{Chase}_{\mathcal{M}^*}(k_i)$ erzeugten Tupeln. ■

Beispiel 6.5. Gegeben sei die bekannte Relation `Student` sowie die Anfrage Q = `SELECT lastName, AVG(grade) FROM Student NATURAL JOIN Grade WHERE firstName = 'Jack' GROUP BY lastName`. Seien $\mathcal{M} = (S, T, \Sigma)$ und $\mathcal{M}^* = (T, S, \Sigma^{-1})$ die zugehörige Schemaabbildung sowie ihre Inverse, wobei S Quell-Schema, T Ziel-Schema und Σ Formalisierung von Q ist. Dann ergibt sich in der Chase-Phase:

$K =$

lastName	AVG(grade)	
Wood	1.77	$\{\{S_3, G_7\}, \{S_3, G_{13}\}, \{S_3, G_{20}\}\}$
Smith	2.5	$\{\{S_7, G_{11}\}, \{S_7, G_{16}\}\}$

$K' =$

AVG(grade)	
2.3	$\{G_7\}$
1.3	$\{G_{13}\}$
1.7	$\{G_{20}\}$
3.3	$\{G_{11}\}$
1.7	$\{G_{16}\}$

und die BACKCHASE-Phase liefert:

$I^* =$

studentID	lastName	firstName	studies
η_1	Wood	η_2	η_3
η_{10}	Smith	η_{11}	η_{12}

courseID	studentID	semester	grade
η_4	η_1	η_5	2.3
η_6	η_1	η_7	1.3
η_8	η_1	η_9	1.7
η_{13}	η_{10}	η_{14}	3.3
η_{15}	η_{10}	η_{16}	1.7

Dabei indizieren die Zeugenmengen $\{S_i, G_j\}$, dass das Anfrageergebnis aus zwei Quell-Relationen entstanden ist. Die Anzahl der Zeugenmengen pro Zeugenbasis gibt Auskunft über die Anzahl der Quell-Tupel. Die Side Table speichert zudem die konkreten Attributwerte des aggregierten Attributes `grade`. □

Fassen wir abschließend die Ergebnisse zur Berechnung der (minimalen) Teil-Datenbank im Kontext von ProSA noch einmal zusammen. Wir schließen mit einer Auflistung aller hieran beteiligten Veröffentlichungen sowie betreuten studentischen Porjekten und Abschlussarbeiten.

6.5. Zwischenfazit

Für die Berechnung der (minimalen) Teil-Datenbank haben wir eine Erweiterung des CHASE&BACKCHASE-Verfahrens entwickelt (siehe Abschnitt 6.4), welche insbesondere die ***why***-Provenance sowie zusätzliche Side Tables in das Verfahren integrieren. Diese zusätzlichen Provenance-Informationen beeinflussen die CHASE-Inversen der jeweiligen Auswertungsoperationen. Insgesamt ergeben sich hieraus vier Klassen von Inversentypen (siehe Abschnitt 6.3), auf welche wir auch in Abschnitt 7.3 noch einmal zu sprechen kommen werden. Diese beschreiben, für welche Auswertungsoperatoren zusätzliche Provenance-Informationen relevant sind und in welchen Fällen auf diese Informationen verzichtet werden kann.

Die Berechnung der inversen Schemaabbildung $\mathcal{M}^*$ zu einer gegebenen Schemaabbildung $\mathcal{M}$ haben wir in Abschnitt 6.2 vorgestellt. Sie basiert auf dem Maximum Extended Recovery-Algorithmus aus [Fag+11a] und bildet die Grundlage für die spätere Invertierung in ProSA (siehe Abschnitt 10.3).

Die Definition der verschiedenen CHASE-Typen ermöglicht uns zudem die Angabe eines Gütekriteriums für die Replizierbarkeit bzw. Reproduzierbarkeit eines wissenschaftlichen Ergebnisses. Hierauf werden wir in Abschnitt 11.2 noch einmal eingehen.

Eigene sowie betreute Arbeiten

Die in diesem Kapitel vorgestellten Ergebnisse basieren auf den folgenden Veröffentlichungen und betreuten studentischen Arbeiten:

- [Aug17] **Tanja Auge**: Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen. *Masterarbeit*, 2017
- [AH18c] **Tanja Auge**, Andreas Heuer: The Theory behind Minimizing Research Data — Result equivalent CHASE-inverse Mappings. *LWDA*, 2018
- [AH18b] **Tanja Auge**, Andreas Heuer: Inverse im Forschungsdatenmanagement. *LWDA*, 2018
- [Aug20] **Tanja Auge**: Extended Provenance Management for Data Science Applications. *PhD@VLDB*, 2020
- [Zim21] Jakob Zimmer: Datenintegration durch inverse Schemaabbildungen: Erweiterung der Rostocker GaLVE-Technik. *Masterarbeit*, 2021
- [Spo22] Dennis Spolwind: Inverse Anfragen in ProSA. *Masterarbeit*, 2022
- [AH22a] **Tanja Auge**, Andreas Heuer: Enhanced Inversion of Schema Evolution with Provenance. (Submitted for Publication)

Der in ProSA verwendete Algorithmus zur Invertierung der (s-t) tgds (siehe Abschnitt 6.2) wurde insbesondere in den beiden Arbeiten [Zim21] und [Spo22] entwickelt, jedoch noch einmal für den konkreten Anwendungsfall angepasst. Die Klassifizierung der häufigsten Auswertungsoperatoren nach ihren Chase-Inversentypen aus Abschnitt 6.3 basiert auf den Chase-Inversen aus [Fag+11b] sowie den Definitionen der tp-relaxten und ergebnisäquivalenten Chase-Inversen aus Abschnitt 6.1. Die Aussagen dieser beiden Abschnitte sowie der in ProSA entwickelte Chase&Backchase (siehe Abschnitt 6.4) sind zudem in [Aug17; AH18a; AH18c] und [AH22a] veröffentlicht.

7. Provenance und Evolution

In vielen Fällen reicht die Rekonstruktion der „einfachen“ (minimalen) Teil-Datenbank jedoch nicht aus. Insbesondere bei bitemporalen Datenbanken interessiert uns nicht die „originale“ (minimale) Teil-Datenbank I^*, d.h. die Datenbank zum Zeitpunkt t, sondern vielmehr die entwickelte Teil-Datenbank J^* zum Zeitpunkt $t+1$. Schauen wir uns hierzu noch einmal die Situation der jungen Wissenschaftlerin aus der Einleitung (siehe Abschnitt 1.1; [AH21]) vor: Sie ist besorgt über eine Vielzahl an Zeitungsartikeln mit den Titeln „Todeszonen in der Ostsee“ oder „Todeszonen nehmen rasch zu: Stirbt die Ostsee?“. Sie möchte sich daher ein eigenes Bild über die Situation machen und entscheidet sich dafür, eine kleinen Studie durchzuführen. Um die Entwicklung des Sauerstoffgehalts der letzten 100 Jahren beurteilen zu können, plant sie, die Ergebnisse aller bisher durchgeführten Studien erneut zu untersuchen. Die junge Wissenschaftlerin möchte die alten Studienergebnisse in ihrer Studie selbst reproduzieren, was wir unterstützen möchten.

Schauen wir uns hierzu zunächst die allgemeine Vorgehensweise zur Berechnung einer (minimalen) Teil-Datenbank bei sich ändernden Datenbanken am Beispiel unsrer jungen Wissenschaftlerin einmal genauer an (siehe Abschnitt 7.1). Anschließend untersuchen wir in Abschnitt 7.2, welche Schema-Änderungen in Kontext des Forschungsdatenmanagements überhaupt von Bedeutung sind. Wir konzentrieren uns hierbei insbesondere auf die Ergebnisse unserer Zusammenarbeit mit dem Leibniz-Institut für Ostseeforschung Warnemünde, welche wir [AH21; Aug+20] zusammengefasst haben. Nennenswert sind in diesem Zusammenhang auch [Man20; Yan18] sowie [Bru+17]. Da eine Schema-Evolution in der Regel nicht ohne Informationsverlust erfolgen kann, bestimmen wir anschließend die Chase-Inversentypen der einzelnen Schema-Evolutionsperatoren (siehe Abschnitt 7.3), um so ein Gütekriterium für den Informationsverlust angeben zu können. Da sich neben dem Schema auch der Datentyp eines Attributes über die Zeit ändern kann, definieren wir zudem einen weiteren Provenance-Typ, die ***what***-Provenance (siehe Abschnitt 7.4). Doch schauen wir uns zunächst den allgemein Ablauf an.

7.1. Berechnung der (minimalen) Teil-Datenbank bei sich ändernden Datenbanken *(basierend auf [AH21])*

Wir verbleiben im Anwendungsgebiet des Forschungsdatenmanagements: In vielen Fällen speichern die Forschungsinstitute ihre Forschungsdaten persistent für jedes Forschungsprojekt sowie in einer sich entwickelnden Institutsdatenbank. Für eine konkrete Auswertung steht dann stets die originale Datenbankinstanz $I(S_t)$ und später eine aktuelle materialisierte Sicht $J(S_{t+1})$ zur Verfügung (siehe Abbildung 7.1). Die originale Instanz I zum Zeitpunkt t steht nach Ablauf des Projekts hingegen in der Regel nicht mehr separat zur Verfügung. Die Gründe hierfür sind vielseitig: So können sich insbesondere bei Langzeitstudien neben dem Schema selbst auch der Speicherort der Daten oder das Speichermedium geändert haben. Im Falle unserer jungen Wissenschaftlerin stellen wir uns also vor, dass uns statt der originalen Daten zum Zeitpunkt t lediglich die aktuelle Institutsdatenbank zur Verfügung steht. Da wir unserer jungen Kollegin diese jedoch nicht vollständig zur Verfügung stellen können, suchen wir insbesondere die Daten J^*, welche zur Reproduktion des ursprünglichen Ergebnisses notwendig sind. Gesucht ist also eine spezielle materialisierte Sicht auf die aktuelle Institutsdatenbank.

Konkret müssen wir die alte Datenbankinstanz $I(S_t)$ zunächst aus $J(S_{t+1})$ zurück berechnen, um die alte Auswertungsanfrage Q erneut ausführen zu können. Die wiederholte Ausführung von Q ist notwendig, um zusätzliche Informationen für die inverse Anfrage Q^{-1}_{prov} zu erhalten. Die rekonstruierten Quell-Tupel aus I^* (rot hervorgehoben) werden anschließend in die materialisierte Sicht J^* evolutioniert (blau hervorgehoben). Um das selbe Ergebnis K (gesamtes Anfrageergebnis) bzw. K^* (ausgewählte Tupel des Anfrageergebnisses) zu erhalten (grün hervorgehoben), muss die Anfrage Q abschließend noch zu Q' transformiert werden. Wiederholt unsere junge Wissenschaftlerin dieses Vorgehen für verschiedene Anfragen, erhält sie eine etwas größere Teil-Datenbank J^*_{All}, welche einen ganzen Pool an Auswertungen ermöglicht.

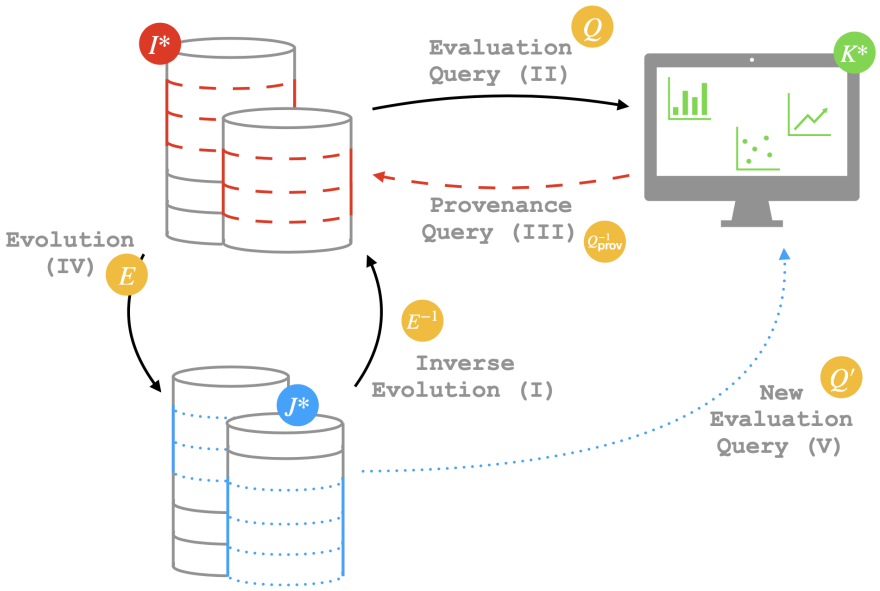


Abbildung 7.1. Kombination von Auswertungsanfrage, Evolution und Provenance, [AH21].

Für die Berechnung der (minimalen) Teil-Datenbank J^* bei sich ändernden Datenbanken klassifizieren wir fünf Hauptschritte. Der Prozess wird erstmals in [AH18a] vorgestellt und in [AH21] weiter verfeinert und erweitert. Insgesamt ergeben sich folgende Schritte:

I. Rekonstruktion der ursprünglichen Datenbank durch Umkehrung der Evolution

II. Berechnung des Anfrageergebnisses(grün hervorgehoben)

III. Rekonstruktion der Quell-Tupel basierend auf dem Anfrageergebnis und zusätzlichen Provenance-Informationen (rot hervorgehoben)

IV. Entwicklung der (minimalen) Teil-Datenbank (blau hervorgehoben)

V. Transformation der Auswertungsanfrage

Ist die Herkunft der Auswertungsanfrage bereits bekannt, d.h. die originale Datenbank I liegt noch vor, können die Schritte I. und II. vernachlässigt werden.

Abbildung 7.1 beschreibt diesen Prozess grafisch. Ausgehend von der aktuellen materialisierten Sicht J (unten links) liefert die Invertierung der Evolution (**I**) die ursprüngliche Datenbankinstanz I (oben links). Das Ergebnis einer Auswertungsanfrage (grün hervorgehoben) kann mit Hilfe zusätzlicher Provenance-Informationen invertiert werden (**II** + **III**). Es entsteht eine (minimale) Teil-Datenbank I^* (rot hervorgehoben), welche für die Reproduktion bzw. Replikation des Anfrageergebnisses K notwendig ist. Im Falle einer temporalen Datenbank muss die (minimale) Teil-Datenbank J^* (blau hervorgehoben) mit Hilfe der (inversen) Evolution aus der aktuellen materialisierten Sicht berechnet werden (**I** + **IV**). Die neue Auswertungsanfrage ergibt sich somit als Kombination der inversen Evolution und der ursprünglichen Auswertungsanfrage (**V**).

I. Rekonstruktion der ursprünglichen Datenbank durch Umkehrung der Evolution Mit Hilfe der inversen Evolution E^{-1} kann die alte, ursprüngliche Datenbank $I(S_t)$ aus der aktuellen materialisierten Sicht $J(S_{t+1})$ berechnet werden. Wir erhalten also: $I(S_t) = E^{-1}(J(S_{t+1}))$. Hierfür werden die Evolution E und ihre (exakte) Inverse E^{-1} als Menge von ein oder zwei s-t tgds formuliert und vom Chase&Backchase verarbeitet. Alle neuen Tupel, welche in der originalen Instanz I noch nicht vorhanden waren — erkennbar an ihrer (Provenance-)ID — werden gelöscht. Die verbleibenden Attribute werden mit der inversen Evolution E^{-1} weiterverarbeitet. Einige Operationen wie das Erstellen neuer Relationen, die Umbenennung oder das Einfügen neuer Attribute sind leicht zu invertieren und können ohne weiteren Informationsverlust durchgeführt werden. Andere Operationen verzeichnen durch Dangling Tuple oder Duplikateliminierung einen Informationsverlust, welcher durch zusätzliche Provenance-Annotationen in einigen Fällen jedoch reduziert werden kann. Details folgen in Abschnitt 7.3.

II. Berechnung des Anfrageergebnisses Die Auswertungsanfrage Q kann als Menge von (s-t) tgds formuliert und ebenfalls durch den Chase verarbeitet werden. Die hieraus resultierende Datenbankinstanz $K^*(T) = Q(I)$ ist in Abbildung 7.1 grün hervorgehoben. Im Kontext des Forschungsdatenmanagments gilt weiterhin $K^* = K$, da in der Regel das gesamte Anfrageergebnis reproduziert bzw. repliziert werden soll.

III. Rekonstruktion der Quell-Tupel basierend auf dem Anfrageergebnis und zusätzlichen Provenance-Informationen Das Ergebnis der Auswertungsanfrage Q kann wie in Kapitel 6 beschrieben durch Anwendung des Chase berechnet werden. Die anschließende Konstruktion der (minimalen) Teil-Datenbank I^* erfolgt durch Invertierung der Auswertungsanfrage Q. Die Inverse Q^{-1}_{prov} liefert mit dann mittels Backchase die (minimale) Teil-Datenbank $I^*(S_t) = Q^{-1}_{\text{prov}}(K^*(T))$ (siehe Abbildung 7.1, rot hervorgehoben). Die (minimale) Teil-Datenbank ist korrekt berechnet, wenn ihre Auswertung das ursprüngliche Anfrageergebnis liefert, d.h. $Q(I) = Q(I^*)$. Wie gut die Teil-Datenbank mit der Original-Datenbank übereinstimmt, hängt dabei unter anderem vom Chase-Inversentyp der Auswertungsanfrage Q ab. Inwiefern dieser durch zusätzliche Provenance-Informationen beeinflusst werden kann, kann in Abschnitt 6.3 sowie [AH18c] nachgelesen werden.

IV. Entwicklung der (minimalen) Teil-Datenbank Ist die (minimalen) Teil-Datenbank I^* erfolgreich berechnet worden, wird diese durch Chasen der Evolution in die aktuelle materialisierte Sicht $J^*(S_{t+1}) = E(Q^{-1}_{\text{prov}}(K(T)))$ transformiert (Abbildung 7.1, blau hervorgehoben). Diese können wir nun der jungen Wissenschaftlerin zur Verfügung stellen. Wegen $Q'(J^*(S_{t+1})) = Q(I(S_t))$, kann das ursprüngliche Anfrageergebnis K^* nun fehlerfrei aus der (minimalen) materialisierten Sicht J^* rekonstruiert werden.

V. Transformation der Auswertungsanfrage Nach Rekonstruktion der (minimalen) Teil-Datenbank J^* müssen wir abschließend noch die Anfrage Q transformieren, da Q in der Regel auf das neue Schema S_{t+1} nicht direkt angewandt werden kann. Dabei ergibt sich die neue Anfrage Q' als Komposition der ursprünglichen Auswertungsanfrage Q sowie der inversen Evolution E^{-1}, d.h.

$$Q'(J^*(S_{t+1})) = Q(E^{-1}(S_{t+1})).$$

Kommen wir noch einmal zu unserem Beispiel mit der jungen Wissenschaftlerin zurück: Nehmen wir an, wir können ihr unter den genannten Bedingungen (d.h. unter Berücksichtigung der Umkehrbarkeit von Evolution und Auswertung) alle notwendigen Informationen zur Verfügung stellen. Dann ist sie dank der (minimalen) Teil-Datenbank unserer materialisierten Sicht J^* sowie der transformierten Anfrage Q' in der Lage, die ursprünglich veröffentlichten Forschungsergebnisse zu rekonstruieren. Der Grad der Genauigkeit hängt dabei von der Genauigkeit der entsprechenden Inversen Q^{-1}_{prov} und E^{-1} ab. Dies kann durch die Chase-Inversen, vorgestellt in Abschnitt 7.3, klassifiziert werden. Details hierzu folgen in Abschnitt 11.2. Schauen wir uns zuvor die Entwicklung des Sauerstoffgehalts der Ostsee noch einmal am konkreten Beispiel an.

Beispiel 7.1. Wir interessieren uns für die Entwicklung des Sauerstoffgehalts $\texttt{O}_2$ im Jahr 1977. Leider hat sich das Format des Datums im Laufe der Jahre geändert. Während es 1977 aus drei separaten Attributwerten vom Typ `VARCHAR` bestand, wird es jetzt in einem gemeinsamen Datum vom Typ `DATE` gespeichert. Um die zur Ermittlung des Sauerstoffgehalts im Jahr 1977 notwendigen Quell-Tupel der Relation $\texttt{Serie}_{2017}$ bestimmen zu können, benötigen wir lediglich die beiden Anfragen Q und E sowie eine materialisierte Sicht $J(S_{t+1})$. Seien hierfür $Q = \pi_{\texttt{ID},\texttt{O}_2}(\texttt{Serie})_{1977}$, $E = \texttt{MERGE Column date AS FUNC (year, month, day) INTO Series}_{2017}$ sowie $J(S_{t+1}) = \texttt{Serie}_{2017}$ gegeben.

Als erstes überführen wir die materialisierte Sicht J der Relation `Serie` aus dem Jahre 2017 zurück in das Jahr 1977. Dafür invertieren wir die durch die Evolution E durchgeführten Änderungen mittels E^{-1}, dargestellt in Tabelle 7.1a. Anschließend führen wir die Auswertungsanfrage Q aus und invertieren das Ergebnis mit Hilfe zusätzlicher Provenance-Informationen.[1] In Tabelle 7.1b verwenden wir hierfür die Provenance-Polynome der

[1] Da wir die Auswertungsanfrage Q mit zusätzlichen Provenance-Informationen versehen haben, ist dies für die Evolution E in ProSA nicht mehr nötig. Sie wird „indirekt“ mit verarbeitet. Grund hierfür ist, dass wir die Provenance-Berechnung in ProSA in zusätzliche Provenance-Attribute auslagert haben. Lediglich die Evolutionsperatoren `MERGE Column` sowie `SPLIT Column` können wegen der zusätzlichen Side Tables auf diese Weise nicht verarbeitet werden.

how-Provenance. Die Zeugenbasen der ***why***-Provenance wären ebenfalls denkbar. Die Zeugenlisten der ***where***-Provenance empfehlen wir aufgrund möglicher Duplikate und Dangling Tuple jedoch zu vermeiden. Grund hierfür ist, dass die ***where***-Provenance lediglich die Provenance-IDs der im Resultat existierenden Tupel ausgibt, nicht jedoch die IDs möglicher Zwischenergebnisse.

Die so erzeugte (minimale) Teil-Datenbank I^* kann anschließend in das Schema von 2017 zurück transformiert werden. Insgesamt erhalten wir J^* aus Tabelle 7.1c. Dabei gilt: Durch die Projektion auf $\texttt{O}_2$ sowie die `ID` gehen alle Informationen über die Attribute `day` und `month` verloren. Diesen Informationsverlust gleichen wir durch das Einfügen zusätzlicher Nullwerte η_i aus. Nur der Wert von `year` kann aus der blauen Annotationen $s_{i,1977}$ rekonstruiert werden. Statt der ursprünglichen materialisierten Sicht $J = \texttt{Series}_{2017}$ genügt es also, die (minimale) materialisierte Sicht $J^* = \texttt{Series}^*_{2017}$ mit den rekonstruierten Quell-Tupeln 1, 3 und 4 abzuspeichern, sofern die Auswertungsanfrage Q sowie die Evolution E ebenfalls bekannt sind. □

ID	$\texttt{O}_2$	date	
1	7.0	1977-03-02	$s_{1,2017}$
2	8.0	1970-03-11	$s_{2,2017}$
3	8.0	1977-06-11	$s_{3,2017}$
4	8.1	2000-11-12	$s_{4,2017}$

$\texttt{Series}_{2017}$ $\xrightarrow{E^{-1}}$

ID	$\texttt{R\_O}_2$	day	month	year	
1	7	2	3	1977	$s_{1,1977}$
2	8	11	3	1970	$s_{2,1977}$
3	8	11	6	1977	$s_{3,1977}$

$\texttt{Series}_{1977}$

(a) Inverse Evolution E^{-1}

ID	$\texttt{R\_O}_2$	day	month	year	
1	7	2	3	1977	$s_{1,1977}$
2	8	11	3	1970	$s_{2,1977}$
3	8	11	6	1977	$s_{3,1977}$

$\texttt{Series}_{1977}$ $\xrightarrow{Q}$

day	$\texttt{R\_O}_2$	
2	7	$s_{1,1977}$
11	8	$s_{2,1977}$ $+s_{3,1977}$

`Result` $\xrightarrow{Q_{\text{prov}}}$

ID	$\texttt{R\_O}_2$	day	month	year	
η_1	7	2	η_2	1977	$s_{1,1977}$
η_3	8	11	η_4	1977	$s_{2,1977}$
η_5	8	11	η_6	1977	$s_{3,1977}$

$\texttt{Series}^*_{1977}$

(b) Auswertungsanfrage Q und Provenance-Anfrage Q^{-1}_{prov}

ID	$\texttt{R\_O}_2$	day	month	year	
η_1	7	2	η_2	1977	$s_{1,1977}$
η_3	8	11	η_4	1977	$s_{2,1977}$
η_5	8	11	η_6	1977	$s_{3,1977}$

$\texttt{Series}^*_{1977}$ $\xrightarrow{E}$

ID	$\texttt{O}_2$	date	
η_1	7.0	1977-2-η_1	$s_{1,2017}$
η_3	8.0	1977-11-η_3	$s_{3,2017}$
η_5	8.0	1977-11-η_5	$s_{4,2017}$

$\texttt{Series}^*_{2017}$

(c) Evolution E

Tabelle 7.1. Berechnung der (minimalen) Teil-Datenbank $J^* = \texttt{Series}^*_{2017}$ auf Basis einer Anfrage $Q = \pi_{\texttt{day},\texttt{O}_2}(\texttt{Series}_{1977})$ und dessen Anfrageergebnisses `Result` [AH21].

Die Entwicklung einer Datenbankinstanz $I(S_t)$ in eine neue materialisierte Sicht $J(S_{t+1})$ kann durch verschiedene Schemaänderungen beschrieben werden, formalisiert durch die in [GZ12] vorgestellten *Schemamodifikationsoperatoren* (SMOs). Insgesamt unterscheiden wir fünfzehn dieser Operatoren. Wie wir im Folgenden zeigen werden, sind im Kontext des Forschungsdatenmanagements insbesondere vier Operatoren von Bedeutung: `CREATE Table`, `ADD Column`, `MERGE Column` und `SPLIT Column`.

7.2. Klassifikation relevanter Evolutionsperatoren *(basierend auf [Aug+20])*

Schemaänderungen werden häufig über sogenannte *Schemamodifikationsoperatoren* (engl. Schema Modification Operators, SMOs) formalisiert. SMOs sind grundlegende Operatoren, welche zur Beschreibung von Schemaänderungen innerhalb einer Datenbank verwendet werden. Üblicherweise werden dreizehn Operatoren unterschieden, welche in Tabelle 7.2 zusammengefasst sind. Jeder Operator erfasst eine atomare Änderung, deren Kombinationen auch komplexere Schemaänderungen ausdrücken können. Die Operatoren werden je nach Anwendungsgebiet unterschiedlich häufig eingesetzt. Zu den relevantesten gehören nach [QLS13; Cur+08; WN11; Sjø93] — unabhängig vom konkreten Anwendungsfall — die Operatoren `CREATE Table`, `DROP Table`, `ADD Column`, `DROP Column` und `RENAME Column`. Diese sind in Tabelle 7.2 grau hinterlegt.

SMOs
COPY Table R INTO S
CREATE Table R(a,b,c)
DECOMPOSE Table R(a,b,c) INTO S(a,b), T(b,c)
DISTRIBUTE Table/PARTITION Table R INTO S WITH cond, T
DROP Table R
JOIN Table R, S INTO T WHERE cond
MERGE Table R, S INTO T
RENAME Table R INTO S

SMOs
ADD Column d [AS const|FUNC(a,b,c)] INTO R
COPY Column c FROM R INTO S WHERE cond
DROP Column c FROM R
MOVE Column c FROM R INTO S WHERE cond
RENAME Column b IN R TO d
NOP

Tabelle 7.2. Schemamodificationsoperatoren, nach [CMZ08; Moo+08].

Zusätzlich zu den SMOs existiert noch eine weitere Klasse von Schemaevolutionsoperatoren. Diese erstmals in [Cur+10] vorgestellten *Integrity Constraint Modification Operators* (ICMOs) beschreiben Änderungen der Integritätsbedingungen. Konkret beschreiben sie Änderungen des Primärschlüssels, des Fremdschlüssels sowie der Constraint Values. Wir werden sie im weiteren Verlauf dieser Arbeit vernachlässigen. Welche ICMOs in der Sauerstoff-Datenbank des IOW eine Rolle spielen, kann jedoch in [Man20] nachgelesen werden.

Häufig auftretende Schemaänderungen im Kontext des Forschungsdatenmanagements Datenbanken, welche sich über einen Zeitraum von mehreren Jahren oder Jahrzehnten entwickeln, unterliegen zwangsläufig gewissen Schemaänderungen. Üblicherweise finden dabei 80% der Schemaänderungen in etwa 20-30% der Tabellen statt, während fast 40% der Tabellen sich nicht ändern [QLS13]. Für diese Feststellung untersuchten die Autoren zehn große populäre Open-Source-Datenbankanwendungen wie Joomla!, TYPO3 oder Mediawiki mit insgesamt mehr als 160k Revisionen auf die Koevolution ihrer Datenbankschemata sowie ihres Codes.

Schemaänderungen werden üblicherweise über die oben definierten Schemamodifikationsoperatoren beschrieben. Eine Klassifikation der wichtigsten Schemamodifikationsoperatoren kann in der Literatur nachgelesen werden. Für die Analyse eingebetteter Datenbanken sei hier unter anderem [WN11] genannt. In [Cur+08] ist zudem eine Studie der Schemaänderungen in Wikipedia zu finden. Beide Arbeiten klassifizieren die Operatoren `ADD Column` und `DROP Column` als besonders relevant. So bestehen knapp 40% der Schemaänderungen im Hinzufügen von Attributen und etwa 25% im Löschen von Attributen. Etwa 25% machen zudem das Umbenennen und Erzeugen neuer Tabellen aus. Alle übrigen Schemaänderungen treten verhältnismäßig selten auf. Auch Typ-Änderungen können auftreten [Cur+08].

Anders verhält es sich im Forschungsdatenmanagement. Wie in Kapitel 2 beschrieben, werden in diesem Falle Daten quasi nie gelöscht. Sie unterliegen hingegen Änderungen wie dem Verschmelzen, Aufsplitten, Kopieren oder Verschieben von Datensätzen. Auch das Hinzufügen von Attributen oder Erstellen neuer Tabellen findet immer wieder statt. Dies zeigen auch unsere Untersuchungen des Sauerstoff-Beispiels des IOW. Unsere Analyse der sogenannten *CTD-Daten* ergibt lediglich vier Schemaänderungen, von denen wiederum das Hinzufügen und Löschen von Attributen die am häufigsten zu beobachtende Änderung darstellt, hier immerhin 72% bzw. 24% der vorgenommenen Operationen (siehe Tabelle 7.3a). Die CTD-Sonde, welche die CTD-Daten aufnimmt, ist ein spezielles Messgerät zur Bestimmung der elektrischen Leitfähigkeit (**C**onductivity), welche zur Bestimmung des Salzgehalts, der Wassertemperatur (**T**emperature) sowie der Tiefe (**D**epth) verwendet wird. Details zur Analyse der CTD-Daten können beispielsweise in Abschnitt [Man20] sowie auf den Seiten des IOW (`https://www.io-warnemuende.de`) nachgelesen werden.

	# changes	percentage share
CREATE Table	1	1.6%
ADD Column	45	72.6%
DROP Column	15	24.2%
RENAME Column	1	1.6%

(a) klassische SMOs, [Aug+20]

	# changes	percentage share
CREATE Table	1	2.3%
ADD Column	35	79.5%
DROP Column	0	0%
RENAME Column	1	2.3%
MERGE Column	4	9.2%
SPLIT Column	3	6.8%

(b) erweitert um `MERGE Column` und `SPLIT Column`

Tabelle 7.3. Klassifizierte Schemaänderungen für das Sauerstoff-Beispiel am IOW.

Neben den Schemaänderungen, welche durch die Operatoren aus [Cur+10; GZ12] beschrieben werden können, definieren wir zwei neue Operatoren: `MERGE Column` und `SPLIT Column`. Diese entsprechen einer speziellen Hintereinanderausführung von `ADD Column`- und `DROP Column`-Operatoren. Wie in Tabelle 7.3b zu sehen, sind etwa

15% aller Schemaänderungen auf Merging- oder Splitting-Änderung zurückzuführen. Sie stellen daher neben dem Hinzufügen von Attributen die größten Änderungen innerhalb der untersuchten IOW-Datenbank dar. So entfallen zehn der `ADD Column`-Operatoren sowie alle 15 `DROP Column`-Operatoren auf die beiden neu eingeführten Operatoren `MERGE Column` sowie `DROP Column`.

`MERGE Column` und `SPLIT Column` Für die Verschmelzung sowie das Aufsplitten von Attributen definieren wir zwei neue Modifikationsoperatoren. Beide können als Hintereinanderausführung bereits bekannter SMOs sowie der Einarbeitung verschiedener Hilfsfunktionen verstanden werden. Die zugehörigen Operatoren `MERGE Column` und `SPLIT Column` sind in Tabelle 7.4 zusammengefasst. Sie bestehend aus einer Hintereinanderausführung verschiedener `ADD Column` und `DROP Column` Operatoren kombiniert mit einer Merging- oder Splitting-Funktion.

`MERGE Column` a,c `AS FUNC`(a,c) `IN` R `TO` d
`ADD Column` d `AS FUNC`(a,c) `INTO` R; `DROP Column` a `FROM` R; `DROP Column` c `FROM` R;

(a) `MERGE Column`

`SPLIT Column` a `IN` R `TO` d `USING FUNC`$(a)_1$, e `USING FUNC`$(a)_2$
`ADD Column` d `AS FUNC`$_1$(a) `INTO` R; `ADD Column` e `AS FUNC`$_2$(a) `INTO` R; `DROP Column` a `FROM` R;

(b) `SPLIT Column`

Tabelle 7.4. SMOs für `MERGE Column` und `SPLIT Column`.

Wie in Tabelle 7.4 zu sehen, wird das Zusammenführen zweier Spalten mittels `MERGE Column` durch die aufeinanderfolgende Ausführung von `ADD Column` und `DROP Column` realisiert. Für die Erstellung der neuen Attributwerte wird eine Funktion `FUNC` verwendet, welche die alten Attribute kombiniert. Dementsprechend besteht das Zusammenführen darin, eine neue Spalte zu erstellen und anschließend die alten zu löschen.

Das Aufsplitten eines Attributes in zwei neue Attribute hat eine ähnliche Struktur wie das Verschmelzen zweier Attribute. Auch hier werden neue Spalten mittels `ADD Column` angelegt und alte mittels `DROP Column` gelöscht. Dabei wird jede neue Spalte durch eine eigene Funktion `FUNC`$_i$ erzeugt. Wir bemerken zudem, dass `MERGE Column` und `SPLIT Column` (quasi-)invers zueinander sind.

Beispiel 7.2. Betrachten wir noch einmal unser eingangs vorgestelltes Sauerstoff-Beispiel. Im Laufe der letzten 100 Jahre hat sich neben dem Speicherformat selbst auch das -schema immer wieder verändert. So werden Attribute immer wieder zusammengefasst und/oder aufgespalten (siehe Tabelle 7.3). Schauen wir hierzu exemplarisch das Datum an. Während es 1977 noch in drei getrennten Attributen `day`, `month` und `year` abgespeichert wird, existiert 2017 nur noch ein zusammengesetztes Datum vom Typ `DATE`. Diese Schemaänderung kann durch

```
MERGE Column beginDateAS FUNC(startYear,startMonth,startDay) INTO Series
```

beschrieben werden. Dies entspricht der Hintereinanderausführung von

```
ADD Column  beginDate AS  FUNC(startYear,startMonth,startDay)  INTO Series
DROP Column  startYear FROM Series;
DROP Column  startMonth FROM Series;
DROP Column startDay FROM Series
```

mit `FUNC` := `CONCAT`(`startYear`, '-', `startMonth`, '-', `startDay`). Andere Hilfsfunktionen zum Verschmelzen der Attributwerte sind selbstverständlich ebenfalls möglich. Analog kann für `SPLIT Column` verfahren werden. □

Im Falle des Sauerstoff-Beispiels konnten wir insgesamt sieben Merge- und Split-Vorgänge identifizieren [Aug+20] und damit 25 `ADD Column` bzw. `SPLIT Column`-Operationen „einsparen“. Wir erhalten so die in Tabelle 7.3b dargestellte neue Verteilung. Wir stellen fest, dass `ADD Column` weiterhin der am häufigsten auftretende Operator bleibt. Den Operator `DROP Column` benötigen wir jedoch nicht mehr oder zumindest nicht mehr explizit. Dies reduziert die Gesamtanzahl der Änderungen zwar nicht, beschreibt die vorliegenden Schemaänderungen aber spezifischer.

7.3. Evolutionsoperatoren und ihre Chase-Inversen *(nach [AH22a])*

Die Rekonstruktion von (veröffentlichten) Evaluierungsergebnissen für sich entwickelnde Datenbanken ist ebenso ein Garant für ein gutes Forschungsdatenmanagement wie die langfristige Bereitstellung der Daten selbst. Basierend auf den in Abschnitt 7.1 und [AH18c] vorgestellten Ergebnissen konzentrieren wir uns hier auf die Kombination der Auswertungsanfragen Q mit der Schema-Evolution E. Beide Schritte werden mit der gleichen Technik, dem Chase (siehe Definition 3.22), verarbeitet.

Neben den in Abschnitt 3.2 vorgestellten Auswertungsoperatoren unterscheiden wir verschiedene *Schemamodifikationsoperatoren* (SMOs, siehe Abschnitt 3.5), welche die Schemaänderungen innerhalb der temporalen Datenbank beschreiben. Für die Rekonstruktion der originalen Datenbank (siehe Abbildung 7.1, Schritt **I**) benötigen wir eine konkrete inverse Evolutionsanfrage E^{-1}. In vielen Fällen ist dies jedoch nicht möglich. Wir transformieren die SMOs daher zunächst ein eine Menge von (s-t) tgds und invertieren diese anschließend. Die so erhaltenen inversen Abhängigkeiten werden nun auf ihre Chase-Inversen untersucht, welche wir in einem weiteren Schritt um zusätzliche Annotationen wie ***why***-Provenance und Side Tables erweitern. Dadurch können wir in einigen Fällen eine „bessere“ Chase-Inverse angeben als zuvor. So ändert sich für `MERGE Table` beispielsweise der Chase-Inversentyp von ergebnisäquivalent zu exakte (siehe Tabelle 7.6 und Tabelle 7.7). Der Inversentyp gilt dabei wieder als Maßeinheit für den erwarteten Informationsverlust (siehe Abschnitt 11.2).

Zu diesem Zweck klassifizieren wir die in Abschnitt 7.2 vorgestellten SMOs bezüglich ihres durch die Invertierung entstehenden Informationsverlusts und optimieren diese, sofern möglich, durch Hinzunahme zusätzlicher Provenance-Informationen. Insgesamt ergeben sich so die folgenden vier Klassen: Klasse I (siehe Tabelle 7.5) umfasst alle Operatoren mit einer exakten Chase-Inversen. Wir nennen diese auch *provenance-invariant*. Die Operatoren der anderen drei Klassen (II bis IV) verzeichnen alle einen Informationsverlust. Im Falle der Klasse II wird dieser durch Duplikate und im Falle von Klasse III durch Dangling Tuple verursacht.

Chase&Backchase Um die Schema-Evolution in die Berechnung unserer (minimalen) Teil-Datenbank zu integrieren (siehe Abbildung 7.1), sind zunächst einige Vorarbeiten notwendig. Sei hierfür $S_1, ..., S_n$ eine Sequenz von Schemata einer sich ändernden Datenbank. Sei weiter E ein Evolutionsschritt von S_t nach S_{t+1} und E^{-1} die zugehörige Inverse. Beide können als Schemaabbildung $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ bzw. $\mathcal{M}^{-1} = (S_{t+1}, S_t, \Sigma^{-1})$ aufgefasst werden. Dabei formalisieren Σ bzw. Σ^{-1} E bzw. E^{-1} als Menge von (s-t) tgds. Dann berechnet der Chase&Backchase die (minimale) Teil-Datenbank I^* basierend auf den Instanzen I und J über S_t bzw. S_{t+1}:

$$I^* = \text{Chase}_{\mathcal{M}^*}(J) = \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)).$$

Die (minimale) Teil-Datenbank I^* ist Ausschnitt bzw. Teil-Instanz von I, d.h. es existiert ein Homomorphismus h von I^* nach I, welcher alle Konstanten auf sich selbst und alle Nullwerte aus I^* auf (andere) Nullwerte oder Konstanten aus I abbildet. Zudem gilt wiederum $Q(I^*) = Q(I)$, d.h. die Auswertung der Teil-Datenbank I^* liefert dasselbe Ergebnis wie die Auswertung der originalen Datenbankinstanz I.

Für eine möglichst verlustfreie Rekonstruktion der Quell-Tupel ergänzen wir den Chase&Backchase um zusätzliche Provenance-Informationen (siehe Algorithmus 4) wie die ***why***-Provenance, die ***what***-Provenance (siehe Abschnitt 7.4) sowie Side Tables. Hierfür erweitern wir zunächst alle Quell-Tupel durch eine global eindeutige Provenance-ID (türkis hervorgehoben), welche in der Chase-Phase zur Berechnung der Zeugenbasis der ***why***-Provenance verwendet wird. Sei hierfür `MERGE Table R, V INTO R` eine gegebene Evolution und $I = \{R(1, \text{Alice}, \text{IT}, t_1), R(2, \text{Bob}, \text{Math}, t_2), V(1, \text{Alice}, \text{IT}, t_3)\}$ eine Quell-Instanz. Dann liefert die Chase-Anwendung $J = \{V(1, \text{Alice}, \text{IT}), V(2, \text{Bob}, \text{Math})\}$ ohne sowie $J = \{V(1, \text{Alice}, \text{IT}, t_1), V(2, \text{Bob}, \text{Math}, t_2), V(1, \text{Alice}, \text{IT}, t_3)\}$ mit zusätzlichen Provenance-Informationen. Hierfür erweitern wir neben den Quell-Tupeln auch die Evolutionsanfrage:

$$\begin{aligned} R(\texttt{ID}, \texttt{name}, \texttt{subject}, t_i) &\rightarrow T(\texttt{ID}, \texttt{name}, \texttt{subject}, t_i), \\ V(\texttt{ID}, \texttt{name}, \texttt{subject}, t_j) &\rightarrow T(\texttt{ID}, \texttt{name}, \texttt{subject}, t_j), \end{aligned}$$

wobei t_i der globalen Provenance-ID entspricht. Informationen, welche nicht über die ***why***-Provenance — hier $\{\{t_1\}, \{t_3\}\}$ für das erste Ergebnistupel $V(1, \text{Alice}, \text{IT})$ und $\{\{t_2\}\}$ für das zweite Ergebnistupel $V(2, \text{Bob}, \text{Math})$

— gesichert werden können, werden in zusätzlichen Side Tables abgespeichert. Diese enthalten die konkreten Attributwerte sowie ihre zugehörigen Provenance-IDs.

In der BACKCHASE-Phase berechnen wir die Teil-Datenbank I^* durch CHASEn von E^{-1} und J. Hier werden verlorengegangene Tupel wie $V(1, \text{Alice}, \text{IT})$ mit Hilfe der Zeugenbasen rekonstruiert oder überflüssige Tupel entfernt. Zudem können künstlich eingeführte Nullwerte durch die in den Side Tables gespeicherten Attributwerte ersetzt werden, sodass I^* möglichst verlustfrei rekonstruiert werden kann. Ein Maß für den Informationsverlust bietet hierbei die Bestimmung des CHASE-Inversentyps.

Algorithm 4 Invertierung von Evolutionsanfragen mittels ***why***-Provenance

Require: Quell-Instanz $I(S_t)$ und Evolutionsanfrage E von S_t nach S_{t+1}, formalisiert als Provenance-erweiterte Schemaabbildung $\mathcal{M}_P = (S_t, S_{t+1}, \Sigma_P)$
Ensure: (minimale) Teil-Datenbank $I^* \preccurlyeq I$ und CHASE-Inversentyp von $\mathcal{M}_P$
CHASE-*Phase:*

1. ergänze Quell-Tupel durch global eindeutige Provenance-IDs;
2. berechne das Anfrageergebnis $J(S_{t+1})$ mittels $\text{CHASE}_{\mathcal{M}_P}(I(S_t))$ mit $\mathcal{M}_P$ Provenance erweiterte Schemaabbildung;
3. speichere alle Attributwerte, die potentiell verlorenen gehen könnten, in zusätzlichen Side Tables

BACKCHASE-*Phase:*

1. berechne die Teil-Datenbank $I^*(S_t)$ mittels $\text{CHASE}_{\mathcal{M}_{P^{-1}}}(J(S_{t+1}))$;
2. ergänze alle verlorenen Tupel durch Auswertung der ***why***-Provenance;
3. ersetze verlorene Attributwerte durch zusätzliche Side Tables;
4. gib den stärksten verfügbaren CHASE-Inversentyp zurück

Wenn Provenance nicht erforderlich ist, werden die Schritte 1 und 3 in der CHASE-Phase und die Schritte 2 und 3 in der BACKCHASE-Phase in beiden Phasen weggelassen. Außerdem verwenden wir Σ anstelle von Σ_P.

Chase-Inverse für die Schema-Evolution Wir interpretieren die atomaren Änderungen, welche während einer Schemaänderung auftreten können, als *Schemamodifikationsperatoren*. Jeder der in Abschnitt 7.2 vorgestellten Operatoren kann als Menge von einer oder zwei s-t tgds dargestellt werden (siehe Tabelle 7.5). Anders als in [CMZ08; DT01] verzichten wir jedoch auf die Darstellung der SMOs als *disjunctive embedded dependencies* (ded). Diese Formalisierung ist zwar mächtiger als die Darstellung durch s-t tgds, kann vom Standard-CHASE jedoch nicht verarbeitet werden.

Einige Operatoren wie `PARTITION Table`, `COPY Column` und `MOVE Column` erfordern spezielle Bedingungen cond_A, welche Einschränkungen auf den Attributwerten vorgeben. Die entsprechenden s-t tgds sind auf Konstanten- und Attributselektion $\sigma_{A_i \theta c}$ bzw. der Form $\sigma_{A_i \theta A_j}$ mit $\theta \in \{<, \leq, =, \geq, >\}$ beschränkt. Andere Bedingungen können durch zusätzliche Side Tables formalisiert werden.

Side Tables enthalten Attributwerte, die für die Rekonstruktion der (minimalen) Teil-Datenbank erforderlich sind. Jedes Tupel ist eine Einschränkung eines Quell-Tupels t und wird durch seine Provenance-ID r_t eindeutig identifiziert. Diese ID wird nicht als Schlüssel, sondern als zusätzliche Provenance-Information interpretiert. Side Tables existieren also immer nur in Verbindung mit einer zugrundeliegenden Zeugenbasis.

Das Hinzufügen der Provenance-Informationen kann — ebenso wie bei den Auswertungsanfrage aus Abschnitt 6.3 — den CHASE-inversen Typ verbessern. In vielen Fällen ist dies jedoch gar nicht notwendig. So garantiert etwa die Hälfte der Operatoren bereits eine exakte CHASE-Inverse, ohne dass zusätzliche Informationen erforderlich sind. Die zugehörigen Operatoren bilden zusammen die erste von vier Klassen (siehe Tabelle 7.5), welche wir analog zu den Klassen der Auswertungsoperatoren definieren. Die übrigen Operatoren werden durch die zusätzlichen Informationen in ihrem Inversentyp beeinflusst. Grund hierfür ist der „natürliche“ Informationsverlust, welcher aber durch zusätzliche Provenance-Informationen reduziert werden kann. So kann der Inversentyp von `MERGE Table` beispielsweise von einer ergebnisäquivalenten CHASE-Inversen zu einer exakten CHASE-Inversen „aufgewertet“ werden. Es ist nur notwendig, die rekonstruierten Tupel auf R und S zu beschränken. Kennen wir die zugrundeliegende Zeugenbasis, ist dies aber überhaupt kein Problem.

SMO	Beschreibung	Inverse	Klasse
`COPY Table`	erstellt ein Duplikat einer bestehenden Tabelle	`DROP Table` `MERGE Table`	I
`CREATE Table`	führt eine neue, leere Tabelle ein	`DROP Table`	I
`DECOMPOSE Table`	zerlegt eine Quell-Tabelle in zwei Tabellen, ohne die darin gespeicherten Daten zu verändern	`ADD Column` `JOIN Table`	III
`DROP Table`	entfernt eine bestehende Tabelle	`CREATE Table`	IV
`JOIN Table`	kombiniert zwei Tupel aus verschiedenen Tabellen	`DECOMPOSE Table`	II
`MERGE Table`	nimmt zwei Quell-Tabellen und erstellt eine neue Tabelle, die deren Vereinigung speichert	`PARTITION Table`	IV
`PARTITION Table`	verteilt Tupel in zwei neu erstellte Tabellen, gemäß einer gegebenen Bedingung	`MERGE Table`	I
`RENAME Table`	ändert den Namen einer Tabelle	`RENAME Table`	I
`ADD Column`	erstellt eine neue Spalte mit Werten, die durch eine benutzerdefinierte Konstante oder Funktion erzeugt werden	`DROP Column`	I
`COPY Column`	kopiert eine Spalte in eine andere Tabelle, gemäß einer vorgegebenen Bedingung	`DROP Column`	I
`DROP Column`	entfernt eine vorhandene Spalte aus einer Tabelle	`ADD Column`	III
`MERGE Column`	Folge von `ADD Column` und `DROP Column` unter Verwendung einer benutzerdefinierten Funktion	`SPLIT Column`	III
`MOVE Column`	ähnlich wie `COPY Column`, jedoch mit Löschung der ursprüngliche Spalte	`MOVE Column`	II, III
`RENAME Column`	ändert den Namen einer Spalte	`RENAME Column`	I
`SPLIT Column`	Folge von `ADD Column` und `DROP Column` unter Verwendung einer benutzerdefinierten Funktion	`MERGE Column`	III
`NOP`	nichts passiert	`NOP`	I

Tabelle 7.5. Schemamodifikationsoperatoren mit zugehörigen Chase-Inversen und Provenance-Klassen.

Durch das Teilen, Zusammenführen und Löschen von Spalten können wie im Falle von `MERGE Table` Duplikate, Dangling Tuple oder fehlerhaft rekonstruierte Tupel entstehen. Wir unterteilen die Operatoren mit Informationsverlust daher in drei weitere Klassen ein. Klasse II zur Behandlung von Dangling Tuple, Klasse III verarbeitet Duplikate und Klasse IV alle weiteren Operatoren, welche einen Informationsverlust nach sich ziehen können. Im weiteren Verlauf dieses Abschnitts konzentrieren wir uns mit `COPY Table` (Klasse I), `JOIN Table` (Klasse II), `MERGE Table` (Klasse IV) und `MERGE Column` (Klasse III) jeweils auf einen Klassenvertreter. Die Ergebnisse sind in Tabelle 7.6 und Tabelle 7.7 zusammengefasst. Die Schemaevolutionsoperatoren sind ebenso wie ihre zugehörigen Inversen jeweils als Schemaabbildungen $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ formalisiert. Die Formalisierungen und Chase-Inversentypen ohne zusätzliche Provenance-Informationen sind in den Spalten 3 und 4 zu finden und die Formalisierungen und Inversentypen mit Provenance in den Spalten 5 und 6. Dangling Tuple sind pink, Duplikate blau und Erweiterungen durch zusätzliche Provenance-Informationen orange hervorgehoben. Die formalen Beweise aller 15 Operatoren sind im Anhang C zu finden. An dieser Stelle beschränken wir uns auf die oben genannte Auswahl.

Komposition verschiedener Operatoren Da die Schemaevolutionsoperatoren innerhalb einer Sequenz unabhängig voneinander arbeiten, können wir auch ihre Inversen unabhängig voneinander bestimmen. Die Inverse einer Komposition von Schemaabbildungen $\mathcal{M}^* = (\mathcal{M}_1 \circ ... \circ \mathcal{M}_n)^{-1} = \mathcal{M}_n^* \circ ... \circ \mathcal{M}_1^*$ ergibt sich somit aus dem Produkt der einzelnen Teil-Inversen $\mathcal{M}_1^*, ..., \mathcal{M}_n^*$. Der Inversentyp von $\mathcal{M}^*$ entspricht dabei höchstens dem Typ des schwächsten Teil-Inversen $\mathcal{M}_i^*$. Weiterhin ist zu beachten: Existiert für eine der Teil-Operationen $\mathcal{M}_i$ keine Inverse, so auch nicht für ihre Zusammensetzung $\mathcal{M} = \mathcal{M}_1 \circ ... \circ \mathcal{M}_n$. Es gilt Korollar 6.1.

Schauen wir uns dies an einem konkreten Beispiel etwas genauer an. Wie in Abschnitt 7.2 definiert, ist `MERGE Column` eine Folge von zwei Operatoren: `ADD Column` und `DROP Column`. Während der Inversentyp von `DROP Column` von der (Nicht-)Existenz von Duplikaten abhängig ist (Klasse III), hat `ADD Column` stets eine exakte Chase-Inverse (Klasse I). Der Inversetyp von `MERGE Column` entspricht also dem Inversentyp von `DROP Column`.

Die Komposition von Auswertungs- und Evolutionsperatoren verhält sich analog. Sei hierfür `SELECT year FROM R WHERE bonus` = 2000 eine Anfrage mit anschließender Zusammenführung der Attribute `day`, `month` und `year`. Seien weiter $\mathcal{M}_Q$ und $\mathcal{M}_E$ zwei Schemaabbildungen, welche die Auswertungsanfrage Q sowie die Evolutionsanfrage E formalisieren. Unter der Voraussetzung, dass durch die Zusammenführung der Daten keine Duplikate entstehen, hat $\mathcal{M}_E$ eine tp-relaxte Chase-Inverse und $\mathcal{M}_Q$ eine relaxte Chase-Inverse. Kombinieren wir beide, so erhalten wir eine relaxte Chase-Inverse.

Klassifikation der Schemamodifikationsoperatoren

Sei für die folgenden Beispiele und Beweise $S_1, ..., S_n$ eine Folge von Schemata einer sich ändernden Datenbank. Sei E ein Evolutionsschritt von S_t zu S_{t+1}. Seien ferner $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen, welche E bzw. E^{-1} formalisieren und seien R, V und T drei Relationen. Die konkreten Schemata sind direkt im jeweiligen Beweis angegeben. Sei ferner r_i die Tupel-ID des i-ten Quell-Tupels und w_j die Zeugenbasis des j-ten Ziel-Tupels. Seien I, J und I^* drei Instanzen über S_t bzw. S_{t+1}, wobei J und I^* wie folgt berechnet werden: $J = \textsc{Chase}_{\mathcal{M}^*}(I)$ und $I^* = \textsc{Chase}_{\mathcal{M}^*}(J) = \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}((I))$.

Klasse I: Provenance-Invariant In einigen Fällen ändert zusätzliche Provenance nichts am zugehörigen Inversentyp. Hierzu gehören insbesondere alle Operatoren, welche bereits ohne weitere Informationen eine exakte Chase-Inverse definieren. Dies betrifft etwa die Hälfte der klassifizierten Modifikationsoperatoren. Konkret gehören hierzu die Operatoren `COPY Table`, `CREATE Table`, `PARTITION Table` and `RENAME Table` sowie `ADD Column`, `COPY Column` und `RENAME Column`. Wir fassen diese sieben Operatoren zu einer gemeinsamen Klasse zusammen. Bezüglich der Chase-Inversen ergibt sich folgende Aussage (siehe Lemma 7.1):

Lemma 7.1. *Sei $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ eine Schemaabbildung der Klasse I. Dann ist die entsprechende inverse Funktion $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^*)$ immer eine exakte* Chase*-Inverse unabhängig von zusätzlicher Provenance.*

Wir werden die obige Behauptung o.B.d.A. für `COPY Table` beweisen. Der Beweis der anderen Operation erfolgt nach dem selben Prinzip und kann im Anhang C nachgelesen werden.

Beweis. `COPY Table` kann auf zwei verschiedene Arten formalisiert werden: als eine s-t tgd oder als Menge von zwei s-t tgds. Sei dazu `COPY Table` ein Evolutionsschritt von S_t nach S_{t+1}. Seien weiter $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ mit

$$\Sigma = \{R(a,b,c) \rightarrow R'(a,b,c) \wedge V(a,b,c)\} \quad , \quad \Sigma^{-1} = \{R'(a,b,c) \wedge V(a,b,c) \rightarrow R(a,b,c)\}$$

zwei Schemaabbildungen, welche `Copy Table` und seine inverse Abbildung formalisieren. O.B.d.A. seien R und V jeweils auf drei Attribute beschränkt. Ferner sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz mit verschiedenen drei-elementigen Tupeln. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned}
I^* &= \textsc{Chase}_{\mathcal{M}^{-1}}(\textsc{Chase}_{\mathcal{M}}(I)) \\
&= \textsc{Chase}_{\mathcal{M}^{-1}}(\textsc{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\
&= \textsc{Chase}_{\mathcal{M}^{-1}}(\{R'(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\
&= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\
&= I
\end{aligned}$$

Wegen $I^* = I$ kann ein Informationsverlust vermieden und eine exakte Chase-Inverse garantiert werden. Da wir bereits eine exakte Inverse haben, ist das Hinzufügen weiterer Provenance-Informationen im Sinne zusätzlicher Zeugenbasen oder Provenance-Polynome über den Quell-IDs/Provenance-IDs hier nicht notwendig.

Seien alternativ die Schemaabbildungen $\mathcal{M}$ und $\mathcal{M}^{-1}$ definiert über

$$\begin{aligned}
\Sigma &= \{R(a,b,c) \rightarrow R'(a,b,c), R(a,b,c) \rightarrow V(a,b,c)\}, \\
\Sigma^{-1} &= \{R'(a,b,c) \rightarrow R(a,b,c), V(a,b,c) \rightarrow R(a,b,c)\}.
\end{aligned}$$

Dann liefert die Anwendung des Chase&Backchase wiederum $I^* = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} = I$. Auch in diesem Fall kann ohne zusätzliche Provenance eine exakte Chase-Inverse angegeben werden. □

Zusätzlich zu `COPY Table` kann auch `COPY Column` auf zwei verschiedene Arten formalisiert werden. In beiden Fällen kann eine exakte Chase-Inverse angegeben werden. Für alle anderen SMOs der Klasse I gibt es nur eine Formalisierung, die eine exakte Chase-Inverse mit und ohne Verwendung zusätzlicher Provenance garantiert. Die entsprechenden Beweise sind im Anhang zu finden.

Korollar 7.1. *Sei $\mathcal{M}$ eine Schemaabbildung der Klasse I, d.h. $\mathcal{M}$ hat eine exakte* Chase*-Inverse. Dann liefert zusätzliche Provenance keine weiteren Informationen.* □

Klasse II: Dangling Tuples Je nach Quell-Instanz können Dangling Tuple auftreten. In diesem Fall kann eine exakte Chase-Inverse nicht garantiert werden. Auch zusätzliche Zeugenbasen und/oder Provenance-Polynome können nur wenig beeinflussen. Speichern wir jedoch die verlorenen Tupel in *Side Tables* (siehe Definition 6.10), ist die Angabe einer exakten Chase-Inversen weiterhin möglich, wie in Lemma 7.2 zusammengefasst. Die formalen Beweise zu `JOIN Table` und `MOVE Column` können ebenfalls im Anhang C nachgelesen werden.

Lemma 7.2. *Sei $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ eine Schemaabbildung der Klasse II. Dann ist die entsprechende* Chase*-Inverse $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ exakt, sofern die Anwendung von $\text{Chase}_{\mathcal{M}}$ keine Dangling Tuple liefert. Liegen Dangling Tuple vor, ist der* Chase*-Inversentyp ergebnisäquivalent oder relaxt. Durch die Verwendung zusätzlichen Provenance-Informationen wie Zeugenbasen und Side Tables können jedoch immer eine exakte* Chase*-Inverse angeben werden.*

Dangling Tuple treten insbesondere bei den Operatoren `JOIN Table` und `MOVE Column` auf. Schauen wir uns die Situation von `JOIN Table` am Beispiel etwas genauer. Sei also `JOIN Table` ein Evolutionsschritt von S_t nach S_{t+1}. Seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ mit

$$\Sigma = \{R(a,b) \wedge V(c,d) \wedge b = d \rightarrow T(a,b,c)\} \quad , \quad \Sigma^{-1} = \{T(a,b,c) \rightarrow R(a,b) \wedge V(c,d) \wedge b = d\}$$

zwei Schemaabbildungen, welche `JOIN Table` und seine inverse Funktion formalisieren. Seien weiter $R(\texttt{id}, \texttt{name})$, $V(\texttt{name}, \texttt{subject})$ und $T(\texttt{id}, \texttt{name}, \texttt{subject})$ drei Relationen und $I = \{R(1, \text{Alice}),\ R(2, \text{Bob}), V(\text{Alice}, \text{Math}), V(\text{Alice}, \text{IT})\}$ eine gegebenen Quell-Instanz. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(\{R(1,\text{Alice}), R(2,\text{Bob}), V(\text{Alice},\text{Math}), V(\text{Alice},\text{IT})\})) \\ &= \text{Chase}_{\mathcal{M}^*}(\{T(1,\text{Alice},\text{Math}), T(1,\text{Alice},\text{IT})\}) \\ &= \{R(1,\text{Alice}), V(\text{Alice},\text{Math}), V(\text{Alice},\text{IT})\} \\ &\preccurlyeq I \end{aligned}$$

Wie in der Abbildung 7.2 zu sehen, können Dangling Tuple (pink hervorgehoben) bei der Anwendung des Chase verloren gehen. Tupel wie $R(2, \text{Bob})$ können dann in der (minimalen) Teil-Datenbank I^* nicht mehr rekonstruiert werden. In diesem Fall rekonstruieren wir lediglich einen Ausschnitt der Quell-Instanz I (siehe Definition 6.2), was uns wiederum die Angabe einer ergebnisäquivalenten Chase-Inversen erlaubt. Sind hingegen keine Dangling Tuple vorhanden, kann weiterhin eine exakte Chase-Inverse garantiert werden.

Zusätzliche Provenance-Informationen wie Zeugenbasen oder Provenance-Polynome (orange hervorgehoben) ändern den Chase-Inversentyp nicht, da sie keine weiteren Informationen liefern. Speichern wir das Dangling Tuple $R(2, \text{Bob})$ jedoch zusätzlich in einer externen Side Table (gray hervorgehoben), kann es der Rekonstruktion von I^* wieder hinzugefügt werden. Zusammengefasst können wir so eine exakte Chase-Inverse garantieren. Denn:

$$I^* = \{R(1,\text{Alice}), S(\text{Alice},\text{Math}), V(\text{Alice},\text{IT})\} \cup \{R(2,\text{Bob})\} = I.$$

`MOVE Column` kann zusätzlich zu Dangling Tuple auch Duplikate enthalten. Die entsprechende SMO kann auf zwei verschiedene Arten formuliert werden. Dieser Operator ist der Einzige in unserer Studie, welcher nicht in jedem Fall eine Chase-Inverse hat. Der formale Beweis kann ebenfalls im Anhang nachgelesen werden.

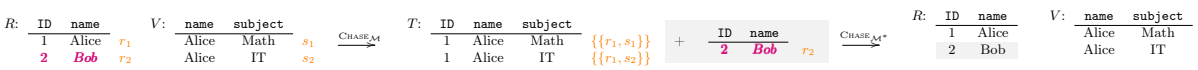
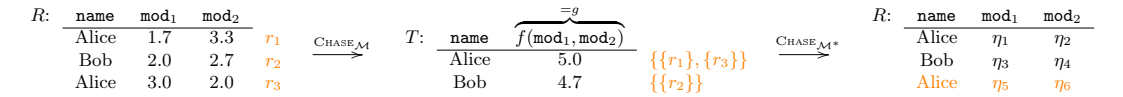
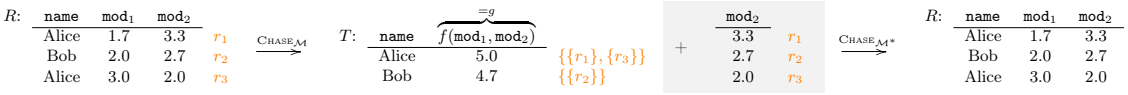


Abbildung 7.2. `JOIN Table` mit Dangling Tuples (pink hervorgehoben); erweitert um zusätzliche Provenance-Informationen (orange hervorgehoben) und Side Tables (grau hervorgehoben).

Klasse III: Duplikate Das Vorhandensein von Duplikaten erlaubt in der Regel nur die Angabe einer relaxten Chase-Inversen. Sie definieren daher eine eigene Klasse. SMOs, die zu dieser Klasse gehören, sind `DROP Column`, `MERGE Column`, `SPLIT Column` und `DECOMPOSE Table`. Der Inversentyp kann nur durch Provenance-Informationen wie Data Provenance und zusätzliche Annotationen verbessert werden. In einigen wenigen Fällen kann dann sogar eine exakte Chase-Inverse garantiert werden.



(a) Inverse ohne Wissen über f^{-1}



(b) Inverse mit Wissen über f^{-1}

Abbildung 7.3. `MERGE Column` mit verschiedenen Chase-Inversen; erweitert um zusätzliche Provenance-Informationen (orange hervorgehoben) und Side Tables (grau hervorgehoben).

Lemma 7.3. *Sei $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ eine Schemaabbildung der Klasse III. Dann ist die entsprechende* Chase-*Inverse $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ relaxt. Existieren keine Duplikate, so ist der Inversentyp sogar tp-relaxt. Zusätzliche Provenance-Informationen wie Zeugenbasen oder Provenance-Polynome sowie Side Tables ermöglichen zudem die Angabe einer tp-relaxten oder exakten* Chase-*Inverse.*

Beweis. Sei `MERGE Column` ein Evolutionsschritt von S_t zu S_{t+1}. Seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ mit

$$\Sigma = \{R(a,b,c) \to T(a, f(b,c))\} \quad , \quad \Sigma^{-1} = \{T(a,g) \to \exists D, E : R(a,D,E)\}$$

zwei Schemaabbildungen, welche den Operator `MERGE Column` und seine zugehörige inverse Funktion formalisieren. O.b.d.A. sei R auf drei und T auf zwei Attribute beschränkt. Sei ferner f eine beliebige Merging-Funktion und $I = \{R(x_{i_1}, x_{i_2}, x_{i_3})_{r_i} \mid i_1, i_2, i_3 \in \mathbb{N} \wedge f(x_{j_2}, x_{j_3}) = f(x_{k_2}, x_{k_3}) \wedge\ r_i \text{ Tupelidentifikator}\}$ eine beliebige Quell-Instanz. Sei weiter r_i Tupelidentifikator des i-ten Tupels und seien $R(x_{j_1}, x_{j_2}, x_{j_3})$ sowie $R(x_{k_1}, x_{k_2}, x_{k_3})$ zwei Tupel, welche nach Anwendung von f ein Duplikat liefern. Sei $w_j = \{\{r_j\}, \{r_k\}\}$ Zeugenbasis von r_j und $w_k = \{\{r_j\}, \{r_k\}\}$ Zeugenbasis von r_k. Ein Zeuge, hier $\{r_j\}$ und $\{r_k\}$, enthält alle Tupel-IDs, die für die Rekonstruktion eines Tupels benötigt werden. Zwei Zeugen stellen innerhalb einer Zeugenbasis ein Duplikat dar (orange hervorgehoben). Für diese Erweiterungen liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, f(x_{i_2}, x_{i_3}))_{w_i} \mid i \in \mathbb{N} \wedge\ \exists j,k : g_j = g_k \\ &\qquad \wedge\ w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\})\} \\ &= \{R(x_{i_1}, \eta_{i_2}, \eta_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge\ \eta_{j_2} = \eta_{k_2}, \eta_{j_3} = \eta_{k_3}\} \\ &\preccurlyeq_{(\text{tp})} I. \end{aligned}$$

Aufgrund der $\exists$-Quantoren erzeugt die inverse s-t tgd immer zwei neue Nullwerte η_{i_2} und η_{i_3}. Somit können die in f verarbeiteten Attributwerte nicht wiederhergestellt werden. Wegen des Duplikats enthält die rekonstruierte Instanz I^* jedoch weniger Tupel als die ursprüngliche Instanz I. Die Inverse $\mathcal{M}^*$ wird also ohne und tp-relaxed mit vorhandenen Duplikaten relaxt. Das Hinzufügen von Zeugenbasen garantiert eine tp-relaxierte Chase-Inverse. In diesem Fall gilt sowohl $\eta_{j_2} \neq \eta_{j_3}$ als auch $\eta_{k_2} \neq \eta_{k_3}$ und das verloren gegangene Duplikat kann rekonstruiert werden (siehe Abbildung 7.3a, orange hervorgehoben).

Seien alternativ $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ definiert über

$$\begin{aligned} \Sigma &= \{R(a,b,c) \to T(a, f(b,c)) \wedge S(c)\} \\ \Sigma^{-1} &= \{T(a,g) \wedge S(c) \to R(a, f^{-1}(g,c), c\}. \end{aligned}$$

Diese Formalisierung kann nur in Kombination mit der Provenance verwendet werden, da hier die Zeugenbasen w_i, Side Tables S_i und die Kenntnis von f^{-1} notwendig sind (orange hervorgehoben). So liefert $f^{-1}(g, c)$ den rekonstruierten Attributwert zu b — in Abbildung 7.3b berechnet durch $g - c$ und die Anwendung des CHASE&BACKCHASE:

$$
\begin{aligned}
I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\
&= \textsc{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, f(x_{i_2}, x_{i_3}))_{w_i} \mid i, i_1, i_2, i_3 \in \mathbb{N} \wedge\ \exists j, k : f(x_{j_2}, x_{j_3}) = f(x_{k_2}, x_{k_3}) \\
&\qquad \wedge\ w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\} \wedge\ f \text{ invertierbare Funktion}\} \\
&\qquad \cup\ \{(x_{i_3})_{r_i} \mid i, i_3 \in \mathbb{N} \wedge\ r_i \text{ Tupelidentifikator}\}) \\
&= \{R(x_{i_1}, f^{-1}(f(x_{i_2}, x_{i_3}), x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge\ f(x_{j_2}, x_{j_3}) = f(x_{k_2}, x_{k_3})\} \\
&= \{R(x_{i_1}, x_{x_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\
&= I.
\end{aligned}
$$

Sei w_i Zeugenbasis und x_{i_3} Element der Side Table $\{(x_{i_3})_{r_i} \mid i, i_3 \in \mathbb{N} \wedge\ r_i$ Tupelidentifikator$\}$. Dann kann x_{i_2} aus $f^{-1}(g_i, x_{i_3})$ wie folgt berechnet werden:

- die Attributwerte g_i und x_{i_1} sind in $J = \textsc{Chase}_{\mathcal{M}}(I)$ explizit enthalten;
- die Umkehrfunktion f^{-1} ist per Definition gegeben;
- der Attributwert x_{i_3} kann mit Hilfe von w_i aus der Side Table ausgelesen werden.

Auch Duplikate können mit Hilfe dieser zusätzlichen Informationen rekonstruiert werden. Insgesamt kann eine exakte CHASE-Inverse garantiert werden. □

Abbildung 7.3 zeigt die obige Situation an einem konkreten Beispiel. Sei R die Tabelle, in welcher die Ergebnisse zweier Modulprüfungen zusammengefasst werden sollen. Kennen wir die Inverse von f^{-1} nicht, ist die zugehörige CHASE-Inverse relaxt (siehe Abbildung 7.3a). Das Duplikat $T(\text{Alice}, 5.0)$ in T kann jedoch mit Hilfe zusätzlicher Provenance-Informationen wie etwa einer Zeugenbasis w_i (orange hervorgehoben), rekonstruiert werden. Eine zusätzliche Side Table erlaubt sogar die Definition einer exakten CHASE-Inverse, sofern f^{-1} bekannt ist (siehe Abbildung 7.3b). Die Beweise der anderen SMOs können ähnlich geführt werden. Sie können im Anhang nachgelesen werden.

Klasse IV: Wiederhergestellt durch Provenance Die Anwendung des CHASE&BACKCHASE auf die Operatoren der ersten Klasse (Klasse I) liefert stets eine exakte CHASE-Inverse. Einen Informationsverlust haben wir hier nicht zu verzeichnen. Anders verhält es sich bei den übrigen Operatoren. Spezielle Probleme wie Dangling Tuple oder Duplikate werden in den Klassen II und III behandelt. Klasse IV enthält daher alle übrigen Operatoren, welche einen Informationsverlust aufweisen. Hierzu gehören insbesondere `DROP Table` (siehe Lemma 7.4) und `MERGE Table` (siehe Lemma 7.5).

Lemma 7.4. *Der Operator* ***`MERGE Table`*** *hat eine ergebnisäquivalente* CHASE*-Inverse. Zusätzliche Provenance-Informationen ermöglichen alternativ die Angabe einer exakten* CHASE*-Inversen.*

Sei `MERGE Table` ein Evolutionsschritt von S_t nach S_{t+1}. Die Vereinigung zweier Relationen R und V kann dann als Schemaabbildung $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ mit

$$\Sigma = \{R(a, b, c) \to T(a, b, c), V(a, b, c) \to T(a, b, c)\}$$

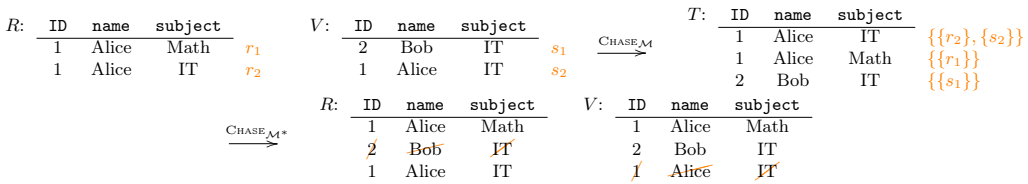


Abbildung 7.4. `MERGE Table` erweitert um zusätzliche Provenance-Informationen (orange hervorgehoben) mit deren Hilfe die falsch rekonstruierten Tupel gelöscht werden können (rot durchgestrichen).

formalisiert werden. Sei weiter $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ mit

$$\Sigma^{-1} = \{T(a,b,c) \rightarrow R(a,b,c), T(a,b,c) \rightarrow V(a,b,c)\}$$

die Inverse zu $\mathcal{M}$. Seien also R, S und T drei Relationen mit einer beliebig gewählten Quell-Instanz I. Dann liefert die Anwendung des Chase&Backchase:

$$I^* \quad = \quad \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) = \textsc{Chase}_{\mathcal{M}^*}(J) \neq I,$$

wobei I^* mehr Tupel als I enthalten kann. In Abbildung 7.4 entspräche dies beispielsweise den Tupeln $R(2, \text{Bob}, \text{IT})$ oder $V(1, \text{Alice}, \text{IT})$, welche ursprünglich nicht in den Quell-Relationen R bzw. T enthalten waren.

Die in I^* falsch rekonstruierten Tupel können jedoch mit Hilfe zusätzlicher Provenance-Informationen (siehe Abbildung 7.4, orange hervorgehoben) identifiziert und gelöscht werden (rot durchgestrichen). Wenn wir wissen, wie die Tupel aus R und V an der Berechnung von J beteiligt sind, können wir diese bei der Berechnung von I^* wieder der richtigen Quell-Relationen zuordnen. So können wir die Quell-Instanz fehlerfrei rekonstruieren und eine exakte Chase-Inverse garantieren. Konkret genügt es zu wissen, welche Tupel-IDs an der Erzeugung von $R(2, \text{Bob}, \text{IT})$ oder $V(1, \text{Alice}, \text{IT})$ beteiligt sind. Eine verallgemeinerte Zeugenbasis oder eine Zeugenliste sind an dieser Stelle daher ausreichend.

Lemma 7.5. *Der Operator* `DROP Table` *hat eine relaxte* Chase*-Inverse. Durch das Hinzufügen weiterer Provenance-Informationen kann der Inversentyp zusätzlich als tp-relaxed definiert werden.*

`DROP Table` und `MERGE Table` können analog behandelt werden. Ein formaler Beweis von Lemma 7.4 und Lemma 7.5 findet sich im Anhang C.

Wir beenden die Untersuchungen von Provenance und Evolution abschließend mit der Definition der ***what***-Provenance (siehe Abschnitt 7.4). Anschließend werden wir in Kapitel 8 den natürlichen Konflikt zwischen der Speicherung und Veröffentlichung von Daten im Sinne der *Provenance* mit dem Schutz dieser Daten im Sinne der *Privacy* untersuchen.

SMO	Without Provenance		With Provenance		
	Σ	Type	Σ^{-1}	Type	Provenance
`COPY Table`	$R(a,b,c) \to R'(a,b,c) \wedge V(a,b,c)$	$=$	$R'(a,b,c) \wedge V(a,b,c) \to R(a,b,c)$	$=$	
	$R(a,b,c) \to R'(a,b,c)$ $R(a,b,c) \to V(a,b,c)$	$=$	$R'(a,b,c) \to R(a,b,c)$ $V(a,b,c) \to R(a,b,c)$	$=$	
`CREATE Table`	$\emptyset \to \exists A,B,C : R(A,B,C)$	$=$	$R(a,b,c) \to \emptyset$	$=$	
`DECOMPOSE Table`	$R(a,b,c) \to T(a,b) \wedge V(b,c)$	$=$	$T(a,b) \wedge V(b,c) \to R(a,b,c)$	$=$	
		$\leftrightarrow$	$T(a,b) \wedge V(b,c) \to R(a,b,c)$	$=$	***why, how***
	$R(a,b,c) \to T(a,b)$, $R(a,b,c) \to V(b,c)$	$\preccurlyeq_{\text{tp}}$	$T(a,b) \to \exists C : R(a,b,C)$, $V(b,c) \to \exists A : R(A,b,c)$	$\preccurlyeq_{\text{tp}}$	
		$\succcurlyeq$	$T(a,b) \to \exists C : R(a,b,C)$, $V(b,c) \to \exists A : R(A,b,c)$	$\preccurlyeq_{\text{tp}}$ + reconstruction of lost tuples (duplicates)	***why, how***
`DROP Table`	$R(a,b,c) \to \emptyset$	$\preccurlyeq$	$\emptyset \to \exists A,B,C : R(A,B,C)$	$\preccurlyeq_{\text{tp}}$ + reconstruction of lost null tuples	***where, why, how***
`JOIN Table`	$R(a,b) \wedge V(a,c) \to T(a,b,c)$	$=$	$T(a,b,c) \to R(a,b) \wedge V(a,c)$	$=$	
		$\succcurlyeq$	$T(a,b,c) \to R(a,b) \wedge V(a,c)$	$\preccurlyeq$	***why, how***
				$=$ + reconstruction of lost tuples (dangling tuple)	***why, how*** + side-tables
`MERGE Table`	$R(a,b,c) \to T(a,b,c)$, $V(a,b,c) \to T(a,b,c)$	$\leftrightarrow$	$T(a,b,c) \to R(a,b,c)$, $T(a,b,c) \to V(a,b,c)$	$=$ + restriction to tuples of R resp. V	***where, why, how***
`PARTITION Table` / `DISTRIBUTE Table`	$R(a,b,c) \wedge \text{cond}_A \to V(a,b,c)$, $R(a,b,c) \wedge \neg\text{cond}_A \to T(a,b,c)$	$=$	$V(a,b,c) \to R(a,b,c)$, $T(a,b,c) \to R(a,b,c)$	$=$	
`RENAME Table`	$R(a,b,c) \to V(a,b,c)$	$=$	$V(a,b,c) \to R(a,b,c)$	$=$	

Tabelle 7.6. SMOs mit exakten ($=$), tp-relaxten ($\preccurlyeq_{\text{tp}}$), relaxten ($\preccurlyeq$) und ergebnisäquivalenten ($\leftrightarrow$) Chase-Inversen erweitert um zusätzliche Provenance-Informationen (orange hervorgehoben); Dangling Tuples (pink hervorgehoben), Duplikate (bau hervorgehoben) sowie falsch rekonstruierten Tupel (grün hervorgehoben).

SMO	Without Provenance			With Provenance		
	Σ	Type	Σ^{-1}	Type	Σ^{-1}	Provenance
ADD Column	$R(a,b,c) \to V(a,b,c,\mathsf{const} \mid f(a,b,c))$	$=$	$V(a,b,c,\mathsf{const} \mid f(a,b,c)) \to R(a,b,c)$	$=$	$V(a,b,c,\mathsf{const} \mid f(a,b,c)) \to R(a,b,c)$	
	$R(a,b,c) \to \exists D : V(a,b,c,D)$	$=$	$V(a,b,c,d) \to R(a,b,c)$	$=$	$V(a,b,c,d) \to R(a,b,c)$	
COPY Column	$R(a,b,c) \wedge V(d,e) \wedge \mathsf{cond}_A \to R'(a,b,c) \wedge T(c,d,e)$ $R(a,b,c) \wedge V(d,e) \wedge \neg\mathsf{cond}_A \to \exists C : R'(a,b,c) \wedge T(C,d,e)$	$=$	$R'(a,b,c) \to R(a,b,c)$ $T(c,d,e) \to V(d,e)$	$=$	$R'(a,b,c) \to R(a,b,c)$ $T(c,d,e) \to V(d,e)$	
	$R(a,b,c) \to T(a,b,c,c)$	$=$	$T(a,b,c,c) \to R(a,b,c)$	$=$	$T(a,b,c,c) \to R(a,b,c)$	
DROP Column	$R(a,b,c,d) \to T(a,b,c)$	$\preccurlyeq_{\text{tp}}$	$T(a,b,c) \to \exists D : R(a,b,c,D)$	$\preccurlyeq_{\text{tp}}$	$T(a,b,c) \to \exists D : R(a,b,c,D)$	
		$\preccurlyeq$	$T(a,b,c) \to \exists D : R(a,b,c,D)$	$\preccurlyeq_{\text{tp}}$	$T(a,b,c) \to \exists D : R(a,b,c,D)$ + reconstruction of lost tuples (duplicates)	***why, how***
MERGE Column	$R(a,b,c) \to T(a,f(b,c)) \wedge S(c)$	$\preccurlyeq_{\text{tp}}$	$T(a,g) \to \exists D,E : R(a,D,E)$	$\preccurlyeq_{\text{tp}}$	$T(a,g) \to \exists D,E : R(a,D,E)$	
		$\preccurlyeq$	$T(a,g) \to \exists D,E : R(a,D,E)$	$\preccurlyeq_{\text{tp}}$	$T(a,g) \to \exists D,E : R(a,D,E)$ + reconstruction of lost tuples (duplicates)	***why, how***
				$=$	$T(a,g) \wedge S(c) \to R(a,f^{-1}(g,c),c)$ + reconstruction of lost attribute values	***how*** + inverse of f + side tables
				$=$	$T(a,g) \wedge S(c) \to R(a,f^{-1}(g,c),c)$ + reconstruction of lost tuples (duplicates) + reconstruction of lost attribute values	***how*** + inverse of f + side tables
MOVE Column	$R(a,b,c) \wedge V(d,e) \wedge \mathsf{cond}_A \to R'(a,b) \wedge T(c,d,e) \wedge S(a,b,c)$	$=$	$R'(a,b) \wedge T(c,d,e) \wedge \mathsf{cond}_A \to R(a,b,c) \wedge V(d,e)$	$=$	$R'(a,b) \wedge T(c,d,e) \wedge \mathsf{cond}_A \to R(a,b,c) \wedge V(d,e)$	
		$\preccurlyeq/\preccurlyeq$	$R'(a,b) \wedge T(c,d,e) \wedge \mathsf{cond}_A \to R(a,b,c) \wedge V(d,e)$	$\preccurlyeq$	$R'(a,b) \wedge T(c,d,e) \wedge \mathsf{cond}_A \to R(a,b,c) \wedge V(d,e)$	***why, how***
				$=$	$R'(a,b) \wedge T(c,d,e) \wedge \mathsf{cond}_A \to R(a,b,c) \wedge V(d,e)$ $S(a,b,c) \to R(a,b,c)$ + reconstruction of lost tuples	***why, how*** + side tables
	$R(a,b,c) \wedge V(d,e) \wedge \mathsf{cond}_A \to R'(a,b) \wedge T(c,d,e)$, $R(a,b,c) \wedge V(d,e) \wedge \neg\mathsf{cond}_A \to \exists C : R'(a,b) \wedge T(C,d,e)$	$=$	$R'(a,b) \wedge T(c,d,e) \to R(a,b,c)$ $T(c,d,e) \to V(d,e)$	$=$	$R'(a,b) \wedge T(c,d,e) \to R(a,b,c)$ $T(c,d,e) \to V(d,e)$	
		$\nleftrightarrow/\nleftrightarrow$	$R'(a,b) \wedge T(c,d,e) \to R(a,b,c)$ $T(c,d,e) \to V(d,e)$	$=$	$R'(a,b) \wedge T(c,d,e) \to R(a,b,c)$ $T(c,d,e) \to V(d,e)$ + restriction to tuples of R resp. V	***why, how***
RENAME Column	$R(a,b,c) \to V(a,b,c)$	$=$	$V(a,b,c) \to R(a,b,c)$	$=$	$V(a,b,c) \to R(a,b,c)$	
SPLIT Column	$R(a,b) \to T(f_1(a),b,f_2(a))$	$\preccurlyeq_{\text{tp}}$	$T(g_1,b,g_2) \to \exists C : R(C,b)$	$\preccurlyeq_{\text{tp}}$	$T(g_1,b,g_2) \to \exists C : R(C,b)$	
		$\preccurlyeq$	$T(g_1,b,g_2) \to \exists C : R(C,b)$	$\preccurlyeq_{\text{tp}}$	$T(f_1(a),b,f_2(a)) \to \exists C : R(C,b)$ + reconstruction of lost tuples (duplicates)	***why, how***
				$=$	$T(g_1,b,g_2) \to R(f^{-1}(g_1,g_2),b)$ + reconstruction of lost attribute values	***why, how*** + inverses of f_1, f_2
				$=$	$T(g_1,b,g_2) \to R(f^{-1}(g_1,g_2),b)$ + reconstruction of lost tuples (duplicates) + reconstruction of lost attribute values	***why, how*** + inverses of f_1, f_2

Tabelle 7.7. SMOs mit exakten ($=$), tp-relaxten ($\preccurlyeq_{\text{tp}}$), relaxten ($\preccurlyeq$) und ergebnisäquivalenten ($\leftrightarrow$) Chase-Inversen erweitert um zusätzliche Provenance-Informationen (orange hervorgehoben); Dangling Tuples (pink hervorgehoben) und Duplikate (blau hervorgehoben).

7.4. *what*-Provenance *(basierend auf [AH21])*

Data Provenance beschreibt den Weg eines (Anfrage-)Ergebnisses zurück zu seinem (physischen) Ursprung. Wir erhalten somit eine Dokumentation, woher ein Datenstück stammt (***where***), warum es im Ergebnis enthalten ist (***why***) und wie es genau erstellt wurde (***how***). Bei der Kombination von Data Provenance und Evolution stellt sich uns jedoch eine weitere Frage, welche mit den bisherigen Provenance-Fragen nicht bearbeitet werden kann. So fehlen uns Informationen über die zulässigen Datentypen sowie ihre zugehörigen Formate [Man20], um auch Änderungen auf Ebene der Datentypen rekonstruieren zu können.

Ziel der ***what***-Provenance ist es, Datenunreinheiten vorzubeugen, welche durch fehlerhafte Rundungsfehler, ungenau definierte oder implementierte inverse Funktionen sowie den Verlust versteckter Informationen verloren gehen könnten. So enthalten die Daten des IOWs beispielsweise versteckte Informationen in Form einer führenden 1 oder 0 zur Unterscheidung von validierten und unvalidierten Daten. Dies ist notwendig, da lediglich die validierten Daten ausgewertet werden, die originalen Daten aber für spätere Analysen noch gespeichert werden müssen. Je nach Datentyp kann es jedoch passieren, dass die führende 0 nach der Evolution und der daraus folgenden Änderung des Datentyps weggeschnitten wird. Die anschließende Invertierung der Evolution kann diesen Verlust nicht ohne Weiteres rekonstruieren.

Auch Rundungsfehler können so vermieden werden. Im Falle des Sauerstoffgehalts variieren die Werte zwischen 7 und 9, so dass selbst kleine Rundungsfehler hier einen großen Effekt haben. Die „bedenkenlose“ Verwendung vordefinierte SQL-Funktionen wie `CONCAT` oder `SUBSTRING` — notwendig für unsere oben eingeführten `MERGE Column` und `SPLIT Column`-Varianten — ist ebenfalls nur dann möglich, wenn der zugrundeliegende Datentyp klar definiert ist. Auch die Tatsache, dass die mathematische Inverse einer Funktion nicht notwendigerweise mit der implementierten Inverse übereinstimmen muss, erklärt die Notwendigkeit einer solchen ***what***-Provenance (siehe Definition 7.1).

Definition 7.1 (***what***-Provenance)**.** Seien $S_t(A_1, ..., A_n)$ und $S_{t+1}(A'_1, ..., A'_m)$ zwei temporale Versionen einer Datenbank. Zusätzlich sei E eine Evolution zwischen S_t und S_{t+1}. Die ***what***-*Basis* ist definiert als Menge von Tupeln der Form $(A'_i, (A_k, \mathbb{D}(A_k)) \times ... \times (A_l, (A_l, \mathbb{D}(A_l)))$ mit $E^{-1}(A'_i) = (A_k, ..., A_l)^T$ und $\mathbb{D}(A_i)$ als Domäne der Attribute A_i. Wenn A kein Urbild hat, schreiben wir $(A, (\emptyset, \emptyset))$. ■

Beispiel 7.3. Gegeben seien zwei Versionen der Relation `Series` aus den Jahren 1977 und 2017, dargestellt in Tabelle 7.1, sowie die Evolutionen E, E^{-1} und die Anfrage Q mit

$$
\begin{aligned}
Q &= \pi_{\text{day},O_2}(\texttt{Series}_{1977}) \\
E &= \texttt{MERGE Column date AS FUNC}_1\texttt{(year,month,day) INTO Series}_{1977}; \\
&\quad \texttt{ADD Column R\_}O_2 \texttt{ AS FUNC}_2(O_2) \texttt{ INTO Series}_{1977}; \\
&\quad \texttt{DROP Column } O_2 \texttt{ FROM Series}_{1977} \\
\texttt{FUNC}_1\texttt{(year,month,day)} &:= \texttt{CONCAT(year,'-',month,'-',day)} \\
\texttt{FUNC}_2(O_2) &:= \texttt{ROUND}(O_2,1) \\
E^{-1} &= \texttt{CREATE VIEW Series}_{1977} \texttt{ AS} \\
&\quad \texttt{SELECT} * \texttt{FROM Series WHERE date} < \text{1998-01-01}; \\
&\quad \texttt{SPLIT Column date IN Series}_{1977} \texttt{ TO year USING FUNC}_3\texttt{(date),} \\
&\quad \texttt{month USING FUNC}_4\texttt{(date), day USING FUNC}_5\texttt{(date)} \\
\texttt{FUNC}_3\texttt{(date)} &:= \texttt{SUBSTRING}(\texttt{date}, 1, 4) \\
\texttt{FUNC}_4\texttt{(date)} &:= \texttt{SUBSTRING}(\texttt{date}, 6, 2) \\
\texttt{FUNC}_5\texttt{(date)} &:= \texttt{SUBSTRING}(\texttt{date}, 9, 2).
\end{aligned}
$$

Die Probleme, die bei unserem Beispiel auftreten können, entstehen bei der Anwendung der fünf Funktionen $\texttt{FUNC}_1$ bis $\texttt{FUNC}_5$. Nehmen wir hierzu an, dass alle Attribute der Relation $\texttt{Relation}_{1977}$ vom Typ `INTEGER` sind. So wird im ersten Schritt der Sauerstoffgehalt gerundet und das Datum auf drei Attribute aufgeteilt. Hierbei gehen sowohl

die Nachkommastellen bei Q_2 als auch die führende Null im Monat verloren. Beides kann bei der abschließenden Evolution nicht mehr rekonstruiert werden.

Merken wir uns in der ***what***-Provenance jedoch den ursprünglichen Datentyp, so können wir bei der abschließenden Evolution die führende Null im Monat wieder ergänzen und die fehlenden Nachkommastellen mit Nullen auffüllen. Konflikte im Datentyp können so vermieden und durch die inverse Evolution hervorgerufene Datenfehler (im Ansatz) korrigiert werden.

Die zugehörige ***what***-Provenance der Evolution lautet:

$$\{(\texttt{ID}, (\texttt{ID}, \texttt{INTEGER})), (O_2, (O_2, \texttt{DOUBLE})),$$
$$(\texttt{year}, (\texttt{date}, \texttt{DATE})), (\texttt{month}, (\texttt{date}, \texttt{DATE})), (\texttt{day}, (\texttt{date}, \texttt{DATE}))\}$$

und die ***what***-Provenance der inversen Evolution:

$$\{(\texttt{ID}, (\texttt{ID}, \texttt{INTEGER})), (O_2, (O_2, \texttt{DOUBLE})),$$
$$(\texttt{year}, (\texttt{year}, \texttt{CHAR}(2)) \times (\texttt{month}, \texttt{CHAR}(2)) \times (\texttt{day}, \texttt{CHAR}(2)))\}.$$

□

7.5. Zwischenfazit

Die Rekonstruktion von (veröffentlichten) Evaluationsergebnissen $I(Q)$ für sich entwickelnde Datenbanken ist ebenso ein Garant für ein gutes Forschungsdatenmanagement wie die langfristige Bereitstellung der Daten selbst, welche wir auf einen minimalen Teil I^* zum Zeitpunkt t bzw. J^* zum Zeitpunkt $t+1$ beschränken wollen. Die Komposition der Auswertungsanfrage Q und der Evolutionsanfrage E sowie ihrer Inversen Q^{-1} und E^{-1} ermöglicht es uns, die minimal notwendige Teil-Datenbank der Original-Datenbanken I und J zu jedem beliebigen Zeitpunkt t bzw. $t+1$ zu bestimmen (siehe Abschnitt 7.1). Für die Rekonstruktion des Evaluationsergebnisses $I(Q)$ benötigen wir also eine (minimale) Teil-Datenbank I^* zum Zeitpunkt t oder J^* zum Zeitpunkt $t+1$ sowie die Evolutionsanfrage E und die Anfrage Q. Dabei gilt, $J^* = E(I^*)$.

Beide Anfragen, d.h. die Auswertungsanfrage Q sowie die Evolutionsanfrage E, werden als Schemaabbildung (siehe Definition 3.18 und 3.19) interpretiert und mit dem Chase&Backchase verarbeitet. Dieser übernimmt sowohl die Verarbeitung der Evaluations- und der Evolutionsanfrage als auch deren Invertierungen. Sind Q und E bekannt, können wir durch Bestimmen des Chase-Inversentyps bereits vor Ausführung der Anfragen den zu erwarteten Informationsverlust angeben. Die Chase-Inversentypen der Auswertungsoperatoren sind in Abschnitt 6.3 und die der Schemaevolutionsoperatoren in Abschnitt 7.3 zusammengefasst.

Die Berechnung der inversen Funktionen sowie die Bestimmung des Inversentyps übernimmt in der praktischen Implementierung ProSA selbst (siehe Kapitel 10). Die Ausführung des Chase wird hingegen an ChaTEAU ausgelagert (siehe Kapitel 9).

Eigene sowie betreute Arbeiten

Die in diesem Kapitel vorgestellten Ergebnisse basieren hauptsächlich auf den folgenden Veröffentlichungen und betreuten studentischen Arbeiten:

- [AH18a] **Tanja Auge**, Andreas Heuer: Combining Provenance Management and Schema Evolution. *ProvenanceWeek*, 2018
- [Yan18] Jiawei Yan: Konzeption und Umsetzung einer Schemaextraktion von langjährigen In-Situ-Forschungsdaten aus der Ostsee. *Masterarbeit*, 2018
- [Man20] Erik Manthey: Beschreibung der Veränderungen von Schemata und Daten am IOW mit Schema-Evolutions-Operatoren. *Bachelorarbeit*, 2020

- [Aug+20] **Tanja Auge**, Erik Manthey, Susanne Jürgensmann, Susanne Feistel, Andreas Heuer: Schema Evolution and Reproducibility of Long-term Hydrographic Data Sets at the IOW. *LWDA*, 2020
- [AH22a] **Tanja Auge**, Andreas Heuer: Enhanced Inversion of Schema Evolution with Provenance. (Submitted for Pulbication)

Die Klassifikation der relevanten Evolutionsoperatoren aus Abschnitt 7.2 basiert auf [Man20], wurde in [Aug+20] aber noch einmal grundlegend überarbeitet. Dies gilt insbesondere für die Anzahl der Schemaänderungen in den vom IOW bereitgestellten Daten (siehe Tabelle 7.3) sowie der Definition der ***what***-Provenance (siehe Definition 7.1). Die Berechnung der (minimalen) Teil-Datenbank bei sich ändernden Datenbanken aus Abschnitt 7.1 basiert auf [AH21] und die Klassifizierung der SMOs nach ihren Chase-Inversen sowie die Bestimmung der Chase-Inversentypen selbst auf [AH22a].

8. Provenance und Privacy

Wie bereits beschrieben existiert ein natürlicher Konflikt zwischen der Speicherung und Veröffentlichung von (Original-)Daten (*Provenance*) sowie dem Schutz dieser Daten (*Privacy*). Dass dieses Problem noch nicht gelöst ist, zeigt eine kurze Literaturanalyse. Des Weiteren untersuchen wir den Kenntnisstand verschiedener (Nicht-) Wissenschaftler zu diesem Thema in einer 2019 durchgeführten Interview-Studie (siehe Abschnitt 8.1). Anschließend untersuchen wir eine Anonymisierung der ***where***-, ***why***- und ***how***-Provenance (siehe Abschnitt 8.2) und stellen zwei mögliche Anonymisierungsmethoden vor (siehe Abschnitt 8.3).

Provenance und Privacy in der Literatur Die Konzepte von Provenance und Privacy werden seit langem unabhängig voneinander untersucht. Eine Kombination ist in der Literatur jedoch selten zu finden. Nennenswert sind unter anderem die Arbeiten von Braun et al. [Bra+06], Cheney [Che11] sowie Davidson et al. [Dav+11]. Andere Arbeiten wie [AL12; SBS18; Ber+14] und [CY20] beschreiben jeweils eigene Provenance-Modelle zur Erhöhung bestimmter Privacy-Anforderungen. Da diese in ProSA jedoch keine Anwendung finden, sollen sie hier aber nicht weiter vertieft werden.

So beschäftigt sich [Bra+06] beispielsweise mit den Problemen, welche bei der automatisierten Sammlung von Provenance-Informationen entstehen können. Hierzu gehören etwa das Aufzeichnen von Provenance-Informationen über die Benutzer von Services wie etwa das Speicherung von Besuchszeiten, Absendern oder konkreten Inhalten. Dies sowie die mangelnde Restriktion gesammelter Informationen kann zu maßgeblichen Verstößen gegen die DSGVO führen. Auch die Autoren von [Tor+17] begründen die Relevanz zur Kombination von Provenance und Privacy mit dem „Recht auf Vergessenwerden“ sowie der Europäischen Datenschutzgrundverordnung. Bzgl. Provenance wird insbesondere das Löschen von Datensätzen als problematisch beschrieben. Grund hierfür ist, dass die Löschung einzelner (Quell-)Datensätze ein ebenfalls in der Datenbank gespeichertes aggregiertes Datum ungültig machen kann, sofern das gelöschte Datum an der Aggregation beteiligt war. Ein Problem, welches in ProSA im Kontext des Forschungsdatenmanagements jedoch nicht auftreten wird.

Zur Behandlung bzw. Vermeidung der in [Bra+06] beschriebenen Probleme definiert Cheney in [Che11] einige Policies zur Veröffentlichung von Provenance unter Privacy-Aspekten. Beide Arbeiten beziehen sich dabei insbesondere auf den Anwendungsbereich der Information System Provenance.

[Dav+11] spezialisiert sich auf die Einhaltung der Privacy-Aspekte *Data*, *Module* und *Structural Privacy* im Kontext von Workflow-Provenance. Hierbei geht es um den Schutz der Daten innerhalb eines Workflows. Zudem muss die Ausgabe aus der Eingabe einwandfrei ableitbar sein und Provenance-Informationen dürfen für Unbefugte nicht einsehbar sein, sodass von Außen keine Rückschlüsse auf die Struktur des Workflows gezogen werden können. Wir werden uns im Weiteren jedoch auf den Aspekt der Data Provenance beschränken

8.1. Interview-Studie zum Verständnis der Begriffe Provenance, Privacy und Forschungsdatenmanagement *(basierend auf [ASH21b])*

Beide Konzepte, d.h. Provenance und Privacy, bilden die Grundlage für gute Forschung. So verlangen Konferenzen und Journale im Zuge nachhaltiger Forschung immer häufiger die Reproduzierbarkeit bzw. Replikation neuer Forschungsergebnisse. Sollte dies nicht möglich sein, so wird zumindest ein Plausibilitätsnachweis des veröffentlichten Ergebnisses gewünscht. Viele Wissenschaftler publizieren bereits Open Access und stellen ihre Daten sowie ihren Code öffentlich oder auf Anfrage zur Verfügung. Insbesondere im Falle personenbezogener Daten oder der Involvierung von Projektpartnern mit wirtschaftlichen oder politischen Interessen, kann eine Bereitstellung von Code

und/oder Daten zum Problem werden. So kann die Rückverfolgung von Daten bis zu ihrem Ursprung beispielsweise mit dem Schutz (personenbezogener) Daten und damit dem Datenschutz kollidieren. Dies führt zu einem natürlichen Konflikt zwischen den Konzepten *Provenance* und *Privacy*.

In Gesprächen mit Kollegen verschiedener Fachbereiche ist uns zudem immer wieder aufgefallen, dass die Begriffe Provenance und Privacy nicht jedem geläufig oder je nach Anwendungsgebiet unterschiedlich definiert sind. Bevor wir uns der Kombination von Provenance und Privacy widmen können, benötigen wir daher zunächst eine einheitliche Definition. Um möglichst individuelle Antworten zu erhalten, haben wir uns für die Durchführung einer Interview-Studie entschieden. Da diese in einer im Rahmen der Dissertation betreuten Bachelorarbeit entstanden ist, stammen alle Probanden aus dem Umfeld der Universität Rostock.

Vorbereitungen

Insgesamt haben wir 21 Personen zu den Themen (i) *persönliche Situation*, (ii) *Provenance und Privacy* und (iii) *Forschungsdatenmanagement* befragt. Das Ziel der Umfrage bestand darin herauszufinden, was die Teilnehmer über die Konzepte von Provenance und Privacy wissen und welche Anforderungen sie an ein gutes Forschungsdatenmanagement stellen. Dabei interessierte uns unter anderem, welche Daten für die Rekonstruktion der Forschungsergebnisse relevant sind und daher langfristig gespeichert werden müssen. Wir gehen jedoch davon aus, dass es nicht immer im Interesse der Wissenschafter und Datenverarbeiter ist, ihre Daten vollständig und so früh wie möglich zu veröffentlichen. So haben einige Wissenschaftler ein ausschließliches Nutzungsrecht für die in einem Projekt entstandenen Daten. Da dieses Vorzugsrecht oft nur für eine bestimmte Zeit gilt, ist eine verfrühte Veröffentlichung der Daten in diesem Fall nicht gewünscht. Auch Privacy-Aspekte können ein Grund für die Nicht-Veröffentlichung von Daten sein. Der Begriff Privacy bezieht sich hierbei nicht auf die Privatsphäre einzelner Personen, sondern auf den Schutz von (Forschungs-)Daten als solche. Letzteres ist vor allem im Kontext von wissenschaftlichen Förderorganisationen wie der DFG interessant[1], welche bestimmte Anforderungen an Projekte und Publikationen im Sinne der wissenschaftlichen Reproduzierbarkeit stellen.

Erhebungsmethode und Stichprobe Da die persönlichen Definitionen und Interpretationen der Teilnehmer hier im Vordergrund stehen, haben wir uns für ein Experteninterview entschieden. Eine quantitative Auswertung ist zweitrangig und dient hier nur der Visualisierung. Die Interviews wurden im Rahmen der Bachelorarbeit [Sch20] erhoben und sind in [ASH21b] erneut ausgewertet worden. Wir beziehen uns hier insbesondere auf die Ergebnisse aus [ASH21b].

Unsere Stichprobe besteht aus 21 Mitgliedern unseres beruflichen Netzwerkes am Standort Rostock. Die Auswahl der Probanden ist damit nicht zwangsläufig repräsentativ. Die Ergebnisse spiegeln dennoch meine persönlichen Erfahrungen wieder. Wir haben zudem darauf geachtet, die drei Zugehörigkeiten *Wissenschaftler der Universität Rostock*, *Nicht-Wissenschaftler* und *Wissenschaftler aus An-Instituten* gleichmäßig ausgewählt zu haben (siehe Abbildung 8.1a). Gleiches gilt für die Forschungsgebiete bzw. Abteilungen, dargestellt in Abbildung 8.1b. An unserer Studie haben insgesamt 16 Männer und fünf Frauen im Alter von Anfang 20 bis Ende 50 Jahren teilgenommen. Die Gruppe der Probanden setzt sich insgesamt aus vier Studierenden, fünf Doktoranden, zwei Professoren, sieben Nicht-Wissenschaftlern und drei wissenschaftlichen Mitarbeitern zusammen.

Durchführung und Auswertung Insgesamt haben wir 20 Interviews (19 Einzel- und ein Doppel-Interview) à etwa 20 Minuten durchgeführt. Die entstandenen Audiodateien wurden transkribiert, anonymisiert und die Originale anschließend gelöscht. Zudem wurden alle persönlichen Informationen nach der Auswertung gelöscht. Direkte Zitate sind im Folgenden durch Anführungszeichen gekennzeichnet, auf Informationen über den Urheber wird aus Gründen des Datenschutzes jedoch verzichtet. Alle Teilnehmer haben zudem eine Datenschutzerklärung unterschrieben und bekamen die Möglichkeit, das anonymisierte Transkript Korrektur zu lesen.

Die Interviews sind unterteilt in drei Teile: (i) *persönliche Situation*, (ii) *Provenance und Privacy* sowie (iii) *Forschungsdatenmanagement*, wobei wir den ersten Teil quantitativ erhoben und ausgewertet haben. Die anderen beiden Teile wurden hingegen qualitativ untersucht.

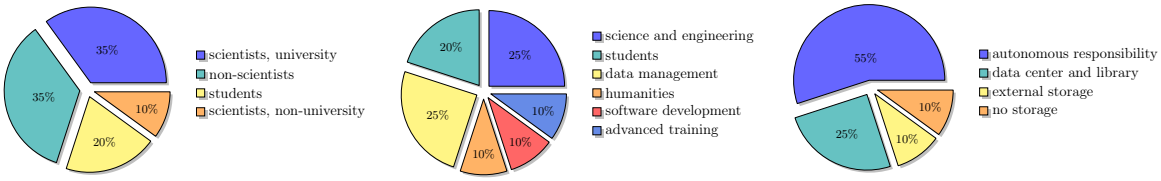
[1] Privacy Policy der DFG: https://www.dfg.de/en/service/privacy_policy/

Auswertung des Interviews

Details zu den einzelnen Fragen können in [Sch20; ASH21b] sowie im Anhang E nachgelesen werden. An dieser Stelle konzentrieren wir uns auf die Hauptergebnisse der Interviews sowie unsere Schlussfolgerungen bzgl. der Definitionen von Provenance, Privacy und Forschungsdatenmanagement.

Persönliche Situation Alle Befragten arbeiten regelmäßig mit (un-)strukturierten Daten, welche durch sie erhoben, verwaltet oder ausgewertet werden. Je nach Anwendungsbereich handelt es sich hierbei um eigene oder fremde Forschungsdaten, welche etwa durch andere Forschungsgruppen oder -institute bereit gestellt werden.

Die erhobenen und verwalteten Datenmengen variieren von wenigen kB bis hin zu mehreren TB. Auch die Speicherdauer variiert von wenigen Sekunden bis hin zu mehreren Jahrzehnten. Die lange Speicherdauer ergibt sich dabei insbesondere aus den Anforderungen von Drittmittelgebern sowie dem Interesse an Langzeitstudien. Einige Teilnehmer räumen jedoch ein, dass eine lange Speicherdauer auch an mangelnden Löschstrategien liegen kann. So werden einmal gesammelte Daten einfach *„geparkt und nie wieder angeschaut“*.



(a) Zugehörigkeit **(b)** Forschungsgebiet/Abteilung **(c)** Übernahme der Datenverwaltung

Abbildung 8.1. Ergebnisse des ersten Teils: Persönliche Situation.

In den meisten Fällen verbleibt die Datenverwaltung im Haus, also in der Bibliothek oder im Rechenzentrum (siehe Abbildung 8.1c). Etwa die Hälfte der Befragten verwaltet seine Daten selbst. Ein Viertel nutzt externe Anbieter oder benötigt gar keine Datenspeicherung.

Provenance und Privacy Ein Drittel der Befragten (Wissenschaftler und Nicht-Wissenschaftler) hat keine Assoziationen zum Begriff *Provenance*. Etwa die Hälfte der Befragten assoziiert Provenance mit der *„Herkunft von Daten oder (physischen) Objekten“* wie historischen Texten, Kunstwerken oder Artefakten. Konkret benannt werden beispielsweise die Nachverfolgung von Lebensmitteln, die Rückverfolgbarkeit von Datenauswertungsprozessen, die Überprüfung von veröffentlichten Forschungsergebnissen sowie der Ausschluss von *„Fake News“*. Weitere Assoziationen sind in der Wordcloud in Abbildung 8.2a zusammengefasst.

Den Begriff *Privacy* assoziieren die meisten Befragten (etwa 70%) mit dem Schutz personenbezogener Daten. Es darf also nicht möglich sein, aufgrund der vorliegenden Daten oder Aktivitäten Rückschluss auf eine bestimmte Person zu ziehen (siehe Abschnitt 3.6). Einige wenige Teilnehmer beziehen sich hingegen auf den Schutz allgemeiner Datensätze. Hierzu gehören neben den personenbezogenen Daten auch (patentierte) Forschungsdaten und -prozesse. Ihrer Meinung nach sind *„Daten generell schützenswert“*. Sie unterscheiden daher zwischen *„persönlichen und personenbezogenen Daten“*. Andere Teilnehmer verstehen Privacy als *„Prozess der Anonymisierung und Reduzierung von Daten auf ein Minimum“* und verweisen auf die *„Erstellung und Einhaltung von Vertraulichkeitserklärungen“* sowie die Beachtung von *„Datenschutzaspekten“*. Vereinzelt werden auch die Verwaltung der Daten sowie ihrer Zugriffsrechten genannt.

Um die Re-Identifizierung eines Datensatzes zu vermeiden, schlägt die Hälfte der Befragten das Unterdrücken von Spalten sowie die Generalisierung einzelner Attributwerte vor. Auch das Permutieren nicht korrelierender Spalten, das Unterdrücken einzelner Tupel oder das Kodieren einzelner Attributwerte werden genannt. Die Anonymisierungsmethode hängt dabei *„stark von der gestellten wissenschaftlichen Frage ab“*, da eine *„Anonymisierung immer einen Informationsverlust zur Folge hat“*.

(a) Assoziationen mit dem Begriff Provenance

(b) Gründe für die Nicht-Veröffentlichung von Forschungsdaten

Abbildung 8.2. Ergebnisse des zweiten und dritten Teils zu den Begriff Provenance und Open Data.

Forschungsdatenmanagement Forschungsdaten beschreiben *„Daten, die im Rahmen des Forschungsprozesses erzeugt oder benötigt werden“*. Hierbei unterscheiden wir zwischen Primärdaten (Rohdaten), Sekundärdaten (verarbeitete Daten) und Metadaten. Aggregieren wir diese, handelt es sich wiederum um Forschungsdaten, solange *„der Prozess dahinter klar ist“*. Dies beinhaltet die *„Forschungsmethode und -frage, die erzeugten Daten, Hypothesen, positive und negative Ergebnisse sowie die Zusammenfassung und Analyse der Daten“*.

Unter dem Begriff *Forschungsdatenmanagement* verstehen alle Befragten den allgemeinen „Umgang mit Forschungsdaten“. Für die Hälfte der Befragten steht dabei insbesondere die *„Archivierung, Speicherung und Löschung von Forschungsdaten“* im Vordergrund. Hierzu gehören neben der Wahl des Speichermediums auch die *„Gewährleistung der Langzeitspeicherung“* (Kompatibilität von Soft- und Hardware) sowie ein *„strukturiertes Speichermanagement“*. Der Schutz personenbezogener Daten sowie deren Anonymisierung sind für etwa 40% der Befragten ebenfalls Aufgabe von Forschungsdatenmanagement. Für ein weiteres Viertel hingegen ist die *Gewährleistung der Reproduzierbarkeit und Rückverfolgbarkeit"* die wichtigste Aufgabe des Forschungsdatenmanagements. Weitere Aufgaben sind in Abbildung 8.3 zusammengefasst. Provenance und Privacy scheinen somit (unabhängig voneinander) fester Bestandteil eines guten Forschungsdatenmanagements zu sein.

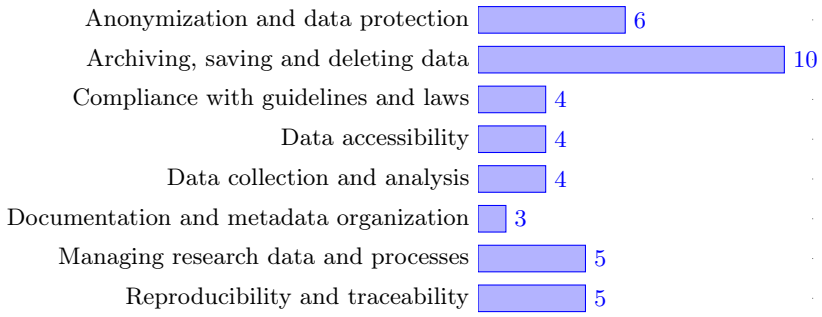


Abbildung 8.3. Ergebnisse des dritten Teils: Aufgaben des Forschungsdatenmanagements; mehrere Antworten pro Teilnehmer möglich.

Etwa die Hälfte der Befragten ist bereit, seine Daten zu teilen, *„solange der Datenschutz gewährleistet ist“*. Sie stellen ihre Daten nach Fertigstellung der zugehörigen Veröffentlichung oder des Forschungsprojekts grundsätzlich zur Verfügung oder machen ihre Entscheidung vom Einzelfall abhängig. So werden die Daten beispielsweise *„innerhalb der Abteilung jederzeit weitergegeben“*, während Anfragen von Außen *„individuell geprüft werden“*. Etwa 30% der Teilnehmer fürchten das *"Risiko von konkurrierenden Wissenschaftlern"*, welche die *„aufwendig generierten Daten“* ebenfalls als Publikationsgrundlage nutzen könnten. Sie veröffentlichen lediglich bearbeitete oder aggregierte Daten. Wiederum andere sehen die Veröffentlichung der eigenen Daten bereits als *„nennenswerte Forschungsleistung“*.

Einige der Befragten fühlen sich zudem verpflichtet, ihre Daten öffentlich zugänglich zu machen. Sie sprechen von einem Konflikt zwischen der Zurückhaltung der Daten zur Garantie der eigenen (wissenschaftlichen) Wettbewerbsfähigkeit und dem Wunsch nach Reproduzierbarkeit und Nachvollziehbarkeit im Sinne einer guten Forschung, was wiederum die Veröffentlichung von Forschungsdaten verlangen würde. Die Gründe, die gegen eine Veröffentlichung der Daten sprechen, sind nach Aussage der Befragten vielzählig. Hierzu gehören etwa die *„eigene Reputation“*, *„Datenschutzaspekte“*, *„finanzielle, politische und wirtschaftliche Interessen“*, *„Patentansprüche“*, eine *„aufwendige*

Datengenerierung“, „*ethische Gründe*“ oder „*die Freiheit der Forschung*“ (siehe Abbildung 8.2b). Bei einem Viertel der Befragten liegt die Entscheidung über die (Nicht-)Veröffentlichung der Daten jedoch nicht beim Wissenschaftler selbst, sondern bei den Projektpartnern oder Forschungsförderern wie der Deutschen Forschungsgemeinschaft (DFG), der Europäischen Union (EU) oder dem Bundesministerium für Bildung und Forschung (BMBF). Auch unsere Interview-Teilnehmer stimmen darin überein, dass drittmittelgeförderte Forschung grundsätzlich öffentlich zugänglich sein sollte.

Zusammenfassung und Diskussion

Zusammengefasst stellen wir fest, dass die wenigsten Befragten mit dem Begriff Provenance vertraut sind. Zudem beschränken sich die Privacy-Definitionen der Befragten in den meisten Fällen auf den Schutz personenbezogener Daten. Wir erweitern diese Definition auf Forschungsdaten im Allgemeinen wie folgt [ASH21a]:

Definition 8.1 (Privacy)**.** *Privacy* bezeichnet den Schutz allgemeiner (Forschungs-)Daten vor unzulässiger Erhebung, Speicherung und Veröffentlichung. Es soll dabei nicht möglich sein, Rückschlüsse auf einzelne Tupel eines Datensatzes zu ziehen. ■

Unsere Privacy-Definition geht dabei bewusst über den Schutz personenbezogener Daten hinaus. Für uns sind Forschungsdaten jeglicher Art schützenswert, da diese oft sensible Attributewerte enthalten oder mit großem finanziellen oder zeitlichen Aufwand erhoben wurden. Dennoch existieren in beinahe allen (Forschungs-)Datensätzen Teile, welche problemlos veröffentlicht werden können, ohne Privacy-Aspekte zu verletzten. Eine Anonymisierung der Daten zur Einhaltung von Privacy-Aspekten mit anschließender Veröffentlichung ist für uns daher zwingender Teil eines guten Forschungsdatenmanagements, um so die Nachvollziehbarkeit, Replizierbarkeit und Reproduzierbarkeit von veröffentlichten Forschungsergebnissen zu gewährleisten bzw. zu erhöhen.

Der Umgang mit personengebundenen Daten sowie deren Anonymisierung ist den meisten Befragten bekannt. Sie erkennen einen datenschutz-problematischen Datensatz und kennen die gängigsten Anonymisierungsmethoden wie Generalisierung, Unterdrückung oder Permutation. Alle Teilnehmer haben zudem eine klare Vorstellung davon, was Forschungsdaten eigentlich sind und wie diese zu verwalten sind. Die Meinungen über Open Data und Open Science gehen jedoch stark auseinander. Hier entscheiden neben Policies von Projektpartnern und Forschungsförderern oft individuelle Gründe für oder gegen eine Veröffentlichung.

Die Auswertung der Interviews bestätigt somit zwei typische Konflikte, welche im Umgang mit Forschungsdaten entstehen:

- Wissenschaftlicher Wettbewerb/Wissenschaftliche Freiheit vs. Wunsch nach Nachvollziehbarkeit und Reproduzierbarkeit im Sinne guter Forschung;
- Rückverfolgbarkeit von Forschungsergebnissen vs. Schutz von (personenbezogenen) Daten.

Der erste Konflikt ist nicht neu und wird schon seit längerem untersucht. Für nachhaltig nutzbare Forschungsdaten und eine gute Forschungsdateninfrastruktur wurden daher 2016 die sogenannten *FAIR Data Principles* formuliert[2]. Sie geben eine Leitlinie vor, um die Daten Auffindbar (**F**indable), Zugänglich (**A**ccessible), Interoperabel (**I**nteroperable), Wiederverwendbar (**R**eusable) zu halten. Der zweite Konflikt soll im Rahmen unseres Projektes ProSA gelöst werden. Unsere Ergebnisse hierzu sind in Kapitel 10 zusammengefasst.

8.2. Anonymisierung der *where*, *why*- und *how*-Provenance *(basierend auf [ASH21a])*

Wie in Abschnitt 8.1 beschrieben, geht der Begriff *Privacy* für uns über das Konzept des (meist persönlichen) Datenschutzes hinaus. Wir verstehen unter Privacy vielmehr den Schutz von (Forschungs-)Daten im Allgemeinen. Da Auswertunganfragen, welche im Rahmen von Projekten auftreten, oft beliebig komplex werden können, ist ihre Invertierung oft nicht zu 100% möglich oder notwendig. Dies hat einen hohen Informationsverlust zur Folge. Zusätzliche Provenance-Informationen ermöglicht die Angabe einer bestmöglichen Inversen (siehe Abschnitt 6.3

[2] FAIR-Principles: `http://www.go-fair.org/fair-principles/` und `https://www.force11.org/group/fairgroup/fairprinciples`

sowie Abschnitt 7.3). Daraus ergibt sich ein natürlicher Interessenkonflikt zwischen der Veröffentlichung der (rekonstruierten) Originaldaten und dem Schutz dieser Daten. Unser Ziel ist es also, die rekonstruierte (minimale) Teil-Datenbank I^* möglichst akkurat im Sinne der Provenance und zugleich möglichst allgemein im Sinne der Privacy zu halten.

Probleme der verschiedenen Provenance-Anfragen Je nach gewählter Provenance-Anfrage kann für eine gegebene Quell-Instanz I sowie eine Auswertungsanfrage Q eine andere Instanz I^* rekonstruiert werden. Der Provenance-Anfragetyp beeinflusst zudem den Chase-Inversentyp. Während die ***where***-Provenance häufig eine ergebnisäquivalente oder relaxte Chase-Inverse liefert, garantieren die ***why***- und ***how***-Provenance in der Regel eine tp-relaxte oder exakte Chase-Inverse (vgl. Tabelle 6.4, Tabelle 7.6 und Tabelle 7.7). Die Reduktion der Provenance-Anfragen

$$\textbf{\textit{where}} \preccurlyeq \textbf{\textit{why}} \preccurlyeq \textbf{\textit{how}}$$

garantiert zudem, dass wir für die Einbindung von Provenance zur Berechnung der (minimalen) Teil-Datenbank lediglich eine dieser Provenance-Anfragen benötigen. So liefert die ***how***-Provenance mehr Informationen als die ***where***-Provenance, hat damit aber vermutlich auch das größere Risiko zur Verletzung von Privacy-Aspekten. Schauen wir uns daher zunächst die möglichen Datenschutz-Probleme bei der Verwendung der verschiedenen Provenance-Anfragen an:

(1) Bei der Verwendung der ***where***-Provenance, welche einer Liste der für das Ergebnis relevanten Tupel-IDs entspricht, liegen im Allgemeinen nicht genügend schützenswerte Daten vor (siehe linke Teil-Datenbank in Abbildung 8.4), um ein Datenschutz-Problem zu verursachen. Eine weitere Rekonstruktion der originalen Daten ist daher oft nicht möglich. Dies spiegelt sich auch im zugehörigen Inversentyp wieder. So sind Chase-Inverse, welche durch zusätzliche ***where***-Provenance berechnet wurden, oft ergebnisäquivalent, d.h. die (minimale) Teil-Datenbank enthält in der Regel lediglich das auf das Quell-Schema erweiterte Anfrageergebnis (siehe Abbildung 8.4, linke Teil-Datenbank). Zusätzliche Informationen, welche das bereits veröffentlichte Ergebnis übersteigen, werden somit nicht preisgegeben und weitere Privacy-Aspekte können vernachlässigbar.

Interpretieren wir die ***where***-Provenance jedoch als Tupel-IDs bzw. Relationennamen mit deren Hilfe die Tupel der Quell-Instanz erneut ausgelesen, kann dies zu erheblichen Datenschutz-Problemen führen. Da ein erneutes Auslesen der Quell-Instanz in ProSA jedoch nicht vorgesehen ist, vernachlässigen wir diesen Fall im Folgenden.

(2) Auch die Einbindung der ***why***-Provenance kann auf den ersten Blick zu erheblichen Datenschutz-Problemen führen. So werden für eine Selektionsanfrage beispielsweise große Teile der Originaldatenbank in der Zeugenbasis abgespeichert. Da aber nur die Tupel-IDs gespeichert werden und die Datenbank nicht erneut ausgelesen wird, können auch hier nur wenige Informationen über das Anfrageergebnis hinaus rekonstruiert werden. Hierzu gehört unter anderem eine Liste aller am Ergebnis beteiligten Tupel, jedoch ohne konkrete Attributwerte. Die (minimale) Teil-Datenbank enthält damit die richtige Tupelanzahl, was oft eine tp-relaxte Chase-Inverse zur Folge hat, jedoch keine konkreten Attribute (siehe mittlere Teil-Datenbank in Abbildung 8.4).

(3) Mit Hilfe der ***how***-Provenance kann die originalgetreueste (minimale) Teil-Datenbank rekonstruiert werden. Dabei werden oft jedoch zu viele Informationen rekonstruiert, so dass die Verletzung von Privacy-Aspekten bei dieser Variante wahrscheinlich ist. Der Unterschied zur ***why***-Provenance besteht darin, dass die Provenance-Polynome der ***how***-Provenance die Rekonstruktion konkreter Attributwerte ermöglicht (siehe rechte Teil-Datenbank in Abbildung 8.4). Dies ist insbesondere bei der Auswertung von Aggregatfunktionen der Fall.

Beispiel 8.1 (basierend auf [ASH21a])**.** Betrachten wir erneut die Relation `Grade` eingeschränkt auf die Attribute `studentID`, `courseID` und `grade`. Diese personenbezogenen Daten sollen nun mit Hilfe der drei Data Provenance-Anfragen ***where***, ***why*** und ***how*** anonymisiert werden. Sei weiter Q die Auswertungsanfrage, welche die Durchschnittsnote pro Studierenden berechnet. Da das Attribut `courseID` für die Auswertung von Q nicht relevant ist, können wir hier lediglich Nullwerte η_i rekonstruieren (grau unterlegt). Die Matrikelnummer (`studentID`) hingegen bleiben vollständig erhalten. Interessant ist somit das zu aggregierende Attribut `grade`.

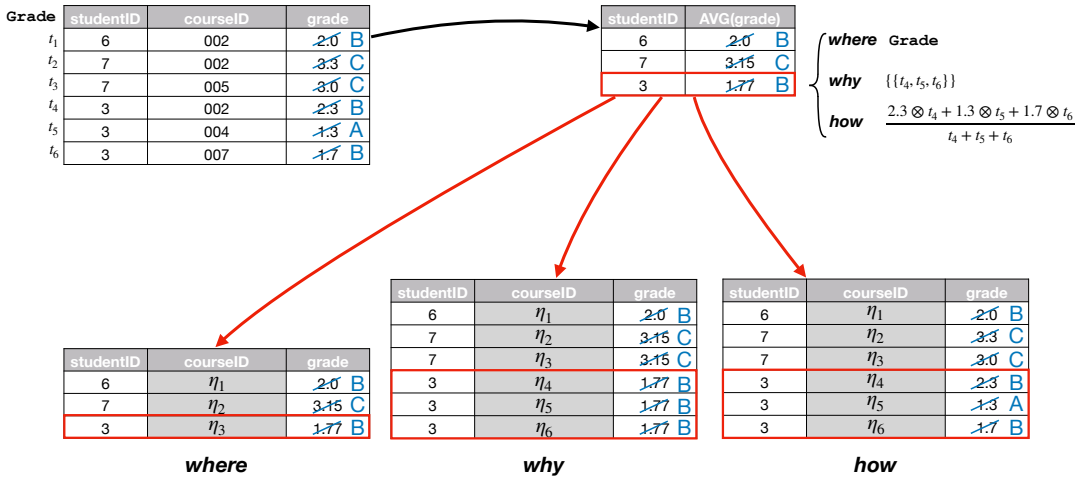


Abbildung 8.4. Mögliche Provenance-basierte Datenbankrekonstruktionen (rot hervorgehoben) inklusive Generalisierung (blau gekennzeichnet) als Privacy-Ansatz.

Je nach Wahl der verwendeten Provenance können nun verschiedene Teil-Datenbanken berechnet werden. Abbildung 8.4 zeigt die Ergebnistabellen im Vergleich zur Quell-Instanz (einem Ausschnitt der Instanz aus Anhang A). Die ***where***-Provenance (links) liefert die kleinste Teil-Datenbank. Die Erweiterung auf die exakte Anzahl der Tupel ist mittels ***why***-Provenance (Mitte) möglich und die Darstellung aller Einzelnoten pro Studierender mit Hilfe der ***how***-Provenance (rechts).

Auch der Chase-Inversentyp hängt von der Wahl der Provenance-Anfrage ab. So liefert die ***where***-Provenance eine ergebnisäquivalente Chase-Inverse. Die ***why***- sowie die ***how***-Provenance hingegen garantieren eine tp-relaxte Chase-Inverse. □

Vergleich und Auswahl von *where*, *why* und *how* Die Wahl der Provenance-Anfrage kann also die Privacy erhöhen bzw. bedrohen. Wie in Beispiel 8.1 zu sehen, rekonstruiert die Zeugenliste der ***where***-Provenance lediglich einen minimalen Teil der Originaldaten. Konkret können zwei Probleme auftreten:

1. die Tupelanzahl geht verloren, d.h. es gilt $|I^*| < |I|$ [3];
2. die (minimale) Teil-Datenbank I^* ist kein Ausschnitt von I, d.h. es existiert kein Homomorphismus $h : I^* \to I$.

Beide Aspekte gehören jedoch zu den in Abschnitt 2.1 und Abschnitt 5.1 geforderten Nebenbedingungen. Sie können insbesondere bei der Verwendung von Aggregatfunktionen wie beispielsweise `COUNT` zum Problem werden. So besteht die (minimale) Teil-Datenbank von oben nach der Rekonstruktion beispielsweise aus einem Tupeln der Form (η_1, 6, η_2). Die 6 ist jedoch nirgendwo als Attributwert verzeichnet. Die anderen fünf Tupel gehen zudem aufgrund von Duplikateleminierung verloren. Da der Informationsverlust der ***where***-Provenance somit im Allgemeinen zu hoch ist und eine tp-relaxte Chase-Inverse nur selten garantiert werden kann, kommt diese als Provenance-Anfrage in ProSA nicht in Frage.

Die Verwendung der ***why***- sowie der ***how***-Provenance hingegen garantiert in den meisten Fällen die Existenz einer tp-relaxten oder exakten Chase-Inversen. Damit bleibt zum einen die richtige Tupelanzahl erhalten, d.h. $|I^*| = |I|$, und zum anderen ist I^* Ausschnitt von I. Beide Provenance-Anfragen genügen somit unseren Ansprüchen zur Rekonstruktion der (minimalen) Teil-Datenbank I^*. Wie oben beschrieben kann es jedoch zur Verletzung von Privacy-Aspekten kommen.

Das Provenance-Polynom $t_1 \cdot t_2 + t_3$ entspricht der Zeugenbasis $\{\{t_1, t_2\}, \{t_3\}\}$. Beide Darstellungen enthalten Informationen über den natürlichen Verbund ($t_1 \cdot t_2$ bzw. $\{t_1, t_2\}$) sowie das Duplikat ($t + t_3$ bzw. $\{\{t\}, \{t_3\}\}$). Für SPJU-Anfragen liefern die ***why***- sowie die ***how***-Provenance somit den selben Informationsgehalt. Da der Aufwand zur Erstellung der Provenance-Polynome jedoch deutlich höher als bei der Generierung der Zeugenbasen ist, entscheiden wir uns in ProSA für die ***why***-Provenance.

[3] Wir gehen davon aus, dass I bereits auf die für die Auswertung der Anfrage Q relevanten Tupel eingeschränkt ist. Enthält I hingegen Tupel, welche für die Auswertung von Q absehbar nicht benötigt werden, kann $|I^*| = |I|$ niemals garantiert werden.

Anders verhält es sich jedoch bei aggregierten Daten. In diesem Fall liefert das Provenance-Polynom eine Berechnungsvorschrift, welche zudem konkrete Attributwerte enthält. Mit Hilfe dieser Informationen können die aggregierten Daten in der (minimalen) Teil-Datenbank I^* rekonstruiert werden. Die Zeugenbasen der ***why***-Provenance erlauben die Rekonstruktion dieser meist sensiblen Daten hingegen nicht (siehe Attribut `grade` in der mittleren und rechten Teil-Datenbank in Abbildung 8.4). Speichern wir die zu aggregierenden Attributwerte jedoch in einer zusätzlichen Side Table (siehe Definition 6.10), können wir den Informationsgehalt der ***why***- sowie der ***how***-Provenance als gleichwertig ansehen. Schauen wir uns dies abschließend noch einmal an einem konkreten Beispiel an.

Beispiel 8.2 (Fortsetzung)**.** Das Ergebnistupel $(3, 1.77)$ kann durch die Zeugenbasis $\{\{t_4, t_5, t_6\}\}$ sowie das Provenance-Polynom $\frac{2.3 \otimes t_4 + 1.3 \otimes t_5 + 1.7 \otimes t_6}{t_4 + t_5 + t_6}$ beschrieben werden. Hieraus können die zugehörigen Quell-Tupel $I^*_{\boldsymbol{why}} = \{(3, \eta_4, 1.77), (3, \eta_5, 1.77), (3, \eta_6, 1.77)\}$ und $I^*_{\boldsymbol{how}} = \{(3, \eta_4, 2.3), (3, \eta_5, 1.3), (3, \eta_6, 1.7)\}$ abgeleitet werden. Wie erwartet können die exakten Werte des Attributes `grade` nicht rekonstruiert werden. Speichern wir diese jedoch in einer zusätzliche Side Table $K' = \{(2.3, t_4), (1.3, t_5), (1.7, t_6)\}$, so gilt $I^*_{\boldsymbol{why}} = I^*_{\boldsymbol{how}}$. □

Wir entscheiden uns daher für die Verwendung der ***why***-Provenance. Ob die Zeugenbasen minimal gewählt sind oder nicht, spielt dabei keine Rolle. Beide liefern in Kombination mit den Side Tables den selben Informationsgehalt, sodass wir uns den Mehraufwand für die Einschränkung der Zeugenbasen auf ihren minimalen Kern in ProSA sparen (siehe Abschnitt 10.7). Interessanter ist die Wahl der Methode, welche wir für die Anonymisierung der ***why***-Provenance verwenden werden.

8.3. Relevante Anonymisierungsmethoden

Die ***why***- sowie die ***how***-Provenance können somit (mit und ohne zusätzliche Side Tables) zu erheblichen Datenschutz-Problemen führen. Um diese zu beheben, eignen sich verschiedene Ansätze wie etwa die Generalisierung, das Unterdrücken von Tupeln oder Attributen, Differential Privacy, die Permutation von Tupeln sowie intensionale (anstellen von extensionalen) Provenance-Antworten (siehe Abbildung 8.5). So kann das zu schützende Attribut `grade` beispielsweise durch Generalisierung anonymisiert werden:

Beispiel 8.3 (Fortsetzung)**.** Für die Anonymisierung generalisieren wir die Attributwerte 1.0 und 1.3 zum Notenbereich A, die Attributwerte 1.7, 2.0 und 2.3 zu B usw. (blau hervorgehoben). Dies führt zu einem (hoffentlich akzeptablen) Informationsverlust, während wir gleichzeitig das sensible Attribut schützen. Eine Reproduktion des Anfrageergebnisses $(3, 1.77)$ ist nach der Generalsierung des Attributes `grade` nicht mehr möglich. Ein Plausibilitätscheck des veröffentlichten Ergebnisses bleibt jedoch weiterhin möglich, da das anonymisierte Ergebnistupel $(3, B)$ durch den oben beschriebenen Homomorphismus auf das originale Ergebnistupel $(3, 1.77)$ abgebildet werden kann. □

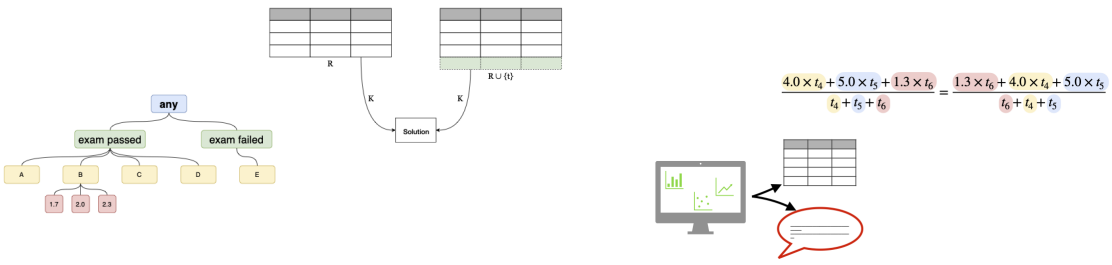


Abbildung 8.5. Anonymisiserungsmethoden: Generalisierung, Differential Privacy, Intensionale Antworten und Permutation (von links nach rechts).

Da die originale Auswertungsanfrage Q auf die anonymisierten (minimalen) Datenbank I^*_{anon} in der Regel nicht mehr angewandt werden kann, müssen wir diese entsprechen transformieren und bereitstellen. Für die angepasste Anfrage Q' sowie die originale Auswertungsanfrage Q muss dabei gelten:

$$Q(I^*) \preccurlyeq Q(I) \text{ sowie } Q'(I^*_{\text{anon}}) \preccurlyeq Q(I^*),$$

d.h. es existiert ein Homomorphismus h, welcher die konkreten Attributwerte aus I^* auf die anonymisierten Werte aus I^*_{anon} abbildet. Wie diese Transformation konkret aussieht, hängt von der der Anonymisierung zugrundeliegenden Konzepthierarchie (siehe Definition 8.2) ab. Auch hier können weiterhin zusätzliche Provenance-Informaitonen genutzt werden[4]. Für die konkrete Umsetzung der Transformation von Q zu Q' verweisen wir unter anderem auf [Lam21; Sch22].

Abschließend bleibt die Frage, in welchen Fällen auf der anonymisierten (minimalen) Teil-Datenbank die transformierte Auswertungsanfrage Q' durchgeführt werden kann und welche zusätzlichen Einschränkungen zu beachten sind. Ein entsprechender Ansatz zur „provenance-unterstützten Datenanlyse zur Steigerung der Privatsphäre“ kann beispielsweise in [Lam21] nachgelesen werden. Lamster stellt dabei fest, dass lediglich „einfache“ Anfragen privacy-konform umgesetzt werden können.

In ProSA entscheiden wir uns bei der Anonymisierung für die Generalisierung, welche die Intensionalisierung zur Folge hat, sowie die Permutation (siehe Abschnitt 10.7). Zu welchem Zeitpunkt der ProSA-Pipeline (Quell-Instanz, Anfrageergebnis, rekonstruierte Teil-Datenbank) die Anonymisierung am sinnvollsten ist, wird ebenfalls in Abschnitt 10.7 diskutiert werden. Eine Generalisierung der Provenance entfällt, da wir die ***why***-Provenance bereits als Generalisierung der ***how***-Provenance ansehen. Wann und in welchem Maße zusätzliche Side Tables eingesetzt werden soll, kann vom Nutzer selbst entschieden werden. Wir sehen ihren Einsatz daher nicht als Privacy-Problem an.

Generalisierung Die Generalisierung einer Datenbank ermöglicht die Anonymisierung einzelner Attributwerte durch ihre Ersetzung mit einer passenden intensionalen Beschreibung. Die so „verschleierten“ Daten können nicht mehr konkret ausgelesen werden. Ein übliches Hilfsmittel, welches die Generalisierung ermöglicht, sind die sogenannten *Konzepthierarchien* nach Definition 8.2.

Definition 8.2 (Konzept-Hierarchie)**.** Sei $\mathbb{C}$ eine nicht-leere endlichen Menge und $\leq$ eine partielle Ordnung auf $\mathbb{C}$. Sei zudem $\top$ das größte und $\bot$ das kleinste Element bzgl. $\mathbb{C}$. Dann ist $(\mathbb{C}, \leq)$ eine *Konzepthierarchie*. Wir sagen, c' liegt auf einer höheren Begriffsebene als c', wenn $c, c' \in \mathbb{C}$ und $c \leq_{\mathbb{C}} c'$ gilt. ■

Sei weiter $R_i = \{A_{i_1}, ..., A_{i_n}\}$ mit $i \in \{1, ..., p\}$ eine Menge von Attributen und R_i eine Menge von Relationenschemata. Dann nenne wir $D := \{r_1, ..., r_n\}$ *Basis-Datenbank* und $r_i \in D$ *Basis-Relation* zu D. Dann nennen wir den Prozess, welcher eine Datenmenge in einer Datenbank von einer niedrigen Begriffsebene innerhalb der Konzepthierarchie auf eine höhere abstrahiert, *Generalisierung* der Datenbank.

Definition 8.3 (Generalisierung einer Datenbank)**.** Eine Datenbank D' ist *Generalisierung einer Basis-Datenbank* D genau dann, wenn gilt:

1. die Anzahl der Tupel in D und D' stimmen überein, d.h. $|D| = |D'|$;
2. für alle Attribute $A \in r(R)$ existiert eine Konzepthierarchie, d.h. $A_D \leq_{\mathbb{C}} A_{D'}$;
3. es existiert ein Homomorphismus $h : D \to D'$ mit $h(t) = t'$ und $t'.A \geq t.A$ für alle Attribute $A \in r(R)$. ■

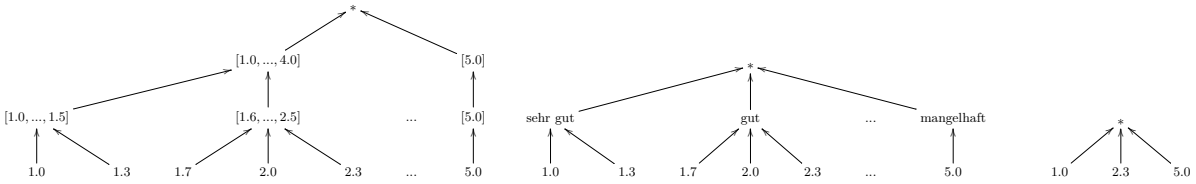
Basierend auf [PY99] und [Sva16] unterscheidet [Lam21] bei der Generalisierung mittels Konzepthierarchien vier verschiedene Typen von Attributen:

- **Typ 1:** Numerische Attribute mit numerischer Konzepthierarche
- **Typ 2:** Beliebige Attribute mit beliebiger Konzepthierarchie
- **Typ 3:** Beliebige Attribute ohne sinnvolle Hierarchie
- **Typ 4:** Attribute ohne Generalisierung

[4] Zu beachten ist jedoch, dass bei der Implementierung von ProSA die Provenance-Informationen nach Berechnung der (minimalen) Teil-Datenbank vernachlässigt werden, sodass Duplikate, welche während der Anonymisierung entstehen, in I^*_{anon} verloren gehen. Grund hierfür sind Schwierigkeiten bei der praktischen Umsetzung.

Die Klassifizierung der Attribute ist intuitiv, weshalb auch wir sie im Folgenden verwenden werden. Attribute des ersten Typs können beispielsweise durch Intervall-Bildung anonymisiert werden. So können die konkreten Attributwerte 1.0, 1.3, 1.7, 2.0 usw. des Attributes `grade` zu den Notenbereichen $[1.0, ..., 1.5]$ oder $[1.6, ..., 2.5]$ und anschließend $[1.0, ..., 4.0]$ zusammengefasst werden. Fassen wir `grade` hingegen als Attribut vom Typ 2 auf, so generalisieren wir die Noten 1.0 und 1.3 zu *sehr gut* und die Noten 1.7, 2.0 und 2.3 zu *gut*.

Typ 3 umfasst nun alle Attribute, die in keine sinnvolle Hierarchie eingeordnet werden können, aber trotzdem generalisiert werden sollen. Die Konzepthierarchie entspricht hier in der Regel den originalen Attributwerten sowie einem allgemeinen Wurzelknoten $*$ oder Any. Abbildung 8.6 fasst die verschiedenen Generalisierungstpen noch einmal zusammen. Die nicht-generalisierbaren Attribute, welche beispielsweise während der Anfrage herausprojiziert werden, fallen unter den Typ 4.



(a) Typ 1 **(b)** Typ 2 **(c)** Typ 3

Abbildung 8.6. Konzepthierarchien für die drei Attributtypen 1 bis 3

Mit Hilfe der Generalisierung mittels Konzepthierarchien kann die k-Anonymität (siehe Definition 3.43) der anonymisierten Datenbank gewährleistet werden werden. Dazu definieren wir eine k-anonyme Generalisierung wie in Definition 8.4 beschrieben. Die generalisierten Datenbank D entspricht damit einer Basis-Datenbanken D', welche D generalisiert.

Definition 8.4 (k-anonyme Generalisierung)**.** Eine Datenbank D' ist eine *k-anonyme Generalisierung* bzgl. einer Menge von Attributen A einer Datenbank D, wenn für jedes Tupel t mindestens k-1 Tupel t_i existieren, sodass $t.A = t_i.A$. ■

Das Ergebnis der Generalisierung ist oftmals eine intensionale Beschreibung der originalen Datenbank. Dies ist insbesondere bei nicht-numerischen Attributwerten der Fall. In ProSA verschmelzen die Generalisierung sowie die Intensionalierung daher immer wieder.

Intensionalisierung Die Anfragetypen der Data Provenance (***where***, ***why***, ***how*** und ***why not***) können sowohl extensional, d.h. durch Angabe konkreter Tupel, als auch intensional, d.h. beschreibend, beantwortet werden. Verschiedene Ansätze zur Bildung intensionaler Provenance-Antworten werden unter anderem in [PY99; Sva16] und [Lam21] beschrieben. Fassen wir diese kurz zusammen.

Intensionale Provenance-Antworten stellen einen Lösungsansatz dar, um Data Provenance und Datenschutz zu kombinieren. Sie kann z.B. durch die Generalisierung von Attributwerten realisiert werden. Die Idee der Verallgemeinerung wird in Abbildung 8.4 durch die Verallgemeinerung der Note von einer konkreten Zahl wie 1,0 oder 1,3 auf einen Notenbereich von A (blau hervorgehoben) dargestellt. Dies führt zu einem (hoffentlich akzeptablen) Informationsverlust, während man sich gleichzeitig einer Lösung für das Problem des Schutzes sensibler Attributwerte nähert.

Für die Generierung intensionaler Provenance-Antworten beschreibt Svacina in [Sva16] eine Methode, bei der ein extensionales Ergebnis entlang einer eigens definierten Konzepthierarchie generalisiert wird. Er erweitert hierfür den in [PY99] vorgestellten Algorithmus, welcher aus einem *Preprocessing* zur Erstellung einer Konzepthierarchie, der *Query Execution* zur Auswertung der Anfrage und zur Bestimmung der zu generalisierenden Attribute sowie der *Answer Generation* zur rekursiven Generalisierung der ausgewählten Attribute besteht. In zwei zusätzlichen Schritten berechnet Svacina die Provenance der Auswertungsanfrage und generalisiert diese zur Formulierung einer intensionalen Antwort. Bei der zugrundeliegenden Konzepthierarchie unterscheidet er drei verschiedene Arten von Attributen: numerische Werte sowie nicht-numerische Werte mit und ohne Hierarchie. Auch wir werden in Abschnitt 10.7 eine ähnliche Konzepthierarchie verwenden.

Ein weiterer Ansatz zur Generierung intensionaler Provenance-Anfragen ist in [Lam21] zu finden. Auch in diesem Fall wird die Generalisierung mittels Konzepthierarchien genutzt, um einen provenance-gestützten anonymisierten Datensatz zu bestimmen. Hierfür entkoppelt Lamster das in [Sva16] vorgestellte Verfahren von PERM, dem System, welches zuvor für die Berechnung der Provenance verwendet wurde. Zusammengefasst verallgemeinert Lamster das Konzept von Svacina, welcher wiederum den in [PY99] entwickelten Ansatz erweitert.

Weitere Anonymisierungsverfahren Eine Permutation der rekonstruierten Quell-Tupel kann die Identifizierung konkreter Tupel verhindern, sollte die (minimale) Teil-Datenbank mit anderen, strukturgleichen Datenquellen verbunden werden. Dafür werden die Tupel der (minimalen) Teil-Datenbank nach Abschluss der Generalisierung beliebig vertauscht. Unter Beachtung der Tupel-IDs können Cluster der Original-Reihenfolge vermieden werden.

Tupel, welche mittels Generalisierung nicht zufriedenstellend anonymisiert werden können, d.h. welche etwa der k-Anonymität widersprechen, können unterdrückt werden. Dies ist im Fall von ProSA jedoch in den meisten Fällen nicht notwendig (siehe Abschnitt 10.7). Weitere Anonymisierungsmethoden sowie ihre Einsatzmöglichkeiten in Kombination mit Data Provenance können in [Sch20; Sch22] oder [Lam21] nachgelesen werden.

8.4. Zwischenfazit

Es existiert ein natürlicher Konflikt zwischen der Speicherung und Veröffentlichung von Daten (*Provenance*) sowie dem Schutz dieser Daten (*Privacy*), welcher auch in der Literatur noch nicht beantwortet wurde. Dieser ist vielen Kollegen jedoch nicht bewusst. Welche Gründe für oder gegen eine Veröffentlichung von (Forschungs-)daten sprechen können, haben wir in einer Interview-Studie untersucht (siehe Abschnitt 8.1).

Für die Integration in ProSA haben wir zudem die drei Anfragetypen ***where***, ***why*** und ***how*** der Data Provenance auf ihre möglichen Anonymisierungen untersucht. Aufgrund ihrer einfachen Berechnung sowie ihrer Rekonstruktionsgenauigkeit — bei der Verwendung zusätzlicher Side Tables ähnlich wie zu den Polynomen der ***how***-Provenance — haben wir uns für die Zeugenbasen der ***why***-Provenance entschieden. Ob diese minimal sind oder nicht, spielt für uns dabei keine Rolle (siehe Abschnitt 8.2).

In Abschnitt 3.6 haben wir verschiedene Anonymisierungsmethoden und -maße vorgestellt, aus denen wir uns in Abschnitt 10.7 jeweils eine heraussuchen werden. Wir entscheiden uns unter anderem für die Generalisierung. Das Ergebnis der Generalisierung ist oftmals eine intensionale Beschreibung der originalen Datenbank. Dies ist insbesondere bei nicht-numerischen Attributwerten der Fall (siehe Abschnitt 8.3). In ProSA verschmelzen die Generalisierung sowie die Intensionalisierung daher immer wieder. Neben diesen beiden Anonymisierungsmethoden verwenden wir in ProSA zudem die Unterdrückung sowie das Permutieren von Tupeln. Da die originale Auswertungsanfrage Q auf die anonymisierten (minimalen) Datenbank I^*_{anon} in der Regel nicht mehr angewandt werden kann, müssen wir diese entsprechen transformieren und bereitstellen. Wir verweisen hierfür auf [Sva16; Lam21; Sch22].

Eigene sowie betreute Arbeiten

Die in diesem Kapitel vorgestellten Ergebnisse basieren auf den folgenden Veröffentlichungen:

- [AH18a] **Tanja Auge**, Andreas Heuer: Combining Provenance Management and Schema Evolution. *ProvenanceWeek*, 2018
- [Aug20] **Tanja Auge**: Extended Provenance Management for Data Science Applications. *PhD@VLDB*, 2020
- [Sch20] Nic Scharlau: Provenance und Privacy in ProSA. *Bachelorarbeit*, 2020
- [ASH21a] **Tanja Auge**, Nic Scharlau, Andreas Heuer: Privacy Aspects of Provenance Queries. *ProvenanceWeek*, 2020

- [ASH21b] **Tanja Auge**, Nic Scharlau, Andreas Heuer: Provenance and Privacy in ProSA – A Guided Interview on Privacy-Aware Provenance. *DEXA Workshops*, 2021
- [Lam21] Maximilian Lamster: Provenance-unterstütze Datenanalyse in Kombination mit intensionalen Antworten zur Steigerung der Privatsphäre. *Masterarbeit*, 2021
- [Sch22] Nic Scharlau: Anonymisierung von Data Provenance in ProSA. *Masterarbeit*, 2022

Die Ergebnisse der Interviewstudie basieren auf der in [Sch20] durchgeführten Interviews. Die in Abschnitt 8.1 vorgestellten Ergebnisse weichen in einigen wenigen Punkten von den Ergebnissen aus [Sch20] ab. Grund hierfür ist eine erneute Auswertung der anonymisierten Interviews im Rahmen des Artikels [ASH21b]. Die Ergebnisse der Abschnitte 8.3 und 8.2 sind zum Teil noch nicht veröffentlicht. Erste Ansätze sind jedoch in [AH18a; Aug20; ASH21a; Sch22] zu finden.

9. ChaTEAU — Chase for Transforming, Evolving, and Adapting databases and queries

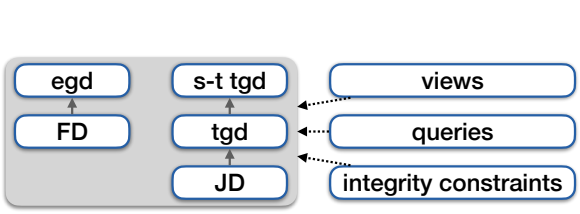
Der Chase hat viele verschiedene Anwendungsmöglichkeiten. Wie in Abschnitt 3.3.2 beschrieben, gehören hierzu neben der semantischen Optimierung, Data Exchange und Data Integration auch Answering Queries under Views sowie Data Cleaning. Während die Chase-Theorie mittlerweile gut verstanden ist, sind bestehende Implementierungen jedoch stets auf einen bestimmten Anwendungsfall bzw. ein Szenario beschränkt, was ihre Wiederverwendung in anderen Umgebungen erschwert. ChaTEAU (**Chase** for **T**ransforming, **E**volving, and **A**dapting databases and queries, **U**niversal Approach) hingegen überwindet diese Einschränkung. Hierfür nehmen wir den logischen Kern des Chase und verallgemeinern die Abhängigkeiten, Anfragen und Instanzen zu einem allgemeinen Chase-Parameter $*$ sowie einem allgemeinen Chase-Objekt $\bigcirc$ (siehe Abschnitt 9.1). ChaTEAU (siehe Abschnitt 9.2) selbst ist eine Implementierung des Chase, welche basierend auf dem Chase-Parameter und Chase-Objekt verschiedene Chase-Anwendungen in einem einzigen Toolkit vereint (siehe Abschnitt 9.3).

9.1. Verallgemeinerung des CHASE auf Anfragen und Instanzen *(basierend auf [Aug+22])*

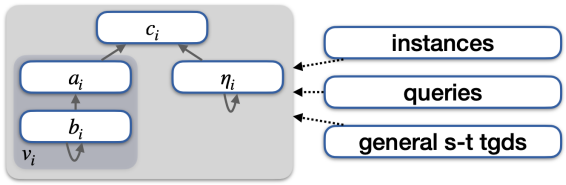
Fassen wir noch einmal zusammen: Der Chase modifiziert ein gegebenes Objekt $\bigcirc$, genannt Chase-*Objekt*, durch Integration eines Chase-*Parameters* $*$. Während $\bigcirc$ sowohl Anfragen als auch Instanzen darstellen kann, verstehen wir $*$ als eine Menge von Abhängigkeiten, welche als (s-t) tgds und/oder egds formalisiert sind. Dabei Erzeugen (s-t) tgds neue Tupel, während egds Nullwerte ersetzen.

Erste Ansätze zur Erweiterung des *Standard*-Chase (siehe Definition 3.22) auf beliebige Objekte und Parameter (siehe Abbildung 9.1) finden sich in [AH19]. Dabei besteht der Chase-Parameter $*$ stets aus Integritätsbedingungen, Sichten, Anfragen oder anderen Abhängigkeiten, welche als Menge von (s-t) tgds und/oder egds dargestellt werden können. Das Chase-Objekt $\bigcirc$ hingegen ist entweder eine Anfrage Q oder eine Datenbankinstanz I. In beiden Fällen können Variablen/Nullwerte durch andere Variablen/Nullwerte bzw. Konstanten ersetzt werden. Die Regeln der Variablensubstitution hängen dabei von bestimmten Bedingungen ab.

Chase-Parameter Der Chase-Parameter $*$ besteht aus einer Menge von Abhängigkeiten Σ in Form von egds und/oder (s-t) tgds. Dabei handelt es sich um Verallgemeinerungen der klassischen *funktionalen Abhängigkeiten* (FDs) und *Verbund-Abhängigkeiten* (JDs). Jede Bedingung, die als eine Menge von (s-t) tgds und/oder egds geschrieben werden kann, kann als Chase-Parameter verwendet werden. Dazu gehören unter anderem Sichten (siehe Fall **II** in Tabelle 9.2), Operationen (Fall **II'**), Integritätsbedingungen (Fall **III-V**) und Anfragen (Fall **VI**), wie in Abbildung 9.1a zu sehen.



(a) Abhängigkeits-Hierarchie und Formalisierung des Chase-Parameters als Menge dieser Abhängigkeiten



(b) Variablen-Hierarchie und Formalisierung des Chase-Objekts als Menge dieser Variablen mit (nicht-)ausgezeichneten Variablen b_i und a_i, Nullwerten η_i und Konstanten c_i

Abbildung 9.1. Hierarchien von Chase-Objekt und Chase-Parameter.

Chase-Objekt Klassische Chase-Systeme wie PDQ [BLT14; BLT15], Graal [Bag+15] oder Llunatic [Gee+20] verarbeiten den Chase stets auf Instanzen (Llunatic) oder Anfragen (PDQ und Graal). Dabei sind Instanzen und Anfragen gar nicht so verschieden, denn das Datenbanktupel $\texttt{Student}(3, \text{'Max'}, \text{'Math'})$ und das Atom $\texttt{Student}(y_{\text{id}}, x_{\text{name}}, \text{'Math'})$ der Anfrage $Q = \exists y_{\text{id}} : \texttt{Student}(y_{id}, x_{\text{name}}, \text{'Math'}) \rightarrow (x_{\text{name}})$ haben eine sehr ähnliche Struktur. Ein Chase-Objekt ist eine Abstraktion von beiden. Das Tupel besteht aus Konstanten (c_i) und Nullwerten (η_1), während der Ausdruck (implizit) $\forall$-quantifizierte Variablen (x_i), $\exists$-quantifizierte Variablen (y_i) und Konstanten (c_i) enthält (siehe Abbildung 9.1b).

Formal bedeutet dies: Eine *Datenbankinstanz* I über einem Schema R besteht aus endlichen Relationen $R_1^I, ..., R_k^I$ (siehe Abschnitt 3.2), wobei jede Relation R_i^I die gleiche Arität wie das Relationssymbol R_i hat. Jedes Tupel $(x_1, ..., x_n)$ in R_i^I besteht aus Konstanten c_i oder Nullwerten η_i. Eine *konjunktive Anfrage* hingegen ist eine Formel erster Ordnung der Form $\exists y : \phi(x, y) \rightarrow \psi(x)$, wobei $\phi(x, y)$ eine Konjunktion von logischen Atomen (*Body*) und $\psi(x)$ ein einzelnes Atom (*Head*) ist. Die *Terme* in $\phi(x, y)$ sind entweder $\forall$-quantifizierte oder $\exists$-quantifizierte Variablen oder Konstanten. Zudem darf der Kopf $\psi(x)$ im Falle von Anfragen keine $\exists$-Variablen enthalten.

Eine Anfrage Q kann in eine *frozen instance* I_Q transformiert werden, indem jedes Atom des Bodies von Q als Tupel in I_Q dargestellt wird [DNR08]. Dabei gibt es verschiedene Varianten mit den Variablen in Q umzugehen. Oft werden $\exists$-Variablen in Nullwerte umgewandelt und $\forall$-Variablen werden als *markierte Nullwerte* oder *spezielle Konstanten* behandelt. Für die Transformation einer konjunktiven Anfrage Q in eine verallgemeinerte Instanz müssen die Atome im Body der Anfrage also als verallgemeinerte Tupel geschrieben werden. Das Kopf-Atom, welches den $\forall$-Variablen im Body sowie möglichen Konstanten entsprechen, können dabei vernachlässigt werden. Für den Body der obigen Anfrage Q erzeugen wir so ein zugehöriges Tupel $t \in I_Q$ wie folgt:

$$\begin{aligned} Q &= \exists y_{\text{id}} : \texttt{Student}(y_{\text{id}}, x_{\text{name}}, \text{'Math'}) \rightarrow (x_{\text{name}}) \\ I_Q &= \{\underbrace{\texttt{Student}(y_{\text{id}}, x_{\text{name}}, \text{'Math'})}_{t}\}. \end{aligned}$$

Diese verallgemeinerten Tupel können zu einer verallgemeinerten Instanz erweitert werden, wie in Definition 9.1 beschrieben. Während ChaTEAU mit allgemeinen s-t tgds umgehen kann, konzentrieren wir uns hier auf Anfragen, welche als s-t tgds mit einem einzigen Atom im Kopf betrachtet werden können.

Definition 9.1 (verallgemeinerte Instanz)**.** Sei I eine Instanz und Q eine konjunktive Anfrage. Eine *verallgemeinerte Instanz* ist entweder:

- eine Menge von (konventionellen) Relationen $R_1^I, ..., R_k^I$, d.h., deren Tupel aus Konstanten und Nullwerten bestehen, oder
- eine Menge von verallgemeinerten Tupeln, bestehend aus den Atomen von $\phi(x, y)$, mit Konstanten, $\forall$-Variablen und $\exists$-Variablen. ■

Algorithm 5 Chase für generalisierte Instanzen (Σ, I_{O_0})

Require: Menge von Abhängigkeiten Σ, generalisierte Datenbankinstnaz I_{O_0}
Ensure: modifizierte generalisierte Datenbankinstanz I_{O_n}

```
while I_{O_j} ≠ ⊥ and I_{O_{j-1}} ≠ I_{O_j} do
  for all Trigger h für σ ∈ Σ do
    if h ist aktiver Trigger then
      if σ ist eine tgd then
        erweiter h und füge neue Tupel zur Instanz I_{O_j} hinzu
      else if σ ist eine egd then
        if die verglichenen Werte unterschiedliche Konstanten sind then
          I_{O_{j+1}} = ⊥
        else
          ersetze Nullwerte und Variablen durch andere Nullwerte, Variablen oder Konstanten
```

Chase für verallgemeinerte Instanzen Wegen der unterschiedlichen Arten von Chase-Objekten O_i müssen die Chase-Schritte $I_{O_i} \rightarrow I_{O_{i+1}}$ auch bei der Verwendung von (s-t) tgds und egds verallgemeinert werden. Daher erweitern wir in ChaTEAU den Standard-Chase [Ben+17] zu einem Chase *für verallgemeinerte Instanzen* (siehe Algorithmus 5).

Die Hauptaufgabe des Chase besteht darin, neue Fakten abzuleiten. Zu diesem Zweck müssen wir strukturerhaltende Abbildungen (*Homomorphismen*, siehe Definition 3.20) zwischen den Abhängigkeiten Σ und dem Chase-Objekt O finden. Mit diesen bildet der Chase eine Menge von Abhängigkeiten Σ auf O ab. Das Ergebnis ist ein modifiziertes Chase-Objekt O'. Zwischen O und O' haben wir auch einen Homomorphismus.

Definition 9.2 (Substitutionsregeln für verallgemeinerte Instanzen)**.** Sei $\phi(\mathbf{x})$ der Body und $\psi(\mathbf{x}, \mathbf{y})$ der Kopf einer Abhängigkeit. Seien I_{O_1} und I_{O_2} verallgemeinerte Instanzen. Wir definieren die folgenden möglichen Substitutionsregeln:

1. eine Konstante kann nur auf sich selbst abgebildet werden:

$$c_i \mapsto c_i$$

2. ein Nullwert kann auf eine Konstante, sich selbst oder einen anderen Nullwert abgebildet werden:

$$\eta_i \mapsto c_j \mid \eta_i \mid \eta_j$$

3. eine $\exists$-Variable kann auf eine Konstante, sich selbst, einen Nullwert oder eine andere $\exists$- oder $\forall$-Variable abgebildet werden:

$$(a)\ y_i \mapsto c_j \mid \eta_j \mid y_i \mid x_j \qquad (b)\ y_i \mapsto c_j \mid y_i \mid y_j \mid x_j$$

4. eine $\forall$-Variable kann auf eine Konstante, einen Nullwert, sich selbst oder eine andere $\forall$- oder $\exists$-Variable abgebildet werden:

$$(a)\ x_i \mapsto c_j \mid \eta_j \mid x_i \mid y_j \qquad (b)\ x_i \mapsto c_j \mid x_i.$$

Insgesamt erhalten wir drei verschiedene Typen von Homomorphismen: Ein *Homomorphismus* $h : \phi(\mathbf{x}) \rightarrow I_{O_1}$ ist eine Abbildung, die (1) und (4a) erfüllt; ein *Homomorphismus* $h : \psi(\mathbf{x}, \mathbf{y}) \rightarrow I_{O_2}$ ist eine Abbildung, die (1), (3a) und (4a) erfüllt; ein *Homomorphismus* $h : I_{O_1} \rightarrow I_{O_2}$ ist eine Abbildung, die (1), (2), (3b) und (4b) erfüllt. ■

Tgds fügen neue Tupel in Instanzen ein oder Atome zu Anfragebodies hinzu. Diese Änderungen werden durch Anwendung eines bestimmten Homomorphismus, genannt *Trigger*, vom tgd-Body auf die generalisierte Instanz erzeugt. Egds hingegen setzen Variablen gleich. Weiter gilt: Ein *aktiver Trigger* ist einer, der (1) neue Tupel oder Ausdrücke durch Anwendung einer (s-t) tgd erzeugt oder (2) zu einer neuen Gleichung von Variablen oder Nullwerten durch Anwendung einer egd führt.

Definition 9.3 (Trigger für verallgemeinerte Instanzen)**.** Ein *Trigger* ist ein Homomorphismus h von einem Body $\phi(x)$ einer Abhängigkeit in eine verallgemeinerte Instanz I_O, d.h. $h : \phi(\mathbf{x}) \rightarrow I_O$. Wir nennen diesen Trigger *aktiv*, wenn folgendes gilt:

(1) tgd: es gibt keine Erweiterung von h zu einem Homomorphismus $\psi(\mathbf{x}, \mathbf{y}) \rightarrow I_O$;

(2) egd: es gilt $h(x_1) \neq h(x_2)$. ■

Ein Trigger ist somit immer dann aktiv, wenn er neue Tupel erzeugt oder Variablen bzw. Nullwerte gleichsetzt. Dies gilt insbesondere dann, wenn im Kopf der (s-t) tgd eine $\exists$-Variable existiert.

Wir erweitern den Standard-Chase auf verallgemeinerte Instanzen. Diese neue Version des Chase modifiziert eine verallgemeinerte Instanz durch eine Folge von Chase-Schritten, bis alle Abhängigkeiten erfüllt sind.

Definition 9.4 (Chase-Schritt für verallgemeinerte Instanzen)**.** Sei $h : \phi(\mathbf{x}) \rightarrow I_{O_i}$ ein aktiver Trigger für eine Abhängigkeit σ und eine generalisierte Instanz I_{O_i}. Die Modifikation von I_{O_i} zu $I_{O_{i+1}}$ durch Anwendung von σ unter h wird Chase-*Schritt* genannt. ■

Definition 9.5 (Chase für verallgemeinerte Instanzen)**.** Sei Σ eine Menge von Abhängigkeiten und I_{O_0} eine verallgemeinerte Instanz. Der (endliche) Chase *für verallgemeinerte Instanzen* ist eine endliche Folge von Chase Schritten $I_{O_i} I_{O_{i+1}}$ $(0 \leq i \leq n)$ mit

- $I_{O_n} = \bot$ (Chase schlägt fehl), oder
- $I_{O_n} = I_{O_{n+1}}$, d.h. es existiert ein Homomorphismus $h : I_{O_n} \to I_{O_{n+1}}$ mit $h(z_j) = z_j$ und $z_j \in \{c_j, \eta_j, x_j, y_j\}$.

■

Schließlich müssen wir das mit dem Chase auf verallgemeinerte Instanzen berechnete Ergebnis interpretieren. Bei Anwendung einer egd scheitert der Chase auf Instanzen, wenn verschiedene Konstanten aufeinander abgebildet werden sollen, und liefert $\bot$, während der Chase auf Anfragen die leere (weil unerfüllbare) Anfrage $\emptyset$ liefert. Das Chase-Ergebnis auf Q entspricht der Transformation von I_{Q_n} in eine neue Anfrage Q'. Dazu bilden die Tupel von I_{Q_n} eine Konjunktion von Atomen im Body von Q'. Der Anfragekopf wird durch Anwendung der Komposition aller Homomorphismen gebildet, die während der Chase-Ausführung gesammelt wurden. Somit funktioniert der in ChaTEAU implementierte Chase auf beliebigen s-t tgds.

9.2. Ablauf und Demonstration von ChaTEAU *(basierend auf [Aug+22])*

Schauen wir uns als nächstes den genauen Ablauf von ChaTEAU an. Dieser sowie die Demonstration des Systems basieren ebenfalls auf dem Demo-Artikel [Aug+22]. Die Implementierung von ChaTEAU sowie weitere Beispiele sind im zugehörigen GitLab-Repository zu finden.

Ablauf von ChaTEAU ChaTEAU ist eine universelle Implementierung des Chase, welche Instanzen und Anfragen zu einem allgemeines Chase-Objekt sowie Integritätsbedingungen, Schemaabbildungen, Sicht-Definitionen etc. zu einem allgemeinen Chase-Parameter abstrahiert. Wie in Abbildung 9.2 dargestellt, verarbeitet ChaTEAU den Chase-Parameter sowie das Chase-Objekt — definiert in einer speziellen XML-Datei — mit Hilfe des Chase und speichert das wiederum in einer XML-Datei. Für die Garantie eines reibungslosen Durchlaufs hat ChaTEAU zudem verschiedene Terminierungs- und Constraint-Tests integriert. Die Zahl der Tests soll in Zukunft aber noch erweitert werden.

ChaTEAU läuft auf verschiedenen Arten von Parametern und Objekten. Das Chase-Objekt ◯ wird von ChaTEAU automatisch erkannt und entsprechend verarbeitet. Für den Chase-Parameter $*$ ist eine solche Unterscheidung nicht notwendig. Er besteht aus einer Menge von Abhängigkeiten, dargestellt als (s-t) tgds und/oder egds. Dies ermöglicht es uns unter anderem Anfragen und Sichten sowie Inklusionsabhängigkeiten als Parameter zu verarbeiten. Das Chase-Objekt hingegen entspricht entweder einer Datenbankinstanz, einer Anfrage oder einer allgemeinen s-t tgd, wobei eine Anfrage als spezielle s-t tgd aufgefasst werden kann.

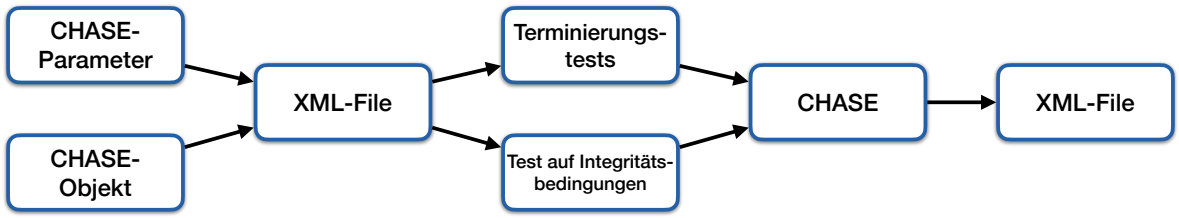


Abbildung 9.2. Konzeptueller Ablauf von ChaTEAU.

Eingabe und Ausgabe Die Ein- und Ausgabe von Instanzen, Anfragen und Abhängigkeiten in ChaTEAU erfolgt über spezielle XML-Dateien (siehe Anhang D). Die Eingabe-Datei definiert die Schemata, das Chase-Objekt sowie den Chase-Parameter. Hierfür werden zunächst die Relationenschemata definiert und mit einem Tag (S für *source* und T für *target*) versehen. Anschließend wird der Chase-Parameter definiert: Jede Abhängigkeit besteht aus einem Body sowie einem Head. Die Atome des Bodies dürfen dabei nur aus $\forall$-quantifizierten Variablen,

sogenannten *A-Variablen*, oder Konstanten besehen. Im Head sind zudem ∃-quantifizierte Variablen, sogenannte *E-Variablen*, erlaubt. In der abschließenden Definition des Chase-Objekts sind je nach Anwendung entweder Konstanten und Nullwerte oder eine Anfrage bestehend aus einem Body und einem Head erlaubt. Eine Beispiel-XML ist im Anhang als XML-Datei D.1 zu finden. Die Ausgabe-Datei besteht wiederum aus dem Chase-Objekt sowie den hierin vorkommenden Schemata. Der Chase-Parameter wird nicht erneut ausgegeben. Eine entsprechende Beispiel-Datei D.2 ist im Anhang D zu finden.

Terminierung Das Einfügen von Tupeln, welche Nullwerte — im Falle von Instanzen als Chase-Objekt — oder neue ∃-quantifizierte Variablen — im Falle von Anfragen als Chase-Objekt — enthalten, kann zu einer Endlosschleife führen, sodass der Chase bzw. dessen Implementierung nicht terminiert. Dies geschieht immer dann, wenn sich zwei tgds gegenseitig befeuern und bei jeder Verwendung neue Nullwerte erzeugt werden wie beispielsweise bei folgender Abhängigkeitsmenge:

$$\begin{aligned} \texttt{Student}(\texttt{studnetID}, \texttt{name}, \texttt{studies}) &\rightarrow \exists \texttt{Grade} : \texttt{Grade}(\texttt{studentID}, \texttt{Grade}) \\ \texttt{Grade}(\texttt{studentID}, \texttt{grade}) &\rightarrow \exists \texttt{Name}, \texttt{Studies} : \texttt{Student}(\texttt{studentID}, \texttt{Name}, \texttt{Studies}). \end{aligned}$$

Bedingungen, welche das Vorhandensein eines Fixpunkts einer Chase-Sequenz garantieren, werden als *Terminierungskriterien* bezeichnet. Die bekanntesten Chase-Terminierungskriterien können in [GST11; GMS12] nachgelesen werden. Sie sind zudem in Abschnitt 3.3.4 kurz zusammengefasst. ChaTEAU implementiert insgesamt fünf verschiedene Kriterien: Die reiche Azyklizität, die schwache Azyklizität, Safety, Azyklizität sowie die Azyklizität mit egd-Rewriting.

Die Azyklizität ist eine sehr leistungsfähige Bedingung, welche auf einem Constraint Rewriting [GST11] basiert, erweiterbar und einfach zu implementieren ist. Das gebräuchlichste Kriterium ist der Test auf schwache Azyklizität. Diese ist beipsielsweise in Llunatic [Gee+20] oder Graal [Bag+15] implementiert. Sie kann ohne großen Aufwand zur reichen Azyklizität sowie Safety erweitert werden. Zusätzlich haben wir uns entschieden, Azyklizität mit egd-Rewriting zu implementieren, um die egds, welche in den meisten bekannten Terminierungskriterien ignoriert werden, ebenfalls behandeln zu können [GST11].

API ChaTEAU ist eine eigenständige Implementierung des Chase. Das System ist als Maven-Projekt implementiert und kann über seine zugehörige GUI aufgerufen werden (siehe Abschnitt 9.2.1). Darüber hinaus kann ChaTEAU über seine API angesprochen werden, sodass es als Baustein oder Bibliothek für die Entwicklung anderer Chase-basierter Anwendungen verwendet werden kann. Mögliche Anwendungsmöglichkeiten von ChaTEAU wie beispielsweise ProSA folgen in Abschnitt 9.3 sowie Kapitel 10. ChaTEAU kann zudem mit verschiedenen Flags aufgerufen werden, zusammengefasst in Tabelle 9.1.

Kennzeichen	Funktion	Beschreibung
-o	Ausgabedatei speichern	Speichert das Chase-Ergebnis in einer ChaTEAU-formatierten XML-Datei.
-c	Checks durchführen	Führt die Terminierungs- und Integritätsprüfungen aus.
-p	Provenance berechnen	Berechnet während der Chase-Ausführung die ***where***-, ***why***- und ***how***-Provenance. Nur nützlich in Kombination mit -o.
-s	Quell-Schema speichern	Übernimmt die Quell-Schemata in die Ergebnisdatei. Nur sinnvoll in Kombination mit -o.

Tabelle 9.1. Flags für die ChaTEAU-Ausführung.

Git-Repository Die Implementierung von ChaTEAU ist im zugehörigen GitLab-Repository unter

`https://git.informatik.uni-rostock.de/ta093/chateau-demo`

zu finden. Dort sind neben dem Quellcode und einer Installationsanleitung auch verschiedene Beispiele sowie eine ausführliche Dokumentation zu finden.

9.2.1. Demonstration der ChaTEAU-GUI

Die ChaTEAU-GUI (siehe Abbildung 9.3) besteht aus einem Fenster, unterteilt in vier Tabs: Start (1), Checks (2), Chase (3) und Log (4). Wir starten mit dem Öffnen einer speziellen XML-Datei, genannt *ChaTEAU-Datei*, im ersten Tab (a). Sie enthält die benötigten Relationenschemata, den Chase-Parameter sowie das Chase-Objekt. Dabei bestimmt das System intern, ob es sich bei dem Objekt um eine Anfrage oder eine Instanz handelt und passt die zugehörigen Tags (A und B) entsprechend an. Vor der Ausführung des Chase im dritten Tab (3) werden verschiedene Terminierungs- und Constraint-Tests durchgeführt (2). Die zu prüfenden Tests können vom Nutzer selbst ausgewählt werden. Sie werden anschließend in einer vorgegebenen Reihenfolge ausgeführt werden. Das entsprechende Protokoll ist im letzten Tab zu finden (4).

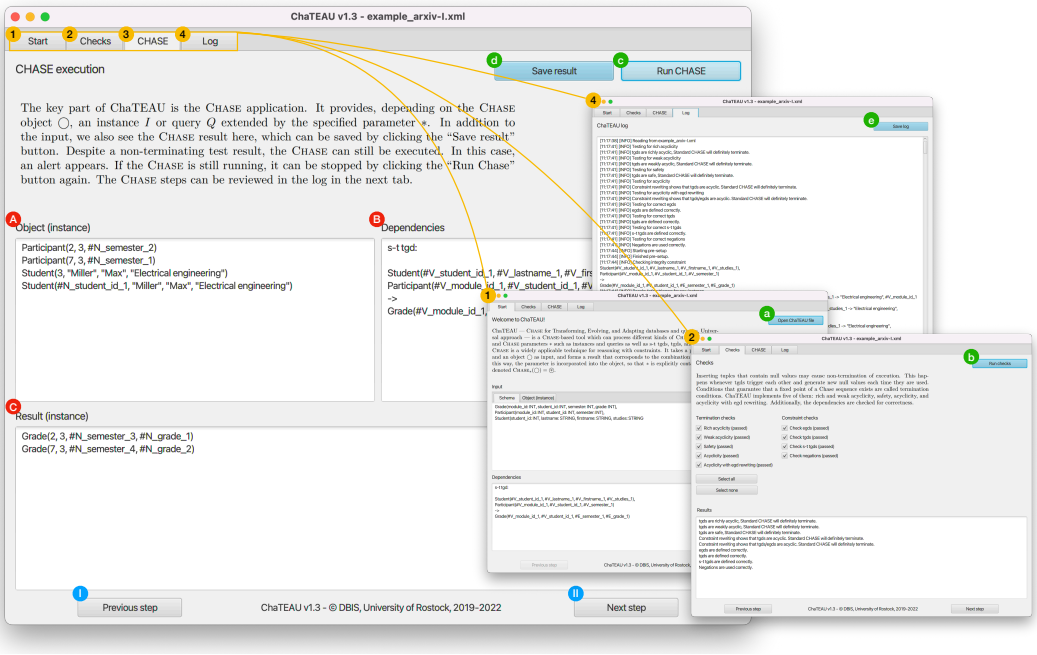


Abbildung 9.3. Überblick über die ChaTEAU-GUI

Die Navigation in ChaTEAU erfolgt über die beiden Buttons Previous Step (I) und Next Step (II) am unteren Rand des Fensters. In der oberen rechten Ecke eines jeden Tabs befindet sich eine Option zum Speichern des Chase-Ergebnisses sowie des Logs, die Möglichkeit zum Einlesen einer Eingabe-Datei sowie der Ausführung der Tests bzw. des Chase (a – e).

Obwohl der Chase nach der Verallgemeinerung von Instanzen und Anfragen auf ein allgemeines Chase-Objekt keine Unterscheidung der verschiedenen Objekte mehr vornimmt, betrachten wir an dieser Stelle beide Anwendungsfälle separat. Grund hierfür ist, dass in der Eingabe-XML trotz allgemeinem Chase-Objekt weiterhin zwischen Anfragen und Instanzen unterschieden wird.

Zur Visualisierung des Chase auf Instanzen wählen wir eine Anfrage Q_1, welche an eine gegebene Datenbankinstanz I gestellt wird. Konkret lassen wir uns die Prüfungsergebnisse aller Studierenden mit dem Namen *Max* ausgeben. Zur Visualisierung des Chase auf Anfragen hingegen, schreiben wir eine Anfrage Q_2 so um, dass eine gegebene Abhängigkeit erfüllt wird. Konkret integrieren wir eine egd, welche die Kursnamen zweier Studierenden gleich setzt. Beide Beispiele sind im Anhang D sowie im zugehörigen Git-Repository zu finden.

Start Die erste Aktion, welche wir in ChaTEAU durchführen, ist das Öffnen der ChaTEAU-Datei. Statt die Abhängigkeiten, Instanzen und/oder Anfragen manuell einzugeben, werden die Inhalte für das Chase-Objekt sowie den Chase-Parameter automatisch aus der ausgewählten XML-Datei heraus gelesen. Dabei passt sich zudem die Beschriftung von Input und Dependencies an das entsprechende Objekt an.

Sei zunächst I eine Datenbankinstanz (Chase-Objekt) bestehend aus jeweils zwei Tupeln einer `Student`- sowie einer `Participant`-Relation. Sei weiterhin Q_1 die Anfrage (Chase-Parameter), welche eine Notenliste aller Studierenden mit dem Namen *Max* erstellt. Diese ist als eine s-t tgd formalisiert, welche die bereits bekannten Attribute `studentID` und `courseID` übernimmt und (neue) Nullwerte für die neuen Attribute `semester` und `score` einführt. In ChaTEAU entspricht dies:

Instance:

$$\{\texttt{Participant}(2,3,4),\ \ \text{participant}(7,3,\#N\_\texttt{semester\_1}),$$
$$\texttt{Student}(3,\text{'Max'},\text{'Math'}),\ \ \text{student}(\#N\_\texttt{studentID\_1},\text{'Max'},\text{'Math'}),\ \ \texttt{Student}(7,\text{'Mia'},\#N\_\texttt{course\_1})\}$$

Dependencies:

$$\texttt{Participant}(\#V\_\texttt{course\_1},\#V\_\texttt{studentID\_1},\#V\_\texttt{semester\_1}),$$
$$\texttt{Student}(\#V\_\texttt{studentID\_1},\text{'Max'},\#V\_\texttt{course\_1})$$
$$->\ \texttt{Grade}(\#V\_\texttt{course\_1},\#V\_\texttt{studentID\_1},\#E\_\texttt{semester\_1},\#E\_\texttt{score\_1})$$

Betrachten wir als zweites eine Anfrage Q_2 (Objekt), welche die IDs, Namen und Kurse eines Studierenden ausgibt. Sei weiter Σ eine Menge von Abhängigkeiten (Parameter); hier eine egd. Diese ersetzt in Q_2 die $\exists$-quantifizierte Variable $\#E\_\texttt{course\_1}$ durch die $\forall$-quantifizierte Variable $\#V\_\texttt{course\_1}$ (blau hervorgehoben), indem sie die Attribute $\#V\_\texttt{course\_1}$ und $\#V\_\texttt{course\_2}$ gleichsetzt. Konkret bedeutet dies: Stimmen für zwei Tupel der Relation `Student` die IDs und Namen überein, so handelt es sich vermutlich um denselben Studierenden, weshalb auch die belegten Kurse übereinstimmen müssen. In ChaTEAU entspricht dies:

Query:

$$\texttt{Student}(\#V\_\texttt{studentID\_1},\#V\_\texttt{name\_1},\#E\_\texttt{course\_1}),\ \texttt{Student}(\#E\_\texttt{studentID\_1},\#V\_\texttt{name\_1},\#V\_\texttt{course\_2})$$
$$->\ (\#V\_\texttt{studentID\_1},\#V\_\texttt{name\_1},\#V\_\texttt{course\_1})$$

Dependencies:

$$\texttt{Student}(\#V\_\texttt{studentID\_1},\#V\_\texttt{name\_1},\#V\_\texttt{course\_1}),\ \texttt{Student}(\#V\_\texttt{studentID\_1},\#V\_\texttt{name\_1},\#V\_\texttt{course\_2})$$
$$->\ \#V\_\texttt{course\_1} = \#V\_\texttt{course\_2}$$

Terminierung In ChaTEAU stehen dem Nutzer fünf verschiedene Terminierungstests zur Verfügung. Die ausgewählten Tests werden in einer intern vorgegebenen Reihenfolge abgearbeitet, ausgeführt über den Button Run selected tests (b). Der Vorgang kann beliebig oft wiederholt werden (z.B. mit unterschiedlichen Terminierungskriterien). Um die Nicht-Terminierung des Chase zu vermeiden, ist vor jeder Chase-Ausführung die Durchführung von mindestens eines Terminierungskriteriums notwendig. Sollte die Terminierung durch mindestens eines der fünf Kriterien nicht gewährleistet werden, erscheint dem Nutzer eine Warnmeldung (siehe Abbildung 9.4). Diese kann vom Nutzer jedoch ignoriert werden.

In beiden Beispielen wird der Chase nach allen fünf Kriterien beendet. Auch die Constraint-Prüfung ist erfolgreich. Die Ausgabe in ChaTEAU sieht dabei wie folgt aus:

```
tgds are richly acyclic -> Standard Chase will definitely terminate.
tgds are weakly acyclic -> Standard Chase will definitely terminate.
tgds are safe -> Standard Chase will definitely terminate.
Constraint rewriting shows that tgds are acyclic
-> Chase will definitely terminate.
Constraint rewriting shows that tgds/egds are acyclic
-> Chase will definitely terminate.
Constraints are defined correctly.
```

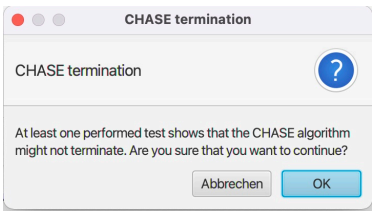


Abbildung 9.4. Warnmeldung in ChaTEAU bei Nicht-Terminierung des Chase

Chase-Ausführung Das Kernstück von ChaTEAU ist die Anwendung des Chase im dritten Tab. Neben der Eingabe (A und B) sehen wir hier auch das Chase-Ergebnis (C), welches neben der Ausgabe in der GUI als separates XML abgespeichert werden kann (d). ChaTEAU erlaubt es dem Nutzer, den Chase auch bei fehlgeschlagenem Terminierungstest auszuführen. In diesem Fall kann der Chase durch erneutes Klicken des Start-Buttons (c) jederzeit abgebrochen werden. Die einzelnen Chase-Schritte werden hier nicht ausgegeben. Sie können jedoch im Log (4) nachgesehen werden.

Unsere Ergebnisinstanz, welche durch Anwendung des Chase entsteht, entspricht dem Ergebnis der SQL-Anfrage `SELECT studentID, name, course FROM Participant NATURAL JOIN Student WHERE name = 'Max'` auf die oben definierte Instanz:

Ergebnis (Instanz):

$$\texttt{Grade}(7, 3, \#N\_\texttt{semester\_2}, \#N\_\texttt{score\_1}),\ \texttt{Grade}(2, 3, \#N\_\texttt{semester\_3}, \#N\_\texttt{score\_2})$$

Die Anwendung des Chase auf die Anfrage Q_2 liefert die oben beschriebene Ersetzung von $\#E\_\texttt{course\_1}$ durch die entsprechende $\forall$-quantifizierte Variable $\#V\_\texttt{course\_1}$ (blau hervorgehoben). Infolgedessen hat sich der Body geändert, während der Head derselbe bleibt:

Result (query):

$$\begin{aligned}&\texttt{Student}(\#V\_\texttt{studentID\_1}, \#V\_\texttt{name\_1}, \#V\_\texttt{course\_1}),\\ &\texttt{Student}(\#E\_\texttt{studentID\_2}, \#V\_\texttt{name\_1}, \#V\_\texttt{course\_2}))\\ &\quad -> \ (\#V\_\texttt{studentID\_1}, \#V\_\texttt{name\_1}, \#V\_\texttt{course\_1})\end{aligned}$$

Logging Die Ausführung des Chase beschränkt sich auf die ersten drei Tabs (1 – 3). Der Log im vierten Tab 4 liefert zusätzliche Informationen, die während der Chase-Ausführung gesammelt werden. Hierzu gehören neben dem Endergebnis auch die Zwischenergebnisse, welche nach Ausführung der einzelnen Chase-Schritte nach Anwendung einer (s-t) tgd oder egd entstehen. Auch Einzelheiten über die durchgeführten Constraint- und Terminierungstests sind im Log zu finden. Es ist somit insbesondere für die Fehleranalyse geeignet und kann mittels e abgespeichert werden.

9.2.2. Weitere Features von ChaTEAU *(noch nicht veröffentlicht)*

Neben der Implementierung des „klassischen“ Chase verallgemeinert auf ein allgemeines Chase-Objekt ◯ sowie einen allgemeinen Chase-Parameter $*$ auf einfachen konjunktiven Anfragen, bietet ChaTEAU zudem einige zusätzliche Operatoren. So können in ChaTEAU auch Konstanten- und Attribut-Attribut-Vergleiche im Head sowie im Body der Anfrage durchgeführt werden. Zudem ist auch die Negation eines Atoms möglich.

Negation Bei der Negation unterscheiden wir verschiedene Ansätze: die Negation einer Relation oder die Nicht-Gleichheit eines Attributwertes. Ersteres wird im Atom durch das Kennwort `negation="true"` gekennzeichnet. Ein entsprechendes Beispiel ist in der XML-Datei 9.1 zu finden. Die Nicht-Gleichheit hingegen wird über den Vergleich `comparison operator="neq"` dargestellt.

Anfrage 9.1 Beispiel für Negation in ChaTEAU

```
<atom name="Students" negation="true">
  <variable name="student_id" type="V" index="1" />
  <variable name="lastname" type="V" index="1" />
  <variable name="firstname" type="V" index="1" />
  <variable name="course" type="V" index="1" />
</atom>
```

Vergleiche Für die Realisierung von Vergleichen — sowohl Attribut-Attribut-Vergleiche als auch Attribut-Konstanten-Vergleiche — bietet ChaTEAU neben Atomen auch sogenannte *Comparisons* an. Diese können je nach Bedarf im Head oder im Body einer (s-t) tgd stehen. Möglich sind dabei die Operatoren `geq` ($\geq$), `gt` ($>$), `leq` ($\leq$), `lt` ($<$), `eq` ($=$) und `neq` ($\neq$). Um die hierfür notwendige Ordnung zu definieren, unterscheiden wir zwischen einem Teil links sowie einem Teil rechts des Operators. Beide Teile können dabei Variablen `variable`, Konstanten `number` oder Funktionen `function` enthalten. Auch eine Verschachtelung der Vergleiche ist möglich. Ein Beispiel für einen Attribut-Konstanten-Vergleich ist in der XML-Datei 9.2 zu finden, welche den Vergleich `studentID` > 1 darstellt.

XML-Datei 9.2 Beispiel für Vergleiche in ChaTEAU

```
<comparisons>
  <comparison operator="gt">
    <left>
      <variable name="studentID" type="V" index="1" />
    </left>
    <right>
      <number value="1" />
    </right>
  </comparison>
</comparisons>
```

Funktionen ChaTEAU unterstützt einige spezielle Funktionen. Hierzu gehören neben der Addition und der Subtraktion von Variablen und Konstanten auch die (skalare) Multiplikation und (skalare) Division angezeigt durch die Operatoren `ADDITION`, `SUBTRACTION`, `MULTIPLICATION` und `DIVISION`. Diese kommen in ProSA insbesondere bei der Darstellung der Aggregatfunktionen zum Einsatz. Ein Beispiel für die Addition ist beispielsweise in der XML-Datei 9.3 zu finden.

XML-Datei 9.3 Beispiel für Funktionen in ChaTEAU

```
<comparisons>
  <comparison operator="eq">
    <left>
      <variable name="c" type="V" index="1" />
    </left>
    <right>
      <function operator="addition">
        <left>
          <variable name="x" type="V" index="1" />
        </left>
        <right>
          <number value="1" />
        </right>
      </function>
    </right>
  </comparison>
</comparisons>
```

Aggregatfunktionen Für die Darstellung der Aggregatfunktionen, welche wir in Abschnitt 10.4 noch im Detail vorstellen werden, benötigen wir unter anderem Vergleiche, Funktionen sowie Negation. Je nach Aggregatfunktion sind dies die Funktionsoperatoren `ADDITION`, `SUBSTRACTION` und `DIVISION` sowie die Vergleichsoperatoren `gt` und `lt`, welche sowohl im Body als auch im Head der einzelnen tgds verwendet werden. Die abschließende s-t tgd sortiert dann durch „Relationen“-Negation alle überflüssigen Tupel aus. Ein Beispiel hierfür kann in Anhang D eingesehen werden.

Benchmark-Anfragen In Abschnitt 4.1 haben wir verschiedene Chase-Systeme anhand von sieben Benchmark-Anfragen (siehe Anhang B) auf ihre Anwendungsmöglichkeiten getestet. Dieselben Tests haben wir auch in ChaTEAU durchgeführt. Da ChaTEAU „lediglich“ eine Implementierung des verallgemeinerten Chase ist und somit keine Backchase-Phase enthält, können die Anfragen QB_1 bis QB_4 nicht getestet werden. Die Anfragen QA_1 bis QA_3 werden jedoch fehlerfrei optimiert. Die entsprechenden XML-Dateien D.3 bis D.5 können im Anhang D nachgelesen werden.

9.3. Anwendungen von ChaTEAU

Der Chase ist ein Universalwerkzeug, welches zur Lösung vieler grundlegender Probleme der Datenbanktheorie eingesetzt werden kann. Hierzu gehören unter anderem die Semantische Anfrageoptimierung, Answering Queries using Views (AQuV), Data Exchange und Data Integration, Data Cleaning sowie die Invertierung von Anfrageauswertungen. All diese Fragestellungen können mit einer Version des Chase gelöst werden. Aus diesem Grund wurden im Laufe der Zeit verschiedene Versionen des Algorithmus, wie etwa der oblivious, Standard- oder Core-Chase (siehe Abschnitt 3.3.3) sowie verschiedene Implementierungen des Chase entwickelt. Jedes System ist dabei speziell für einen Anwendungsbereich entwickelt (siehe Abschnitt 4.1).

	Parameter $\star$	**Objekt** $\bigcirc$	**Ergebnis** $\circledast$	**Ziel**
0.	Abhängigkeiten	DB-Schema	DB-Schema mit IBs integrity constraint	optimierter DB-Entwurf
I.	Abhängigkeiten	Anfrage	Anfrage	semantische Optimierung
II.	Sichten	Anfrage	Anfrage mit Sichten	AQuV
II'.	Operatoren	Anfrage	Anfrage mit Operatoren	AQuO
III.	s-t tgds, egds	Quell-DB	Ziel-DB	Data Exchange, Data Integration
IV.	tgds, egds	DB	modifizierte DB	Data Cleaning
V.	tgds, egds	unvollständige DB	Anfrageergebnis	Certain Answers
VI.	s-t tgds, egds, tgds	DB	Anfrageergebnis	invertierbare Auswertung

Tabelle 9.2. Chase-Varianten, Ausschnitt aus Tabelle 3.3.

ChaTEAU hingegen kann durch seine Chase-Implementierung basierend auf allgemeinen Chase-Objekten und Chase-Parametern vielseitig eingesetzt werden. Eine Unterscheidung zwischen konkreten Datenbankinstanzen oder Anfragen ist somit nicht mehr notwendig. Um den verallgemeinerten Chase auch über ChaTEAU hinaus nutzen zu können, kann das System über die mitgelieferte GUI eigenständig genutzt oder als Abhängigkeit in andere Chase-basierte Anwendungen eingebunden werden. Hierzu gehören etwa die in Tabelle 9.2 genannten Anwendungsmöglichkeiten. Auch am Lehrstuhl DBIS arbeiten wir zur Zeit an verschiedenen Fragestellungen (siehe Fall **I.**, **II.'** und **VI.** in Tabelle 9.2), welche mit Hilfe des in ChaTEAU implementierten Chase beantwortet werden können.

Semantische Anfrageoptimierung (Fall I.) Zu den ersten Chase-Anwendungen gehört das klassische Szenario der *semantischen Anfrageoptimierung*. Hierbei wird eine Menge von Integritätsbedingungen so in eine Anfrage eingearbeitet, dass diese unter den gegebenen Abhängigkeiten optimiert wird. Hierfür werden die vorliegenden Integritätsbedingungen (Chase-Parameter) zunächst in die Anfrage (Chase-Objekt) eingeChASEd und in einem zweiten Schritt, der sogenannten Backchase-Phase, so optimiert, dass die Anzahl der verwendeten Verbünde minimiert wird. Abschließend wird die Anfrage in eine der klassischen Anfragesprachen (SQL, Relationenalgebra mit Erweiterungen oder Datalog) zurückübersetzt.

Für die semantische Anfrageoptimierung ist somit wieder eine Variante des Chase&Backchase notwendig, wobei ChaTEAU sowohl für die Ausführung der Chase- als auch der Backchase-Phase genutzt werden kann. Um die aufwendigere Backchase-Phase effizient zu gestalten, können wir auch an dieser Stelle wieder auf bekannte Techniken wie Graphen [Mei14] oder Provenance-Informationen [DH13; Ile14] zurückgreifen.

Answering Queries using Operators (Fall II.') Im DBIS-Projekt PArADISE [GH16a; GH17; Gru22] verwenden die Autoren Techniken zum Query Rewriting und Query Containment, um eine effiziente und datenschutzkonforme Verarbeitung von Anfragen zu erreichen. Hierfür wird eine Anfrage in Fragmente und Restanfragen aufgeteilt, sodass die Daten von Fragmentanfragen auf ressourcenbeschränkten Geräten gefiltert und voraggregiert werden können. Restanfragen führen dann (nur) den letzten, komplexen Teil der ursprünglichen Anfrage auf leistungsfähigeren Geräten wie Cloud-Servern aus. Dadurch werden weniger Daten verarbeitet und an einen Cloud-Server weitergeleitet, sodass der Grundsatz der Datenminimierung erfüllt ist [GH16b; GH18].

Einige der verwendeten Umschreibetechniken basieren auf einer Technik zur Lösung des *Answering Queries using Views-Probem* (AQuV). Hierfür wird zunächst in der Chase-Phase eine Menge von Sichten (Chase-Parameter) in eine Anfrage (Chase-Objekt) eingearbeitet. Die Sichten werden dabei in Form von s-t tgds definiert. In einer zweiten Backchase-Phase werden die Unteranfragen der auf ihre Sichten eingeschränkten Anfrage (*universeller Plan*) auf Äquivalenz zur originalen Anfrage überprüft. ChaTEAU arbeitet hierfür die inversen Sicht-Definitionen (Chase-Parameter) in den universellen Plan (Chase-Objekt) ein.

Zur Lösung des AQuV-Problems erhalten wir also eine Variante des Chase&Backchase, dessen Phasen jeweils durch ChaTEAU ausgeführt werden können. Um die aufwendige Backchase-Phase effizienter zu gestalten, kann diese um Pruning oder ***why***-Provenance nach [DH13] ergänzt werden. Im Falle von AQuO werden die Operatoren in der Chase-Phase in die Anfrage eingearbeitet und die erweiterte Anfrage in der Backchase-Phase anschließend automatisch fragmentiert und vervollständigt. Eine Implementierung des AQuV-Problems mit Hilfe von ChaTEAU ist in [Tre22] zu finden. Details zur originalen Implementierung von $\text{Prov}_{\text{C\&B}}$ können in Abschnitt 4.1 nachgelesen werden.

Provenance management using Schema mappings with Annotations (Fall VI.) *ProSA* [Aug18; AH19; Aug20] fasst unsere Fragestellungen im Bereich des Forschungsdatenmanagements zusammen. Wir kombinieren den Chase mit Provenance-Informationen wie Data Provenance und weiteren Annotationen zur Berechnung einer (minimalen) Teil-Datenbank eines ursprünglichen Forschungsdatensatzes, die für die Reproduzierbarkeit und Replizierbarkeit der Forschungsergebnisse archiviert werden muss. Hierfür werten wir zunächst eine gegebene Auswertungsanfrage (Chase-Parameter) über einer Quell-Datenbank (Chase-Objekt) aus. Anschließend invertieren wir die Anfrage (Chase-Parameter) und werten diese auf dem Anfrageergebnis (Chase-Objekt) erneut aus. Wir erhalten so die in Kapitel 6 vorgestellte (minimale) Teil-Datenbank. Um die rekonstruierte Teil-Datenbank zu optimieren, verwenden wir auch hier zusätzlich Provenance-Informationen.

Praktische Anwendung findet die ProSA-Theorie nicht nur in einer eigenen Implementierung (siehe Abschnitt 10.9), sondern beispielsweise auch im *Hidden-Markov-Modell* [Aug17]. Basierend auf der Darstellung des Hidden-Markov-Modells in Form von SQL-Anweisungen [MH17], können wir auch hier Aussagen zum Thema Provenance und Invertierbarkeit treffen, ohne das Modell im Detail zu kennen. Wir benötigen lediglich Informationen über die zu Grunde liegenden Operationen. Hier die Addition und Subtraktion, skalare Multiplikation und Division sowie die Matrix-Vektor- und Matrix-Matrix-Multiplikation. Da sich für die Komposition der Operationen eine ergebnisäquivalente Chase-Inverse ergibt, müssen wir hier bei der Invertierung, d.h. der Berechnung der minimalen Teil-Datenbank, stets mit einem großen, aber dennoch „verkraftbaren" Informationsverlust rechnen. Die rekonstruierten Informationen genügen somit für die erneute Auswertung des Modells, nicht jedoch, um neue Auswertungen durchzuführen.

9.4. Zwischenfazit

ChaTEAU ist eine verallgemeinerte Implementierung des Chase, welche in vielen verschiedenen Bereichen Anwendung finden kann. Hierzu gehören unter anderem die semantische Optimierung, Data Exchange und Data Integration, Answering Queries using Views sowie das Data Cleaning. Dies ist möglich, da ChaTEAU nicht wie bisherige Implementierungen (siehe Abschnitt 4.1) auf Instanzen oder Anfragen, sondern auf einem allgemeinen Chase-Objekt $\bigcirc$ arbeitet. Die einzuarbeitenden Abhängigkeiten werden als Chase-Parameter $*$ verallgemeinert. Hierzu gehören neben den Inklusionsabhängigkeiten selbst auch Sichten, Anfragen oder andere Abhängigkeiten, welche als Menge von (s-t) tdgs und egds dargestellt werden können (siehe Abschnitt 9.1).

ChaTEAU kann eigenständig über seine mitgelieferte GUI zur Ausführung des Chase verwendet werden. Ist eine Interpretation oder Weiterverarbeitung des Chase-Ergebnis wie etwa bei einer Optimierung gewünscht, kann ChaTEAU über seine API angesprochen und in andere Chase-basierte Anwendungen integriert werden. In diesem Fall stehen dem Nutzer die vier Flags aus Tabelle 9.1 zur Spezifikation von ChaTEAU zur Verfügung. Eine dieser Anwendungen ist ProSA, welche wir uns im nächsten Kapitel genauer ansehen werden.

Eigenen sowie betreute Arbeiten

ChaTEAU ist Rahmen der vorliegenden Dissertation insbesondere in Kooperation mit verschiedenen studentischen Projekten und Abschlussarbeiten entstanden. Hierzu gehören:

- [Jur18] Martin Jurklies: Chase und Backchase – Entwicklung eines Universal-Werkzeugs für eine Basistechnik der Datenbankforschung. *Masterarbeit*, 2018
- [Ren19] Fabian Renn: Erweiterung des Chase-Werkzeugs ChaTEAU um Anfragetransformationen. *Bachelorarbeit*, 2019
- [AH19] **Tanja Auge**, Andreas Heuer: ProSA – Using the Chase for Provenance Management. *ADBIS*, 2019
- [Gör20] Andreas Görres: Erweiterung des Chase-Werkzeugs ChaTEAU um ein Terminierungskriterium. *Masterarbeit*, 2020
- [Zim20] Jakob Zimmer: Vereinheitlichung des Chase auf Instanzen und Anfragen am Beispiel ChaTEAU. *Bachelorarbeit*, 2020
- [Ros20] Florian Rose: Erweiterung des Chase-Werkzeugs ChaTEAU um eine Backchase-Phase. *Masterarbeit*, 2020
- [Alb+21] Michael Albus, Eduard Buch, Lukas Görtz, Moritz Hanzig, Eric Maier: Qualitätssicherung für ChaTEAU. *KSWS/Projekt*[1], SS2021
- [Aug+22] **Tanja Auge**, Nic Scharlau, Andreas Görres, Jakob Zimmer, Andreas Heuer: ChaTEAU — A Universal Toolkit for Applying the Chase. `https://arxiv.org/pdf/2206.01643.pdf`, 2022

Der Fokus der einzelnen Arbeiten lag hierbei stets auf der Entwicklung und Implementierung einer konkreten Fragestellung. Die Idee hinter ChaTEAU sowie eine zusammenfassende Erläuterung der zugrundeliegenden Theorie sind zudem in den beiden Veröffentlichungen zu finden.

[1] KSWS: *Komplexe Software Systeme*, Modul an der Universität Rostock im Bachelorstudiengang Informatik; Projekt: Modul an der Universität Rostock im Bachelorstudiengang Informatik

10. ProSA — Provenance Management using Schema Mappings with Annotations

Zur Garantie der Nachvollziehbarkeit eines Forschungsergebnisses im Sinne der Reproduzierbarkeit bzw. Replizierbarkeit und Plausibilität fordern viele Zeitschriften und Konferenzen immer häufiger die Herausgabe der zugehörigen Quell-Daten. In der Regel sind jedoch nicht alle in der Quell-Datenbank gespeicherten Daten für die konkrete Auswertung relevant. Stattdessen genügt meist ein minimaler und evtl. anonymisierter Teil der Quell-Datenbank. Zur Ermittlung dieses Datensatzes verwendet ProSA — sowohl das Projekt als auch die Implementierung — eine Version des Chase&Backchase. Zusätzliche Provenance-Informationen wie die Zeugenbasen der ***why***-Provenance sowie Side Tables ermöglichen es, so viele Informationen wie möglich zu rekonstruieren. Wir versuchen dabei jedoch, die rekonstruierte Menge der zu speichernden Daten aus Privacy-Aspekten so gering wie möglich zu halten (siehe Kapitel 2 und Kapitel 5).

Grundlage für die Anfrageauswertung in ProSA ist der Chase. Dieser wird jedoch nicht von ProSA selbst durchgeführt, sondern ist in ein zweites, separates System namens *ChaTEAU* (siehe Kapitel 9) ausgelagert. Im Verlauf der ProSA-Ausführung wird ChaTEAU insgesamt viermal aufgerufen: zweimal bei der Berechnung der (minimalen) Teil-Datenbank (siehe Abbildung 10.1, rot hervorgehoben) und zweimal bei der Integration der Evolution (blau hervorgehoben).

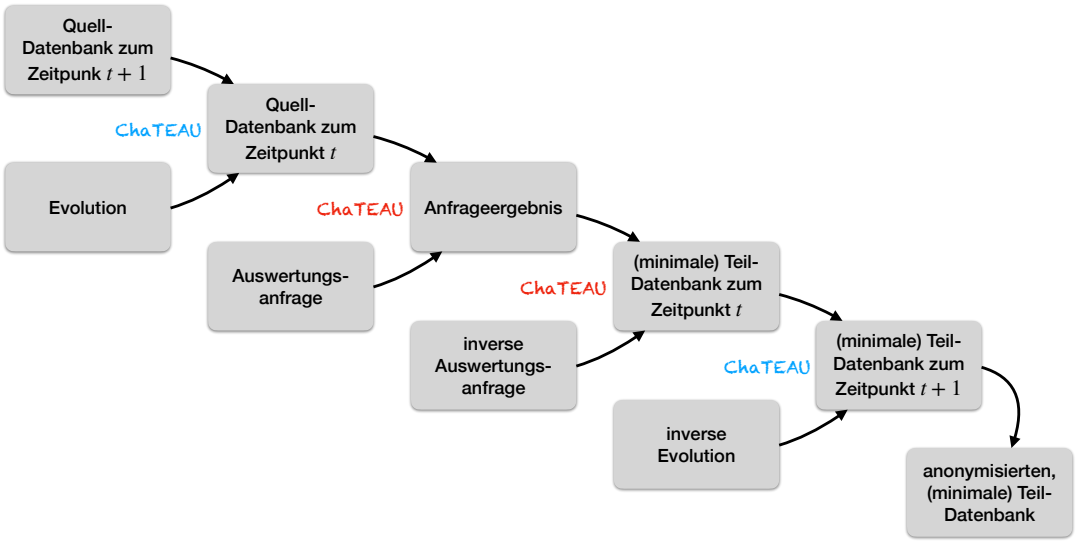


Abbildung 10.1. Einbindung von ChaTEAU in ProSA (Anteile der Evolution blau hervorgehoben; Auswertungsanfrage rot hervorgehoben).

Neben den konzeptuellen Beschreibungen für ChaTEAU in Kapitel 9 sowie für ProSA in Kapitel 10 verweisen wir für weitere Beispiele sowie den Quellcode auf die beiden zugehörigen Git-Repositories:

- ChaTEAU: `https://git.informatik.uni-rostock.de/ta093/chateau-demo`
- ProSA: `https://git.informatik.uni-rostock.de/ta093/prosa-demo`

Eine Liste aller relevanten Veröffentlichungen — hierzu gehören insbesondere [AH19; Aug20; AHH22] — sowie aller zugehörigen betreuten studentischen Abschlussarbeiten und Projekte ist am Ende des Kapitels zu finden. Zu nennen sind hier insbesondere [Kav22; Sch22; Spo22; Han22].

10.1. Konzeptueller Ablauf von ProSA *(basierend auf [AHH22])*

Bei der Bearbeitung von Forschungsprojekten fallen weltweit riesige Mengen an Forschungsdaten an. In der Regel werden jedoch nicht alle Daten für eine spätere Verarbeitung benötigt. In ProSA haben wir eine Technik entwickelt, um die für die Reproduktion bzw. Replikation eines (veröffentlichten) Forschungsergebnisses notwendigen Originaldaten zu bestimmen. Durch die Archivierung dieser Quell-Daten soll die Reproduzierbarkeit und Replizierbarkeit des Forschungseregbnisses garantiert werden.

Eine komplette Rekonstruktion der Forschungsdatenbank ist jedoch nicht in allen Fällen notwendig oder sinnvoll. So kann beispielsweise nicht jede Anfrage verlustfrei invertiert werden oder ein Informationsverlust wird aus Speicherplatzgründen bewusst in Kauf genommen. Es stellt sich daher die Frage, in welchen Fällen und in welchem Maße die Rekonstruktion der Quell-Daten erfolgen soll und welche zusätzlichen Annotationen gegebenenfalls benötigt werden. Hierfür nutzen wir neben Zeugenbasen (siehe Definition 3.37, [BKT01]) und Provenance-Polynomen (siehe Definition 3.38, [GT17]) noch zusätzliche Side Tables (siehe Definition 6.10). Details hierzu können in Abschnitt 6.4 nachgelesen werden.

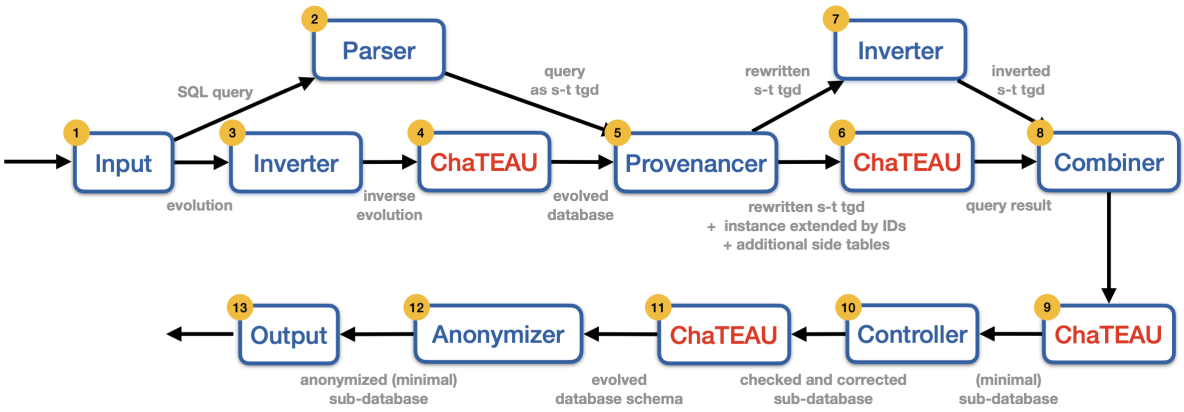


Abbildung 10.2. Konzeptueller Ablauf in ProSA.

Für die Bestimmung der (minimalen) Teil-Datenbank kombinieren wir in ProSA das Provenance Management mit dem Chase-Algorithmus. Der Ablauf von ProSA ist in Abbildung 10.2 dargestellt. Insgesamt werden dreizehn Schritte durchlaufen:

- Ein- und Ausgabe über die ProSA-GUI (Schritt **1** und **13**):
  - Eingabe: Anbindung der Datenbank, Eingabe der SQL-Anfrage, Upload der Evolutions-Dateien
  - Ausgabe: anonymisierten (minimalen) Teil-Datenbank
- Parsen der SQL-Anfrage als Menge von (s-t) tgds (Schritt **2**)
- Ausführen der inversen Evolution E (Schritte **3** und **4**)
- Auswerten der Anfrage inklusive zusätzlicher Provenance-Informationen $Q(I)$ (Schritte **5** und **6**)
- Berechnung der (minimalen) Teil-Datenbank I^* (Schritte **7** bis **10**)
- Evolution der (minimalen) Teil-Datenbank (Schritt **11**)
- Anonymisieren der Teil-Datenbank zu I^*_{anon} (Schritt **12**)

Den Mittelpunkt von ProSA bildet die Auswertung (Chase-Phase) sowie die Invertierung der Anfrage Q (Backchase-Phase) mit Hilfe des Chase-Algorithmus in **6** und **9**. Dieser wird an das Java-Tool ChaTEAU aus Kapitel 9 ausgelagert. Da ChaTEAU als Eingabe eine XML-Datei erwartet, welche eine Menge von Abhängigkeiten in Form von (s-t) tgd oder egd sowie eine Instanz bzw. Anfrage enthält, muss die SQL-Anfrage zunächst in eine Menge von (s-t) tgds transformiert und die Datenbankinstanz als sogenannte *Fakten* in die XML-Datei übernommen werden. Dies geschieht ebenso wie die Invertierung der SQL-Anfrage im ProSA-eigenen Parser **2**. Die Einbindung zusätzlicher Provenance-Informationen (**3** und **5**) ermöglicht die Maximierung der (minimalen) Teil-Datenbank. Die anschließende Generalisierung **12** garantiert die Einhaltung von Privacy-Aspekten. Zusätzlichen besitzt ProSA eine GUI, welche die Eingabe **1** sowie Ausgabe **13** unterstützt. Die folgenden Erläuterungen des Ablaufs basieren insbesondere auf [AHH22].

GUI (1 und 13) Die Eingabe — eine Datenbank-Instanz I, eine SQL-Abfrage Q und eine mögliche Evolution E — sowie die Ausgabe der anonymisierten (minimalen) Teil-Datenbank I^*_{anon} werden über eine GUI organisiert, welche die zentrale Schnittstelle zwischen dem Benutzer und ProSA bildet.

Die GUI (siehe Abbildung 10.8 bis 10.11) besteht aus sieben Registerkarten: Im Configuration-Tab wird eine Verbindung zur Datenbank hergestellt und eine mögliche Evolution in Form einer XML-Datei hinterlegt. Für eine bessere Nutzerfreundlichkeit wird dieser Tab bei einem Neustart stets übersprungen. Sowohl die hinterlegte Datenbank als auch die Evolutions-Datei können aber jederzeit verändert werden. Der zweite sowie der fünfte Tab zeigen die Ergebnisse der (inversen) Evolution E bzw. E^{-1}. Eine Interaktion mit der GUI ist an dieser Stelle nicht möglich. Die Berechnung der (minimalen) Teil-Datenbank I^* findet im dritten und vierten Tab statt. Hierfür wird zunächst eine SQL-Anfrage Q eingegeben, welche anschließend mit Hilfe des ChaseBackchase verarbeitet wird. Die Anonymisierung der Teil-Datenbank zu I^*_{anon} bildet den Abschluss von ProSA (Tab 6). Das Log speichert schließlich alle zusätzliche Informationen, welche während der einzelnen Schritte in ProSA sowie ChaTEAU angefallen sind.

Sei zur Veranschaulichung die Anfrage Q_1 gegeben, welche den Namen sowie die Note des Studierenden *Jack* wiedergibt. Sei weiter I ein Ausschnitt der in Anhang A.1 vorgestellten Universitätsdatenbank, definiert wie folgt:

studentID	name	
1	Stefan	s_{id_1}
3	Jack	s_{id_3}
7	Jack	s_{id_4}

Student

```
SELECT name, grade
FROM Student NATURAL JOIN Grade
WHERE name = "Jack";
```

studentID	grade	
1	1.0	$g_{\text{id}_{12}}$
3	1.3	$g_{\text{id}_{13}}$
7	1.7	$g_{\text{id}_{16}}$
3	1.7	$g_{\text{id}_{20}}$

Grade

Parser 2 ProSA verwendet für die Verarbeitung jeglicher Schemaabbildungen den Chase, welcher auf einer Menge Σ von Abhängigkeiten wie egds und (s-t) tgds arbeitet (siehe Abschnitt 3.3). Für eine bessere Benutzerfreundlichkeit werden Anfragen in ProSA jedoch in SQL gestellt. Daher besteht die Hauptaufgabe des Parsers darin, eine gegebene SQL-Anfrage in eine Menge von (s-t) tgds umzuwandeln. Egds sind für die Darstellung von konjunktive SPJU-Anfragen, Aggregatfunktionen wie `MAX`, `MIN`, `COUNT`, `SUM` und `AVG`, verschachtelte Anfragen sowie Gruppierungen nicht notwendig.

Für die Darstellung einfacher konjunktiver Anfragen wird in der Regel nur eine s-t tgd benötigt. So entspricht die SQL-Anfrage `SELECT` $\psi(x)$ `FROM` $\varphi(x)$ beispielsweise der s-t tgd $\forall x : \varphi(x) \rightarrow \psi(x)$[1], wobei $\psi(x)$ das Anfrageergebnis darstellt. Es gibt jedoch Anfragen, welche als Menge mehrerer (s-t) tgds definiert sind. Dazu gehören z.B. die Vereinigung, geschachtelte Anfragen oder die obigen Aggregatfunktionen. So entspricht die Anfrage (`SELECT` $\psi(x)$ `FROM` $\varphi_1(x)$) `UNION` (`SELECT` $\psi(x)$ `FROM` $\varphi_2(x)$) beispielsweise den beiden s-t tgds $\forall x : \varphi_1(x) \rightarrow \psi(x)$ und $\forall x : \varphi_2(x) \rightarrow \psi(x)$. Für die obigen Anfrage Q_1 ergibt sich so:

$$\texttt{Student}(\texttt{studentID}, \text{'Jack'}) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{grade}) \rightarrow \texttt{Result}(\text{'Jack'}, \texttt{grade}).$$

Für die Transformation wird die SQL-Anfrage auf ihre Schlüsselwörter untersucht und entsprechend geparst. Das Konzept hierzu wird in Abschnitt 10.2 vorgestellt. Details zur konkreten Umsetzung bzw. Implementierung können in [Kav22] nachgelesen werden.

Einbeziehen von Provenance 5 Der Provenancer bereitet die Einbindung von zusätzlichen Provenance-Informationen zur Generierung in der Chase-Phase vor. Hierfür fügt er global eindeutige Provenance-IDs zu den (s-t) tgds der Abhängigkeitsmenge Σ sowie zu den Quell-Tupeln aus I hinzu. Die um die Provenance-IDs erweiterte Abhängigkeitsmenge nennen wir Σ_P und die um IDs erweiterte Quell-Instanz I_P.

Für unser obiges Beispiel ergeben sich so

$$\Sigma = \{\texttt{Student}(\texttt{studentID}, \text{'Jack'}, s_{\text{id}}) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{grade}, g_{\text{id}}) \rightarrow \texttt{Result}(\text{'Jack'}, \texttt{grade}, s_{\text{id}}, g_{\text{id}})\}$$

[1] Der $\forall$-Quantor wird an dieser Stelle üblicherweise vernachlässigt.

sowie

$$I_P = \{\texttt{Student}(\eta_1, \text{'Jack'}, s_{\text{id}_3}), ...\}.$$

Für einige informationsverdichtende Anfragen ist die Erstellung zusätzlicher Side Tables notwendig, in welcher Zwischenergebnisse abgespeichert werden. In diesem Fall erhält auch die Side Table eine entsprechende Provenance-ID. Für die Frage nach der besten Abschlussnote ergibt sich beispielsweise

$$\begin{aligned} \texttt{Grade}(\texttt{studentID}_1, \texttt{grade}_1, s_{\text{id}_1}) \wedge \texttt{Grade}(\texttt{studentID}_2, \texttt{grade}_2, s_{\text{id}_2}) \wedge \texttt{grade}_1 > \texttt{grade}_2 & \\ \rightarrow \texttt{interim}(\texttt{grade}_1, s_{\texttt{studentID}_1}) & \quad \text{(tgd)}, \\ \texttt{Grade}(\texttt{studentID}, \texttt{grade}, s_{\text{id}}) \wedge \neg\texttt{interim}(\texttt{grade}, s_{\text{id}}) \rightarrow \texttt{Result}(\texttt{grade}, s_{\text{id}}) & \quad \text{(s-t tgd)}. \end{aligned}$$

Invertierung der s-t tgds (3 und 7) In diesem Modul wird eine geparste Anfrage Q invertiert. Die inverse Anfrage Q^{-1} entspricht immer einer s-t tgd, welche dann in der BACKCHASE-Phase verarbeitet wird. Für die Invertierung der Anfrage erweitern wir den Maximum-Extended-Recovery-Algorithmus von [Fag+11a], indem wir die invertierten Abhängigkeiten in eine minimale Anzahl von s-t tgds gruppieren. Im Fall von $Q^{-1}_{1,P}$ liefert der Inverter zum Beispiel:

$$\texttt{Result}(\text{'Jack'}, \texttt{grade}, s_{\text{id}}, g_{\text{id}}) \rightarrow \exists\texttt{studentID} : \texttt{Student}(\texttt{studentID}, \text{'Jack'}, s_{\text{id}}) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{grade}, g_{\text{id}}).$$

Gleiches gilt für die Evolution E. So lautet die Inverse der Evolution

$$\texttt{Grade}(\texttt{studentID}, \texttt{grade}) \rightarrow \exists\texttt{Semester} : \texttt{Grade}_{\texttt{new}}(\texttt{studentID}, \texttt{grade}, \texttt{Semester})$$

beispielsweise

$$\texttt{Grade}_{\texttt{new}}(\texttt{studentID}, \texttt{grade}, \texttt{semester}) \rightarrow \texttt{Grade}(\texttt{studentID}, \texttt{grade}).$$

Auch hier können Provenance-IDs (lila hervorgehoben) ergänzt werden.

Auswertung der Anfrage (ChaTEAU, Chase-Phase) 6 Durch Anwendung des CHASE ermittelt ChaTEAU das Anfrageergebnis $Q(I)$, indem es die Abhängigkeiten aus Σ_P in die Datenbankinstanz I_P einarbeitet[2], d.h. es gilt $Q(I) = \text{CHASE}_{\Sigma_P}(I)$ bzw. $I_{\Sigma_P} \models \Sigma_P$. Neben dem um die ***why***-Provenance erweiterten Anfrageergebnis liefert die CHASE-Phase auch die ***where***- und ***how***-Provenance sowie das Anfrageergebnis ohne zusätzliche Provenance-Informationen. Diese beiden Provenance-Typen dienen der Vollständigkeit. Für alle weiteren Berechnungen beschränken wir uns jedoch auf die ***why***-Provenance.

Für die obige Anfrage Q_1 ergibt sich die folgende Ergebnisrelation `Result`, wobei das Tupel $\texttt{Result}(\text{'Jack'}, 1.3) \in Q_{1,P}(I_P)$ intern als $\texttt{Result}(\text{'Jack'}, 1.3, s_{\text{id}_3}, g_{\text{id}_{13}})$ usw. gespeichert wird. Auf diese Weise werden Duplikate beibehalten, aber dem Benutzer nicht angezeigt.

name	grade	
Jack	1.3	$\{\{s_{\text{id}_3}, g_{\text{id}_{13}}\}\}$
Jack	1.7	$\{\{s_{\text{id}_3}, g_{\text{id}_{20}}\}, \{s_{\text{id}_4}, g_{\text{id}_{16}}\}\}$

Result

Vorbereitung der Backchase-Phase 8 Der Combiner kombiniert die Inverse Q_P^{-1} mit dem Anfrageergebnis $Q_P(I_P)$, um eine gültige Eingabe für die BACKCHASE-Phase zu erhalten.

[2] Zur Vereinfachung schreiben wir Q bzw. Σ anstelle der zugehörigen Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ und Q^{-1} bzw. Σ^{-1} anstelle der zugehörigen Schemaabbildung $\mathcal{M}^* = (T, S, \Sigma^{-1})$ sowie $Q(I)$ für das Ergebnis der CHASE-Anwendung von I und $\mathcal{M}$. Des weiteren bezeichnen Q_P bzw. Σ_P und I_P die jeweilige Provenance-Erweiterungen.

Berechnung der (minimalen) Teil-Datenbank (ChaTEAU, Backchase-Phase) 9 In unserem Kontext entspricht der Backchase dem Standard-Chase angewandt auf das Ergebnis der ersten Chase-Anwendung. Die Chase- und Backchase-Phase unterscheiden sich für uns also lediglich in ihrer Eingabe. Statt der Abhängigkeitsmenge Σ_P verarbeitet ChaTEAU in dieser Komponente die Abhängigkeitsmenge Σ_P^{-1} und statt der Instanz I_P die Instanz $Q_P(I_P)$. Für die (minimale) Teil-Datenbank gilt dann

$$I_P^* = \textsc{Chase}_{\Sigma_P^{-1}}(Q_P(I_P)) = \textsc{Chase}_{\Sigma_P^{-1}}(\textsc{Chase}_{\Sigma_P}(I_P)).$$[3]

Mit Hilfe der inversen Anfrage

$$Q^{-1} = \texttt{Result}(\text{'Jack'}, \texttt{grade}) \rightarrow \exists \texttt{studentID} : \texttt{Student}(\texttt{studentID}, \text{'Jack'}) \wedge \texttt{grade}(\texttt{studentID}, \texttt{grade})$$

können wir angewandt auf $Q(I)$ vier Tupel in I^* rekonstruieren: $\texttt{Student}(\eta_2, \text{'Jack'})$, $\texttt{Student}(\eta_1, \text{'Jack'})$, $\texttt{grade}(\eta_1, 1.3)$ und $\texttt{grade}(\eta_2, 1.7)$. Zusätzliche Provenance-Information ermöglichen zudem die Rekonstruktion weiterer Tupel.

Grund hierfür ist die Weiterverarbeitung der in der Chase-Phase bestimmten Provenance-Informationen. Mit ihrer Hilfe können verloren gegangene Attributwerte durch Nullwerte aufgefülllt und Duplikate rekonstruiert werden. Für unser obiges Beispiel ergeben sich für I^* die folgenden sechs Tupel:

studentID	name	
η_1	Jack	s_{id_3}
η_2	Jack	s_{id_7}
η_3	Jack	s_{id_3}

Student

studentID	grade	
η_1	1.3	$g_{\text{id}_{13}}$
η_2	1.7	$g_{\text{id}_{16}}$
η_3	1.7	$g_{\text{id}_{20}}$

grade

Überprüfen und Korrigieren der berechneten Teil-Datenbank 10 Der Controller prüft, ob die berechnete (minimale) Teil-Datenbank I^* bzgl. Q korrekt berechnet ist. Dies ist wichtig, da die Teil-Datenbank nur dann die Anforderungen aus Kapitel 2 bzw. Kapitel 5 erfüllt und zur Reproduzierbarkeit bzw. Replizierbarkeit oder zum Nachweise der Plausibilität von $Q(I)$ herangezogen werden kann. Dazu berechnet der Controller $Q(I^*)$ und vergleicht dies mit dem Anfrageergebnis $Q(I)$. Hierfür wird erneut der Chase verwendet. Schlägt die Prüfung fehl, d.h. es gilt nicht $Q(I^*) = Q(I)$ — das Anfrageergebnis $Q(I^*)$ ist nicht replizierbar bzw. reproduzierbar — oder zumindest $Q(I^*) \preccurlyeq Q(I)$ — das Anfrageergebnis $Q(I^*)$ ist nicht plausibel —, ist die (minimale) Teil-Datenbank falsch berechnet worden, und es wird eine Fehlermeldung zurückgegeben.

Darüber hinaus eliminiert der Controller redundante Tupel, welche in der Backchase-Phase fälschlicherweise entstanden sind. Dies passiert beispielsweise bei Duplikaten im Anfrageergebnis. Dies können wir jedoch beheben, indem wir die Schlüsseleigenschaften der Provenance-IDs ausnutzen. Im Falle der Anfrage Q_1 kann so der Nullwert η_3 global durch η_1 ersetzt werden. Das überflüssige Tupel $\texttt{Student}(\eta_3, \text{'Jack'})$ entfällt und das Tupel $\texttt{grade}(\eta_3, 1.7)$ zu $\texttt{grade}(\eta_1, 1.7)$ korrigiert. Die zugehörige egd lautet:

$$\texttt{Student}(\texttt{studentID}_1, \texttt{name}_1, s_{\text{id}}) \wedge \texttt{Student}(\texttt{studentID}_2, \texttt{name}_2, s_{\texttt{studentID}}) \rightarrow \texttt{studentID}_1 = \texttt{studentID}_2 \wedge \texttt{name}_1 = \texttt{name}_2.$$

Anonymisierer 12 Das bewusste Sammeln von zusätzlichen Informationen sowie die Einhaltung von Privacy-Aspekten stehen im ersten Moment im Widerspruch zueinander. Aufgabe des Anonymisierers ist es, die mit Hilfe von Provenance möglichst exakte Rekonstruktion der (minimalen) Teil-Datenbank so zu anonymisieren, dass Privacy-Aspekte eingehalten werden. Als Anonymisierungsmethode haben wir uns für Generalisierung, Intensionalisierung und Unterdrückung entschieden, als Anonymisierungsmaß haben wir die k-Anonymität gewählt. Dazu anonymisieren wir I^* durch Generalisierung mit Hilfe von zuvor definierten Domänengeneralisierungshierarchien (siehe Definition 3.51). Die Berechnung der möglichen Quasi-Identifikatoren erfolgt durch ProSA selbst.

[3] Zur Vereinfachung schreiben wir von nun an Q, I und Σ anstelle von Q_P, I_P sowie Σ_P.

Für die Anonymisierung in ProSA bieten sich verschiedene Zeitpunkte an. Wir haben uns für eine Anonymisierung nach Abschluss des Chase&Backchase-Verfahrens zur Berechnung der (minimalen) Teil-Datenbank entschieden. Um Konflikte, welche durch eine erneute Schemaänderung zustande kommen können, zu vermeiden, findet die Anonymisierung zudem nach der Evolution der (minimalen) Teil-Datenbank I^* zu J^* als letzter Schritt der ProSA-Pipeline statt. Details hierzu können in Abschnitt 10.7 nachgelesen werden.

Für Anfrage Q_1 bedeutet dies: Die Attribute `name` und `studentID` können nicht weiter generalisiert werden. Eine Generalisierung des Attributes `grade` ist jedoch möglich. So kann die Note 1.3 zu *sehr gut* und die Note 1.7 zu *gut* generalisiert werden. Details hierzu können ebenfalls in Abschnitt 10.7 sowie Abschnitt 8.3 nachgelesen werden. Insgesamt liefert dies die anonymisierte (minimale) Teil-Datenbank I^*_{anon}:

studentID	name	
η_1	Jack	s_{id_3}
η_2	Jack	s_{id_7}

Student

studentID	grade	
η_1	sehr gut	$g_{\text{id}_{13}}$
η_2	gut	$g_{\text{id}_{16}}$
η_1	gut	$g_{\text{id}_{20}}$

grade

Berechnung der (minimalen) Teil-Datenbank bei sich ändernden Schemata (ChaTEAU, Chase-Phase 4 und Backchase-Phase 11) Liegt neben der Anfrage Q sowie der Datenbankinstanz I noch eine Evolution E von S_t nach S_{t+1} in Form einer SMO vor, wird diese zunächst in eine Menge von (s-t) tgds transformiert und invertiert. Anschließend wir diese in einem weiteren Chase&Backchase verarbeitet. Die (minimale) Teil-Datenbank I^* über dem Schema S_t kann so in ein neues Schema S_{t+1} übertragen werden und wir erhalten I^* zum Zeitpunkt $t+1$. Zusätzliche Tupel, welche erst im Laufe der Evolution entstanden sind, können dabei vernachlässigt werden. Sie haben keinen Einfluss auf die Berechnung des Anfrageergebnisses $Q(I^*)$ bzw. $Q(I)$. Details hierzu sind in Abschnitt 7.1 zu finden.

10.2. Transformation einer SQL-Anfrage in eine Menge von (s-t) tgds

Für die Bestimmung der (minimalen) Teil-Datenbank kombiniert ProSA Techniken des Provenance Managements mit dem Chase. In unserem Fall übernimmt der Chase die Auswertung der Datenbank-Anfrage sowie ihrer Invertierung, während Provenance die Berechnung einer möglichst großen Teil-Datenbank unterstützt. Für die Chase-Berechnung nutzt ProSA das ebenfalls in Rostock entwickelte Java-Tool ChaTEAU (siehe Kapitel 9). Dieses verarbeitet als Eingabe eine XML-Datei; in ProSA werden Anfragen für eine höhere Benutzerfreundlichkeit jedoch in SQL gestellt. Die Transformation der SQL-Anfrage in eine Menge von (s-t) tgds und egds übernimmt ein Parser, dessen theoretische Grundlagen hier kurz vorgestellt werden sollen. Details zur praktischen Implementierung können in [Kav22; RSZ21] nachgelesen werden.

Wir beginnen mit einigen Vorüberlegungen, welche in Tabelle 10.1 zusammengefasst sind. Einfache SQL-Anfragen mit `SELECT`-, `FROM`- und `WHERE`-Klauseln können leicht in Relationenalgebra, in Datalog oder als Menge von (s-t) tgds (von der Quell-Instanz in eine neue Ergebnisinstanz `Result`) transformiert werden. So kann die `SELECT`-Klausel als Projektion verstanden werden, bei der die projizierten Attribute als Ergebnisattribute in den Head der (s-t) tgd übernommen werden. Die notwendigen Quell-Relationen können der `FROM`-Klausel entnommen werden. Sie bilden die Atome des (s-t) tgd-Bodies. Das Verbundattribut sowie einfache Konstanten-Selektionen auf $<$, $\leq$, $=$, $\neq$, $\geq$,$>$ sind in der `WHERE`-Klausel zu finden. Die Selektion sowie der Verbund können zudem über die SQL-Schlüsselwörter `NATURAL JOIN`, `JOIN ON` oder `AND` dargestellt werden. Sind mehrere Relationen an der Anfrage beteiligt, so werden diese stets als Komposition der entsprechenden Relationenschemata im Body der s-t tgds interpretiert. Fassen wir unsere Erkenntnisse zusammen, kann die „einfache“ SQL-Anfrage

```
SELECT a_i
FROM R NATURAL JOIN S
WHERE b_j = 13
```

wie folgt als s-t tgd aufgefasst werden:

$$R(a_1, ..., a_n) \wedge S(b_1, ..., b_m) \wedge a_i = b_i \wedge b_j = 13 \rightarrow \texttt{Result}(a_i).$$

SQL-Schlüsselwort	Übersetzung als s-t tgd
SELECT	Projektion, außer bei SELECT *
FROM	Relationenschemata der Quell-Relationen
WHERE	Selektion auf $<, \leq, =, \neq, \geq, >$ oder Darstellung eines Verbundes
NATURAL JOIN	Konjunktion über gleichnamige Attribute
JOIN ON	Konjunktion über zwei Attributen
AND	Konjunktion von Relationen (als alternative zu NATURAL JOIN) oder weitere Selektion auf $<, \leq, =, \neq, \geq, >$
AS	Konjunktion einer Relation mit der Bedingung für die Umbenennung
OR	zwei s-t tgds oder Disjunktion
UNION	zwei s-t tgds oder Disjunktion
MAX, MIN, COUNT, SUM, AVG	zusätzliche Funktionen und Side Tables notwendig
INTERSECT	Konjunktion von Relationen
EXCEPT	Negation einer Relation
geschachtelte Anfragen	abhängig von der Darstellung in Relationenalgebra $\Rightarrow$ darstellbar, sofern sie auf eine der Basis-Anfragen zurückgeführt werden kann
IN	Konjunktion von Relationen
NOT IN	Negation eines Attributes
ALL , DISTINCT	mengentheoretisch nicht darstellbar
ORDER BY	mengentheoretisch nicht darstellbar
GROUP BY	darstellbar über zusätzliches Gruppierungsattribut

Tabelle 10.1. Vorüberlegungen zur Übersetzung von SQL-Schlüsselwörtern als Menge von (s-t) tgds.

In den meisten Fällen kann eine SQL-Anfrage als einzelne s-t tgd aufgefasst werden. Dies gilt jedoch nicht für Aggregationen (siehe Abschnitt 10.4) sowie die Schlüsselwörter OR , UNION oder GROUP BY . In diesen Fällen sind mehrere (s-t) tgds zur Darstellung der SQL-Anfrage notwendig. Zur Korrektur fälschlich bestimmter Tupel, wie in Abschnitt 10.1 beschrieben, werden zudem zusätzliche Korrektur-egds benötigt. Die Mengenoperation INTERSECT sowie geschachtelten Anfragen werden, sofern letztere auf die Basis-Operatoren der Relationenanlgebra $\pi, \sigma, \times, \cup, -, \beta$ zurückgeführt werden können, ebenfalls als Konjunktion der beteiligten Relationenschemata im Body der s-t tgd dargestellt. Im Falle der SQL-Schlüsselwörter EXCEPT und NOT IN benötigen wir zudem die Negation auf Relationen- bzw. Attributebene im Body der (s-t) tgds. Vergleiche hierzu die Darstellungen in der Tabelle 10.2.

Da der Chase mengenbasiert arbeitet, sind Duplikate innerhalb der Ergebnisinstanz sowie eine Ordnung der (s-t) tgds nicht vorgesehen. Die Schlüsselwörter ALL , DISTINCT und ORDER BY werden daher nicht formalisiert. Für die Darstellung von Aggregatfunktionen sowie der Gruppierung behelfen wir uns mit zusätzlichen Provenance- und Gruppierungsattributen. Hierfür sei auf den Abschnitt 10.4 verwiesen.

Für den im Kontext von ProSA relevanten Anwendungsfall definieren wir neun Basis-Anfragen sowie deren Formalisierungen als Menge von s-t tgds, welche in Tabelle 10.2 zusammengefasst sind. Auch Kombinationen von SQL-Schlüsselwörtern bzw. Anfragen mit „vollständigen“ SFW-Blöcken lassen sich auf diese Basisanfragen (Selektion, Projektion, Verbund, Vereinigung, Schnitt, Differenz und Umbenennung) zurückführen. Zudem können viele komplexere Anfragen als geschachtelte Anfragen verstanden werden. So entspricht die SQL-Anfrage

```
SELECT studentID, MIN(grade) AS min_grade
FROM Grade
GROUP BY studentID
```

der (s-t) tgd-Menge

$$\begin{aligned} \texttt{Grade}(\texttt{studentID}, \texttt{grade}_1) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{grade}_2) \wedge \texttt{grade}_1 > \texttt{grade}_2 &\rightarrow H(\texttt{studentID}, \texttt{grade}_1) \\ \texttt{Grade}(\texttt{studentID}, \texttt{grade}) \wedge \neg H(\texttt{studentID}, \texttt{grade}) &\rightarrow \texttt{Result}(\texttt{studentID}, \texttt{grade}), \end{aligned}$$

wobei H einer Hilfstabelle zur Speicherung der Zwischenergebnisse entspricht. Details zur Darstellung von Aggregatfunktionen sowie der Gruppierung als Menge von s-t tgds können in Abschnitt 10.4 nachgelesen werden.

Operation	**Basisoperation in SQL**	**s-t tgd**
Projektion (Spezialfall)	`SELECT` a_i `FROM` R	$R(a_1, ..., a_n) \rightarrow \texttt{Result}(a_i)$
	`SELECT` $*$ `FROM` R	$R(a_1, ..., a_n) \rightarrow \texttt{Result}(a_1, ..., a_n)$
Duplikate	`SELECT DISTINCT` a_i `FROM` R	$R(a_1, ..., a_n) \rightarrow \texttt{Result}(a_i)$
Selektion	`SELECT` $*$ `FROM` R `WHERE` $a_k \theta c$	$R(a_1, ..., a_k, ..., a_n) \wedge a_k \theta c \rightarrow \texttt{Result}(a_1, ..., a_k, ..., a_n)$
	`SELECT` $*$ `FROM` R `WHERE` $a_i = a_j$ `AND` $a_k = c$	$R(a_1, ..., a_i, ..., a_j, ..., a_k, ..., a_n) \rightarrow \texttt{Result}(a_1, ..., a_i, ..., a_i, ..., c, ..., a_n)$
	`SELECT` $*$ `FROM` R `WHERE` $a_k \theta a_j$	$R(a_1, ..., a_k, ..., a_n) \wedge a_k \theta a_j \rightarrow \texttt{Result}(a_1, ..., a_k, ..., a_n)$
Verbund	`SELECT` $*$ `FROM` R `NATURAL JOIN` S	$R(a_1, ..., a_n, c) \wedge S(b_1, ..., b_m, c) \rightarrow \texttt{Result}(a_1, ..., a_n, c, b_1, ..., b_m)$
	`SELECT` $*$ `FROM` R, S, T `WHERE` $R.a_i = S.b_j$ `AND` $S.b_k = T.c_l$	$R(a_1, ..., a_n) \wedge S(b_1, ..., b_m) \wedge a_i = b_j \wedge T(c_1, ..., c_l) \wedge b_k = c_l \rightarrow \texttt{Result}(a_1, ..., a_n, b_1, ..., b_m, c_1, ..., c_l)$
Vereinigung	`SELECT` $*$ `FROM` R `UNION` S	$R(a_1, ..., a_n) \rightarrow \texttt{Result}(a_1, ..., a_n)$, $S(a_1, ..., a_n) \rightarrow \texttt{Result}(a_1, ..., a_n)$
Schnitt	`SELECT` $*$ `FROM` R `INTERSECT` S	$R(a_1, ..., a_n) \wedge S(a_1, ..., a_n) \rightarrow \texttt{Result}(a_1, ..., a_n)$
Differenz	`SELECT` $*$ `FROM` R `EXCEPT` S	$R(a_1, ..., a_n) \wedge \neg S(a_1, ..., a_n) \rightarrow \texttt{Result}(a_1, ..., a_n)$
Umbenennung	`SELECT` a_i, a_j `AS` x `FROM` R	$R(a_1, ..., a_i, a_j, ..., a_n) \wedge a_j = x \rightarrow \texttt{Result}(a_i, x)$
Geschachtelte Anfrage	`SELECT` $*$ `FROM` R `WHERE` a_i `IN` (`SELECT` a_j `FROM` S)	$R(a_1, ..., a_i, ..., a_n) \wedge S(b_1, ..., a_j, ..., b_m) \wedge a_i = a_j \rightarrow \texttt{Result}(a_1, ..., a_n)$
Aggregation	`SELECT AVG`(a_i) `FROM` R	siehe Abschnitt 10.4

Tabelle 10.2. Darstellung von SQL-Schlüsselwörtern als Menge von (s-t) tgds.

Basierend auf den Ergebnissen aus [RSZ21] lässt sich die Transformation im Parser (zunächst ohne Aggregatfunktionen) wie folgt zusammenfassen (siehe Algorithmus 6): Zunächst wird ein Graph erstellt, welcher die Variablen der Quell-Relationen als Knoten enthält. Der Verbund zweier Relationen wird durch eine Kante zwischen den entsprechenden Verbundattributen realisiert. Aus dem Graphen lassen sich nun alle Informationen für den Aufbau von Head und Body der s-t tgds ablesen. So enthält der Body alle im Baum enthaltenen Knoten als Variablen der komponierten Relationen, wobei alle Variablen innerhalb einer Zusammenhangskomponente gleichgesetzt werden. Das Ergebnisschema enthält die in der `SELECT`-Klausel definierten Attribute. Dieses bildet den Head der (s-t) tgds. Betrachten wir dies an einem konkreten Beispiel.

Algorithm 6 SQL to s-t tgd (Transformation)

Require: SQL-Anfrage Q
Ensure: Anfrage Q' als s-t tgd
1. Aufbau des Graphen:
   a) Knoten: Variablen der Quell-Relationen
   b) Kanten: Verbundattribute bestimmen und durch Kante verbinden
2. Zusammenhangskomponenten im Graphen bestimmen
3. Body:
   a) Quell-Relationen aufstellen und komponieren
   b) Attribute innerhalb einer Zusammenhangskomponente des Baumes gleichsetzen
4. Head: Variablen der `SELECT`-Klausel übernehmen, dabei pro Zusammenhangskomponente nur eine Variable ins Ergebnisschema übernehmen

Beispiel 10.1. Gegeben sei SQL-Anfrage:

```
SELECT lastName, grade
FROM Student NATURAL JOIN Grade
WHERE studentID = '2';
```

Basierend auf den Quellrelationen `Student`(`studentID`, `lastName`, `firstName`, `studies`), kurz S, und `Grade`(`studentID`, `courseID`, `grade`), kurz G, ergibt sich ein Graph mit acht Knoten (siehe Abbildung 10.3).

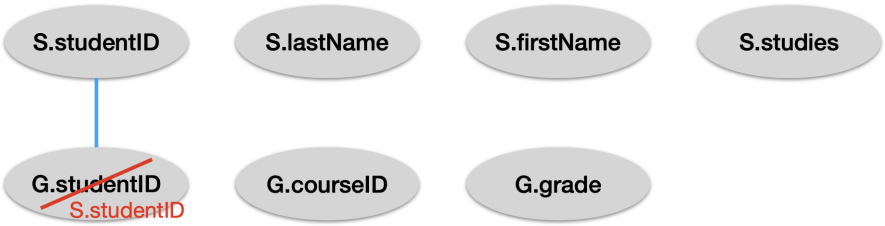


Abbildung 10.3. Parser-Baum für die obige Beispielanfrage.

Die Knoten $S.\texttt{studentID}$ und $G.\texttt{studentID}$ bilden eine Zusammenhangskomponente und werden im Body gleichgesetzt. In Abbildung 10.3 ist die Zusammenhangskomponente durch eine blaue Kante gekennzeichnet, die Gleichsetzung ist in rot hervorgehoben. Für den Body ergibt sich somit:

$$S(\texttt{studentID}, \texttt{lastName}, \texttt{firstName}, \texttt{studies}) \wedge G(\texttt{studentID}, \texttt{courseID}, \texttt{semester}, \texttt{grade}).$$

Die Attribute der Ergebnisrelation `Result` im Head der s-t tgd werden aus der `SELECT`-Klausel heraus gelesen. Dabei werden die selben gleichgesetzten Attributnamen wie im Body verwendet:

$$\texttt{Result}(\texttt{lastName}, \texttt{grade}).$$

Insgesamt ergibt sich so die s-t tgd:

$$S(\texttt{studentID}, \texttt{lastName}, \texttt{firstName}, \texttt{studies}) \wedge G(\texttt{studentID}, \texttt{courseID}, \texttt{grade}) \rightarrow \texttt{Result}(\texttt{lastName}, \texttt{grade}).$$

□

Für die Verarbeitung von Aggregatfunktionen wird zunächst eine entsprechende Fallunterscheidung eingeführt. Enthält die SQL-Anfrage keines der Schlüsselwörter `MAX`, `MIN`, `COUNT`, `SUM` oder `AVG`, so nutzt der Parser für die Transformation Algorithmus 6. Andernfalls wird zunächst die Aggregatfunktion nach Tabelle 10.3 transformiert und anschließend um die s-t tgds aus Algorithmus 6 erweitert. Auch die Einführung des SQL-Schlüsselwortes `OR`, das Statement `SELECT` $*$ sowie die verschiedenen Verbund-Varianten stellen eine Herausforderung dar. Details hierzu können in [RSZ21] und [Kav22] nachgelesen werden, welche die Konzeption und Implementierung des Parsers maßgeblich übernommen haben. Schauen wir uns nun die Invertierung der als Menge von (s-t) tgds transformierten Auswertungsanfrage Q an.

10.3. Invertierung der Auswertungsanfrage

Für die Invertierung einer SQL-Anfrage bzw. der Schema-Evolution, muss diese zunächst als Menge von (s-t) tgds vorliegen (siehe Abschnitt 10.2). Sei also Q eine Auswertungsanfrage, dargestellt als logische Formel erster Ordnung. Die *naive* Möglichkeit zur Invertierung von Q besteht im Vertauschen von Body und Head, wobei anschließend bekannte Konstanten sowie $\exists$-Quantoren für alle ungebundenen Variablen im Kopf ergänzt werden. Die (naive) Inverse der Anfrage

$$Q : \texttt{Student}(\texttt{studentID}, \texttt{name}) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{courseID}, 2.0) \rightarrow \texttt{Result}(\texttt{name})$$

entspricht dann[4]

$$Q^{-1} : \texttt{Result}(\texttt{name}) \rightarrow \exists \texttt{StudentID}, \texttt{CourseID} : \texttt{Student}(\texttt{StudentID}, \texttt{name}) \wedge \texttt{Grade}(\texttt{StudentID}, \texttt{CourseID}, 2.0).$$

Auch eine Rückführung einer Auswertungsanfrage auf ihre Basis-Operatoren — Projektion, Selektion, Verbund, Vereinigung, Schnitt, Differenz und Umbenennung — ist denkbar. Für die Invertierung wird die SQL-Anfrage zunächst bzgl. ihrer Basis-Operatoren geschachtelt, einzeln invertiert und anschließend wieder zusammengesetzt.

[4] Für eine bessere Übersicht sind alle $\exists$-quantifizierten Variabeln groß geschrieben.

Der Algorithmus hierzu kann in [För+21] nachgelesen werden. Wir beschränken uns an dieser Stelle auf ein Beispiel. Die zugehörigen Inversen sind in Tabelle 6.2 zusammengefasst. Für die Invertierung der Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM` und `AVG` verweisen wir auf Abschnitt 10.4.

Beispiel 10.2 (Union)**.** Gegeben sei folgende Beispielanfrage, welche den Verbund sowie die Vereinigung dreier Tabellen R, S und T beschreibt:

```
SELECT a, b
FROM R NATURAL JOIN S
WHERE c = 3
UNION
    ( SELECT a, b
    FROM T
    WHERE c = 4 )
```

Diese wird zunächst so geschachtelt, dass die Teil-Anfragen $q_{1,2}$ und $q_{2,1}$ lediglich eine Selektion, die Teil-Anfragen $q_{1,3}$ und $q_{2,2}$ eine Projektion und die Teil-Anfrage $q_{1,1}$ einen Verbund ausführt. Hierbei ist die folgende Reihenfolge zwingend einzuhalten: (i) Projektion, (ii) Selektion und (iii) Join. So ergibt sich:

```
SELECT a, b                                                    } q1,3
FROM ( SELECT *                                          }
       FROM ( SELECT *                          } q1,1   } q1,2
              FROM R NATURAL JOIN S )           }        }
              WHERE c = 3)                      }        }
UNION                                                    }
       ( SELECT a, b                                     } q2,2
       FROM ( SELECT *                          }
              FROM T                            } q2,1
              WHERE c = 4 ) )                   }
```

Die SFW-Blöcke $q_{1,i}$ und $q_{2,i}$ werden nun von innen nach außen einzeln in tgds $\sigma_{1,i}$ und $\sigma_{2,i}$ umgewandelt. Abschließend definieren wir je eine s-t tgd μ_1 und μ_2, welche die letzten Zwischenergebnisse in die Ergebnisrelation `Result` schreibt:

$$\sigma_{1,1}: \quad R(a,b) \wedge S(b,c) \rightarrow \texttt{Interim}_{1,1}(a,b,c)$$
$$\sigma_{1,2}: \quad \texttt{Interim}_{1,1}(a,b,c) \wedge c = 3 \rightarrow \texttt{Interim}_{1,2}(a,b,3)$$
$$\sigma_{1,3}: \quad \texttt{Interim}_{1,2}(a,b,3) \rightarrow \texttt{Interim}_{1,3}(a,b)$$
$$\mu_1: \quad \texttt{Interim}_{1,3}(a,b) \rightarrow \texttt{Result}(a,b)$$

$$\sigma_{2,1}: \quad T(a,b,c) \wedge c = 4 \rightarrow \texttt{Interim}_{2,1}(a,b,4)$$
$$\sigma_{2,2}: \quad \texttt{Interim}_{2,1}(a,b,4) \rightarrow \texttt{Interim}_{2,2}(a,b)$$
$$\mu_2: \quad \texttt{Interim}_{2,2}(a,b) \rightarrow \texttt{Result}(a,b)$$

Als nächstes invertieren wir die einzelnen (s-t) tgds $\sigma_{i,j}$ und μ_i durch Vertauschen von Body und Head. Alle ungebundenen Head-Variablen werden zudem durch einen $\exists$-Quantor gebunden. Für eine bessere Übersicht werden diese groß geschrieben. Insgesamt erhalten wir:

$$\mu_1^{-1}: \quad \texttt{Result}(a,b) \rightarrow \texttt{Interim}_{1,3}(a,b)$$
$$\sigma_{1,3}^{-1}: \quad \texttt{Interim}_{1,3}(a,b) \rightarrow \exists C: \texttt{Interim}_{1,2}(a,b,C)$$
$$\sigma_{1,2}^{-1}: \quad \texttt{Interim}_{1,2}(a,b,C) \rightarrow \texttt{Interim}_{1,1}(a,b,3)$$
$$\sigma_{1,1}^{-1}: \quad \texttt{Interim}_{1,1}(a,b,3) \rightarrow R(a,b) \wedge S(b,3)$$

$$\mu_2^{-1}: \quad \texttt{Result}(a,b) \rightarrow \texttt{Interim}_{2,2}(a,b)$$
$$\sigma_{2,2}^{-1}: \quad \texttt{Interim}_{2,2}(a,b) \rightarrow \exists C: \texttt{Interim}_{2,1}(a,b,c)$$
$$\sigma_{2,1}^{-1}: \quad \texttt{Interim}_{2,1}(a,b,c) \rightarrow T(a,b,4)$$

Abschließend werden die (s-t) tgds zu einer Menge von s-t tgds über R, S, T und `Result` zusammengefasst. Hierbei starten wir beim Body von μ_1^{-1} bzw. μ_2^{-1} und enden bei Head der letzten tgd $\sigma_{1,3}^{-1}$ bzw. $\sigma_{2,2}^{-1}$. Hierbei werden alle

„unterwegs“ auftretenden ∃-Quantoren und Konstanten in den Head der Inverse Q^{-1} integriert. Und wir erhalten:

$$Q_1^{-1}: \quad \texttt{Result}(a,b) \rightarrow R(a,b) \wedge S(b,3)$$
$$Q_2^{-1}: \quad \texttt{Result}(a,b) \rightarrow T(a,b,4).$$

□

In ProSA entscheiden wir uns für die direkte Invertierung, welche wir in Abschnitt 6.3, Algorithmus 3 vorgestellt haben. Da diese Variante unabhängig von den zugrundliegenden Basis-Operatoren für Auswertungsanfragen (siehe Tabelle 6.4) oder Schemamodifikationsoperatoren (siehe Tabelle 7.6 und Tabelle 7.7) ist, können wir diese sowohl für die Invertierung der Auswertungsanfragen also auch für die Evolution nutzen. Schauen wir uns nun an, wie wir auch die Aggregatfunktionen als Menge von (s-t) tgds darstellen können.

10.4. Einbindung von Aggregatfunktionen *(noch nicht veröffentlicht)*

Wir betrachten die Standard-Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM` und `AVG`. Für die Integration dieser Aggregatfunktionen in ProSA machen wir uns einige bekannte Eigenschaften und Regeln zu nutze. So kann `AVG` als Kombination von `COUNT` und `SUM` dargestellt werden. In [MG12] entspricht dies etwa der Datalog-Regel

$$\texttt{AVG}(S, A) :- \texttt{SUM}(S, A1), \texttt{COUNT}(S, A2), A = A1/A2.$$

Des weiteren kann `COUNT` als Spezialfall von `SUM` angesehen werden, indem wir das Inkrementieren eines Attributwertes als Spezialfall der Addition auffassen. Auch `MAX` und `MIN` unterscheiden sich kaum. Wir konzentrieren uns in unseren Überlegungen daher zunächst auf `MAX`, `SUM` und `AVG`, welche wir ausführlich diskutieren wollen. Eine Zusammenfassung der Formalisierungen ist in Tabelle 10.3 gegeben. Hierbei steht R für die originale, zu aggregierende Relation, R_S für ihre Sortierung bzgl. einer neu eingeführten ID, `Result` für das Anfrageergebnis und H, S, S' und S'' für notwendige Hilfsrelationen.

Die ersten beiden tgds von `COUNT`, `SUM` und `AVG` beschreiben jeweils die Sortierung der originalen Relation R. Diese beiden tgds können vernachlässigt werden, sofern die Provenance-IDs bereits sortiert sind. Dies ist bei ProSA beispielsweise der Fall.

Doch schauen wir uns zunächst noch einmal den Stand der Forschung an. So definieren die Autoren von [MG12] induktive Regeln zur Berechnung des Maximums oder der Summe einer Datalog-Menge. Die erste Regel spezifiziert dabei zunächst den Basisfall der Induktion, d.h. das Maximum oder die Summe einer leeren Menge. Die zweite Regel definiert dann induktiv das gewünschte Ergebnis. Im Falle der Summe etwa durch:

$$\texttt{SUM}(\{\}, 0)$$
$$\texttt{SUM}(X \mid S, A) :- \texttt{SUM}(S, A_1), A = A_1 + X.$$

Für die konkrete Berechnung verwenden die Autoren drei spezielle Operatoren: den `memberOf`-Operator, den Verkettungsoperator sowie den `setOf`-Operator. Details hierzu können in [MG12] nachgelesen werden. Wir übernehmen die Idee des `memberOf`- und des `setOf`-Operators in Form zusätzlicher Hilfsrelationen, welche sich die Provenance-IDs der verarbeiteten Tupel merken. Die Reihenfolgeabhängigkeit, welche im Chase nicht gegeben ist, modellieren wir zudem durch die Darstellung der Aggregatfunktion als eine Menge von mehreren Quell-tgds sowie einer abschließende s-t tgd.

Einen anderen Ansatz verfolgen die Autoren von [GM15]. Sie definieren das sogenannte *Datalog^{SUM}-Programm* als endliche Menge von Datalog- und Datalog$^{\text{SUM}}$-Regeln, wobei eine Datalog$^{\text{SUM}}$-Regel eine Regel der Form

$$p(\overline{Y}, S) \leftarrow body(\overline{W}, V), sum((\overline{X}), V, S)$$

ist. Das Programm besteht aus zwei Gruppen von Regeln. Die erste Regelgruppe partitioniert die Werte der Variablen in X und definiert für jede Partition eine lineare Ordnung auf den Tupeln des Bodys. Die zweite Regelgruppe berechnet die Summe jeder Partition durch Iteration über die entsprechende lineare Ordnung. Details können in

[GM15] nachgelesen werden. Wir übernehmen für unsere Überlegungen die Idee der zwei Regelgruppen zur Einführung einer linearen Ordnung und anschließenden Berechnung der Anzahl, Summe sowie Durchschnittsbildung, vgl. hierzu jeweils die ersten beiden tgds in der Tabellen 10.3b bis 10.3d.

$$\begin{aligned} R(a_i, b_i) \wedge R(a_j, b_j) \wedge b_i < b_j \quad &\rightarrow \quad H(b_i) \\ \text{bzw.} & \\ R(a_i, b_i) \wedge R(a_j, b_j) \wedge b_i > b_j \quad &\rightarrow \quad H(b_i) \end{aligned}$$

$$R(a, b) \wedge \neg H(b) \quad \rightarrow \quad \texttt{Result}(b)$$

(a) $\texttt{MAX}(b)/\texttt{MIN}(b)$, dargestellt als Menge von s-t tgds

$$\begin{aligned} R(a_i, b_i, t_i) \wedge \neg R_S(1, a_j, b_j, t_j) \quad &\rightarrow \quad S'(t_i) \wedge R_S(1, a_i, b_i, t_i) \\ R(a_i, b_i, t_i) \wedge \neg S'(t_i) \wedge R_S(x_j, a_j, b_j, t_j) \wedge \neg S(t_j) \quad &\rightarrow \quad S'(t_i) \wedge R_S(x_j + 1, a_i, b_i, t_i) \wedge S(t_j) \end{aligned}$$

$$R_S(x, a, b, t) \wedge \neg S(t) \quad \rightarrow \quad \texttt{Result}(x)$$

(b) $\texttt{COUNT}(*)$, dargestellt als Menge von s-t tgds

$$\begin{aligned} R(a_i, b_i, t_i) \wedge \neg R_S(x_j, a_j, b_j, t_j) \quad &\rightarrow \quad S'(t_i) \wedge R_S(1, a_i, b_i, t_i) \\ R(a_i, b_i, t_i) \wedge \neg S'(t_i) \wedge R_S(x_j, a_j, b_j, t_j) \wedge \neg S(t_j) \quad &\rightarrow \quad S'(t_i) \wedge R_S(x_j + 1, a_i, b_i, t_i) \wedge S(t_j) \end{aligned}$$

$$\begin{aligned} R_S(1, a, b, t) \quad &\rightarrow \quad S''(t) \wedge H(b, 1) \\ R_S(x, a, b, t) \wedge \neg S''(t) \wedge H(c, x - 1) \quad &\rightarrow \quad S''(t) \wedge H(f(b, c), x) \wedge f(b, c) = b + c \end{aligned}$$

$$H(c, x) \wedge \neg S(x) \quad \rightarrow \quad \texttt{Result}(c)$$

(c) $\texttt{SUM}(b)$, dargestellt als Menge von s-t tgds

$$\begin{aligned} R(a_i, b_i, t_i) \wedge \neg R_S(x_j, a_j, b_j, t_j) \quad &\rightarrow \quad S'(t_i) \wedge R_S(1, a_i, b_i, t_i) \\ R(a_i, b_i, t_i) \wedge \neg S'(t_i) \wedge R_S(x_j, a_j, b_j, t_j) \wedge \neg S(t_j) \quad &\rightarrow \quad S'(t_i) \wedge R_S(x_j + 1, a_i, b_i, t_i) \wedge S(t_j) \end{aligned}$$

$$\begin{aligned} R_S(1, a, b, t) \quad &\rightarrow \quad S''(t) \wedge H(b, 1) \\ R_S(x, a, b, t) \wedge \neg S''(t) \wedge H(c, x - 1) \quad &\rightarrow \quad S''(t) \wedge H(f(b, c), x) \wedge f(b, c) = b + c \end{aligned}$$

$$H(c, x) \wedge \neg S(x) \quad \rightarrow \quad \texttt{Result}(d) \wedge d = c/x$$

(d) $\texttt{AVG}(b)$, dargestellt als Menge von s-t tgds

Tabelle 10.3. Aggregatfunktionen, dargestellt als Menge von (s-t) tgds.

Die Initialisierung der Aggregatfunktionen lagern wir zum Teil in die XML-Dateien aus, welche als Eingabe für ChaTEAU dienen. Ein Beispiel hierfür ist im Anhang D zu finden. Hierfür erweitern wir die Eingabe, d.h. die SQL-Anfrage wie beispielsweise `SELECT AVG(grade) FROM` GRADE sowie die hinterlegte Datenbank mit der Tabelle GRADE, um zusätzliche Side Tables, welche die bereits verarbeiteten Tupel tracken, sowie zwei weitere Side Tables $H(c, x)$ und $R_S(x, a, b, t)$, welche verschiedene Zwischenergebnisse speichern. Auch die Ergänzung von Tupel-IDs ist Teil dieser Vorbereitung. Die In- und Dekrementierung der Variable x kann als skalare Funktion aufgefasst werden. Sie wird ebenso wie die skalare Multiplikation, Addition, Subtraktion und Division in der XML-Datei vermerkt und in der CHASE-Phase ausgeführt. Für ChaTEAU macht es dabei keinen Unterschied, ob eine einzelne s-t tgd oder eine Menge von mehreren (s-t) tgds mit Hilfe des CHASE verarbeitet werden. Das Erstellen der zugehörigen XML-Datei wird vom Parser aus Abschnitt 10.2 übernommen.

Fassen wir noch einmal zusammen: Die Formalisierung der Aggregatfunktionen besteht aus ein bis zwei Mengen von tgds sowie einer abschließenden s-t tgd. Die ersten beiden Quell-tgds — im Falle von `MAX` und `MIN` nicht notwendig — führen eine lineare Ordnung auf der Quell-Relation ein und zählen die Anzahl der Tupel in R. Hierfür schauen wir zunächst, ob R_S bereits ein Tupel enthält. Ist dies nicht der Fall, fügen wir zu R_S ein beliebiges Tupel aus R hinzu und initialisieren dies mit der ID 1. Anschließend fügen wir nach und nach neue Tupel zu R_S

hinzu und erhöhen dabei jeweils die neu definierte ID um 1, bis alle Tupel aus R abgearbeitet sind — hier hilft die Hilfsrelation[5] S'. Die dritte und vierte Quell-tgd, nutzen diese Ordnung zur Bestimmung des Maximums bzw. zur Berechnung der Summe aller b-Werte aus. Auch hier kommen weitere Hilfsfunktionen zum Einsatz. Während H die berechneten Zwischensummen speichert, garantiert S'', dass jedes Tupel aus R_S nur einmal in die Berechnung der Summe usw. eingeht. Die dritte Gruppe besteht schließlich aus einer s-t tgd, welche das Ergebnis der Aggregatfunktion auf die Ergebnisrelation `Result` abbildet. Da die Tupelanzahl bereits in der ersten Regelgruppe bestimmt wird, kann die zweite Regelgruppe im Falle von `COUNT` vernachlässigt werden. Schauen wir uns die in Tabelle 10.3 zusammengefassten Formalisierungen noch einmal am Beispiel an.

Maximum Für `MAX`(b) benötigen wir zwei Abhängigkeiten. Die erste tgd $R(a_i, b_i, t_i) \wedge R(a_j, b_j, t_j) \wedge b_i < b_j \rightarrow H(b_i)$ vergleicht paarweise alle Tupel in R und speichert jeweils den kleineren Attributwert b_i in der Hilfsrelation $H(b_i)$ ab. Abschließend wird mittels $R(a, b, t) \wedge \neg H(b) \rightarrow$ `Result`(b) (formalisiert als s-t tgd) der eine b-Wert bestimmt, welcher nicht in $H(b)$ gespeichert ist. Hierbei handelt es sich dann um das Maximum.

Summe Für die Darstellung von `SUM`(b) benötigen wir drei (induktive) Regelgruppen. Die ersten beiden tgds $R(a_i, b_i, t_i) \wedge \neg R_S(x_j, a_j, b_j, t_j) \rightarrow S'(t_i) \wedge R_S(1, a_i, b_i, t_i)$ und $R(a_i, b_i, t_i) \wedge \neg S'(t_i) \wedge R_S(x_j, a_j, b_j, t_j) \wedge \neg S(t_j) \rightarrow S'(t_i) \ \wedge \ R_S(x_j + 1, a_i, b_i, t_i) \ \wedge \ S(t_j)$ definieren eine Ordnung auf R. Die nächsten beiden tgds $R_S(1, a, b, t) \rightarrow S''(t) \ \wedge \ H(b, 1)$ und $R_S(x, a, b, t) \wedge \neg S''(t) \ \wedge \ H(c, x - 1) \rightarrow S''(t) \wedge H(f(b, c), x) \wedge f(b, c) = b + c$ berechnen die Summe der in R vorkommenden b-Werte und speichern diese in einer Hilfsrelation H. Die Hilfsrelationen S, S' und S'' speichern jeweils die bereits untersuchten Tupel. Sie existieren nur intern und sind außerhalb der beschriebenen Chase-Anwendung nicht von Bedeutung. Die s-t tgd $H(c, x) \wedge \neg S(x) \rightarrow$ `Result`(c) schreibt abschließend die berechnete Summe in die Ergebnisrelation `Result`.

Beispiel 10.3 (`SUM`(b))**.** Sei $R(a, b, t)$ die unten angegebene Quell-Relation. Wir initialisieren zunächst die notwendigen Side Tables $S(t)$, $S'(t)$ und $S''(t)$ sowie $R_S(x, a, b, t)$ und $H(c, x)$. Wir führen den Chase in drei Schritten aus: Als erstes werden die Tupel der Relation R durchnummeriert (die Reihenfolge der Tupel ist irrelevant) und in R_S abgespeichert. Eine mögliche Reihenfolge wäre:

R:

a	b	t
Max	3	t_1
Lukas	5	t_2
Julia	1	t_3
Daniel	12	t_4
Daniel	12	t_5

R_S:

x	a	b	t
1	Daniel	12	t_4
5	Max	3	t_1
3	Lukas	5	t_2
4	Julia	1	t_3
2	Daniel	12	t_5

S':

t
t_4
t_1
t_2
t_3
t_5

S:

t
t_4
t_1
t_2
t_3

Sind alle Tupel durchnummeriert, können die b-Werte addiert werden. Dafür wird, beginnend beim ersten Tupel, jedes Tupel einmal ausgewählt und der b-Wert in unserer Side Table addiert. Das zugehörige Ergebnis lautet:

R_S:

x	a	b	t
1	Daniel	12	t_4
2	Max	3	t_1
3	Lukas	5	t_2
4	Julia	1	t_3
5	Daniel	12	t_5

H:

c	x
12	1
15	2
20	3
21	4
33	5

S'':

t
t_4
t_1
t_2
t_3
t_5

Abschließend wird die Summe mit Hilfe einer s-t tgd $H(c, x) \wedge \neg S(x) \rightarrow$ `Result`(c) in die Ergebnisrelation `Result` geschrieben. Diese wird genau einmal angewandt und liefert das endgültige Ergebnis `Result`(33). □

Average `AVG`(b) wird analog zu `SUM`(b) als Menge von fünf (s-t) tgds dargestellt. Dabei unterscheiden sich die beiden Darstellungen lediglich in der abschließenden s-t tgd $H(c, x) \ \wedge \ \neg S(x) \rightarrow$ `Result`$(d) \ \wedge \ d = c/x$.

[5] Wir nennen in diesem Zusammenhang die Side Tables auch Hilfsrelationen, da sie lediglich zum Zählen bereits verarbeiteter Tupel sowie zum Speichern von Zwischenergebnissen genutzt werden. Attributwerte der originalen Tupel wie in den Abschnitten 6.3 und 7.3 beschrieben, werden hier nicht abgespeichert.

Inverse Aggregatfunktionen werden in ProSA nicht wie „klassische“ konjunktive Anfragen über den in Abschnitt 6.2 definierten Inverter invertiert. Stattdessen werden die zugehörigen Inversen abhängig von den gegebenen Provenance-Informationen explizit angegeben.

Bei der Invertierung der Aggregatfunktionen `COUNT`, `SUM` und `AVG` unterscheiden wir stets zwei verschiedene Inversentypen. Wird die Auswertungsanfrage `SELECT SUM(b) FROM R` naiv invertiert, so liefert das Anfrageergebnis $\texttt{Result}(x)$ stets ein Nulltupel zurück. Die zugehörige Inverse lautet dann

$$\texttt{Result}(c) \rightarrow \exists A, B : R(A, B).$$

Mittels zusätzlicher Provenance-Informationen wie den zuvor berechneten Hilfsrelationen $S'(t)$ und $H(c, x)$ können die konkreten Attributwerte für b wieder zurück berechnet werden. Alle weiteren Attributwerte wie etwa für das Attribut A werden wieder mit entsprechenden Nullwerten aufgefüllt. Die zugehörigen Inversen sind in Tabelle 10.6 zu finden.

Für die Aggregatfunktionen `SUM` und `AVG` muss zunächst die Hilfsrelation $H(c, x)$ „rückwärts“ ausgelesen werden. Hierfür benötigen wir erneut zwei tgds sowie eine neue Hilfsrelation $S'''(x)$, welche alle in verarbeiteten Tupel zählt. Die abschließende s-t tgd liefert schließlich das Ergebnis der Invertierung R.

$$S'(t) \wedge \neg S''(t) \quad \rightarrow \quad \exists A, B : R(A, B) \wedge S''(t)$$

(a) Inverse zu $\texttt{COUNT}(*)$, dargestellt als s-t tgd

$$\begin{aligned} \texttt{Result}(c) \wedge H(c, x) \wedge H(d, x-1) \wedge \neg S'''(x-1) \quad &\rightarrow \quad \exists A : R(A, c-d) \wedge S'''(x-1) \\ H(c, x) \wedge H(d, x-1) \wedge S'''(x) \wedge \neg S'''(x-1) \quad &\rightarrow \quad \exists A : R(A, c-d) \wedge S'''(x-1) \end{aligned}$$

$$H(c, x) \wedge S'''(1) \quad \rightarrow \quad \exists A : R(A, c)$$

(b) Inverse zu $\texttt{SUM}(b)$, dargestellt als Menge von (s-t) tgds

$$\begin{aligned} \texttt{Result}(e) \wedge c = e \cdot x \wedge H(c, x) \wedge H(d, x-1) \wedge \neg S'''(x-1) \quad &\rightarrow \quad \exists A : R(A, c-d) \wedge S'''(x-1) \\ H(c, x) \wedge H(d, x-1) \wedge S'''(x) \wedge \neg S'''(x-1) \quad &\rightarrow \quad \exists A : R(A, c-d) \wedge S'''(x-1) \end{aligned}$$

$$H(c, x) \wedge S'''(1) \quad \rightarrow \quad \exists A : R(A, c)$$

(c) Inverse zu $\texttt{AVG}(b)$, dargestellt als Menge von (s-t) tgds

Tabelle 10.6. Darstellung der invertierten Aggregatfunktionen als Menge von (s-t) tgds.

Die Invertierung von $\texttt{MAX}/\texttt{MIN}(b)$ liefert mittels $\texttt{Result}(b) \rightarrow \exists A : R(A, b)$ stets ein Nulltupel mit maximalem/minimalem b-Wert. Eine Rekonstruktion der korrekten Tupelanzahl ist ohne weitere Informationen nicht möglich. Hierfür müsste bereits bei der Berechnung des Anfrageergebnisses die Tupelanzahl bestimmt werden. Dies kann wie im Falle von $\texttt{COUNT}(*)$ beispielsweise durch die Definition einer Ordnung ermöglicht werden.

Chase-Inversentypen Ohne zusätzliche Provenance-Informationen wie Zeugenbasen oder Polynome sowie Side Tables kann für die Aggregatfunktionen `MAX` und `MIN` lediglich eine relaxte Chase-Inverse angegeben werden (siehe Tabelle 10.7). Grund hierfür ist, dass Duplikate verloren gehen. Der Chase-Inversentyp der Gruppierung entspricht dem Inversentyp der zugehörigen Aggregatfunktion[6].

Aggregatfunktion	Typ (ohne Provenance)	Typ (mit Provenance)
`MAX`	$\preccurlyeq$	$\preccurlyeq_{\text{tp}}$
`MIN`	$\preccurlyeq$	$\preccurlyeq_{\text{tp}}$
`COUNT`	$\preccurlyeq_{\text{tp}}$	$\preccurlyeq_{\text{tp}}$
`SUM`	$\leftrightarrow$	$=$
`AVG`	$\leftrightarrow$	$=$

Tabelle 10.7. Chase-Inversentypen mit und ohne zusätzlichen Provenance-Informationen

[6] Wir verwenden die Gruppierung stets in Kombination mit einer Aggregatfunktion.

Für die anderen drei Aggregatfunktionen ergänzen wir die Darstellung der (s-t) tgds um zusätzliche Provenance-IDs und Hilfsrelationen (siehe Tabelle 10.3b bis 10.3d). Im Fall von `SUM` und `AVG` können wir so den Chase-Inversentyp von ergebnisäquivalent auf exakt verbessern. Ohne diese zusätzlichen Informationen entspricht die Chase-Inverse stets der Identitätsabbildung

$$\texttt{Result}(c) \rightarrow \exists A_1, ..., A_{i-1}, A_{i+1}, ..., A_n : R(A_1, ..., A_{i-1}, c, A_{i+1}, ..., A_n)$$

und liefert lediglich ein Tupel mit dem aggregierten Ergebnis c für das Attribut A_i zurück.

Für die Aggregatfunktion `COUNT` hingegen ändern zusätzliche Informationen den Provenance-Typen nicht, da hier lediglich die Anzahl der Tupel und nicht die Attributwerte selbst von Bedeutung sind. In ProSA verwenden wir trotzdem stets die Darstellungen aus Tabelle 10.3a mit zusätzlichen Hilfstabellen $H(c, x)$, $S(t)$, $S'(t)$ sowie $S''(t)$.

Umsetzung in der ProSA-Software Für die Umsetzung von Aggregatfunktionen in der Implementierung von ProSA benötigen wir jedoch eine leicht abgewandelte Darstellung. Grund hierfür ist, dass die Provenance-IDs t_i nicht direkt angesprochen werden können. Stattdessen behelfen wir uns mit einem Duplikat R'' der originalen Relation R, welche nach und nach mit allen abgearbeiteten Tupeln befüllt wird, sowie den bekannten Hilfsrelationen S und H. Die Darstellung von `SUM`(b) ist exemplarisch in Tabelle 10.8 zu sehen, wobei die Unterschiede zu den oben eingeführten Darstellungen (siehe Tabelle 10.3c sowie Tabelle 10.6b) orange hervorgehoben sind. Für weitere Details sei auf das Git-Repository von ProSA verwiesen.

$$\begin{aligned}
R(a_i, b_i) \wedge \neg R_S(x_j, a_j, b_j) &\rightarrow R''(a_i, b_i) \wedge R_S(1, a_i, b_i) \\
R(a_i, b_i) \wedge \neg R''(a_i, b_i) \wedge R_S(x_j, a_j, b_j) \wedge \neg S(x) &\rightarrow R''(a_i, b_i) \wedge R_S(x_j + 1, a_i, b_i) \wedge S(x)
\end{aligned}$$

$$\begin{aligned}
R_S(1, a, b) &\rightarrow H(b, 1) \\
R_S(x, a, b) \wedge \neg H(d, x) \wedge H(c, x - 1) &\rightarrow H(f(b, c), x) \wedge f(b, c) = b + c
\end{aligned}$$

$$H(c, x) \wedge \neg S(x) \rightarrow \texttt{Result}(c)$$

(a) `SUM`(b), dargestellt als Menge von (s-t) tgds

$$\begin{aligned}
\texttt{Result}(c) \wedge H(c, x) \wedge H(d, x - 1) \wedge \neg H'(x - 1) &\rightarrow \exists A : R(A, c - d) \wedge H'(x) \\
H(c, x) \wedge H(d, x - 1) \wedge H'(x) \wedge \neg H'(x - 1) &\rightarrow \exists A : R(A, c - d) \wedge H'(x - 1) \\
H(c, 1) &\rightarrow \exists A : R(A, c)
\end{aligned}$$

(b) Inverse zu `SUM`(b), dargestellt als Menge von (s-t) tgds

Tabelle 10.8. Darstellung von `SUM`(b), umgesetzt in ProSA.

Gruppierung Die Gruppierung erfolgt in SQL in der `GROUP BY`-Klausel durch Angabe einer Liste von Gruppierungsattributen. Wir beschränken uns im Folgenden auf das Vorliegen EINES Gruppierungsattributes. Der Ansatz kann jedoch beliebig erweitert werden. Sei hierzu eine Auswertungsanfrage wie folgt gegeben:

```
SELECT g,f(b)
FROM R
GROUP BY g
```

Beim Parsen der Anfrage (siehe Abschnitt 10.2) erweitern wir das Schema $R(a_1, ..., a_n)$ zu $R(g, a_1, ..., a_m, t)$, wobei a_i den Attributen, t der Provenance-ID und g dem Gruppierungsattribut entsprechen. Um die Gruppierung im Falle von Aggregatfunktionen zu ergänzen, genügt es daher die Regeln aus Tabelle 10.3 um die zusätzliche Variable g zu ergänzen. Exemplarisch seien in Tabelle 10.9 die erweiterten Regeln für die Aggregatfunktionen `MAX` und `SUM` angegeben. Das Gruppierungsattribut ist jeweils orange hervorgehoben. Erste Ideen hierzu können zudem in [Kav22] nachgelesen werden.

$$
\begin{aligned}
R(g_i, a_i, b_i) \wedge R(g_j, a_j, b_j) \wedge b_i < b_j &\rightarrow H(g_i, b_i) \\
R(g, a, b) \wedge \neg H(g, b) &\rightarrow \texttt{Result}(g, b)
\end{aligned}
$$

(a) MAX(b), dargestellt als Menge von s-t tgds

$$
\begin{aligned}
R(g_i, a_i, b_i, t_i) \wedge \neg R_S(x_j, g_j, a_j, b_j, t_j) &\rightarrow S'(t_i) \wedge R_S(1, g_i, a_i, b_i, t_i) \\
R(g_i, a_i, b_i, t_i) \wedge \neg S'(t_i) \wedge R_S(x_j, g_j, a_j, b_j, t_j) \wedge \neg S(t_j) &\rightarrow S'(t_i) \wedge R_S(x_j + 1, g_i, a_i, b_i, t_i) \wedge S(t_j)
\end{aligned}
$$

$$
\begin{aligned}
R_S(1, g, a, b, t) &\rightarrow S''(t) \wedge H(g, b, 1) \\
R_S(x, g, a, b, t) \wedge \neg S''(t) \wedge H(g, c, x - 1) &\rightarrow S''(t) \wedge H(g, f(b, c), x) \wedge f(b, c) = b + c
\end{aligned}
$$

$$
H(g, c, x) \wedge \neg S(x) \rightarrow \texttt{Result}(g, c)
$$

(b) SUM(b), dargestellt als Menge von s-t tgds

Tabelle 10.9. Darstellung der Aggregatfunktionen erweitert um GROUP BY .

Bei den Aggregatfunktionen MAX und MIN werden durch Hinzufügen des Gruppierungsattributes g die Attribute b_i und b_j lediglich innerhalb einer Gruppe verglichen. Die Ausführung der (s-t) tgds bleibt ansonsten unverändert. Im Fall der anderen drei Aggregatfunktionen COUNT, SUM und AVG muss die Nummerierung x hingegen pro Gruppe angepasst werden. Dies hat zur Folge, dass die erste tgd des ersten und zweiten Blocks nun mehrfach angewandt werden, da die Nummerierung je Gruppe stets bei 1 beginnt. Doch schauen wir uns dies am konkreten Beispiel einmal an.

Beispiel 10.4 (SUM(b), Fortsetzung)**.** Sei erneut $R(a, b, t)$ die Relation von oben. Wir wählen a als Gruppierungsattribut. Seien weiter $S(t)$, $S'(t)$, $S''(t)$, $R_S(g, x, a, b, t)$ und $H(a, c, t)$ die notwendigen Side Tables. Dann liefert die Anwendung des Chase (bei gleicher Reihenfolge und Chase-Anwendung wie oben):

R:

a	b	t
Max	3	t_1
Lukas	5	t_2
Julia	1	t_3
Daniel	12	t_4
Daniel	12	t_5

R_S:

x	g	a	b	t
1	Daniel	Daniel	12	t_4
1	Max	Max	3	t_1
1	Lukas	Lukas	5	t_2
1	Julia	Julia	1	t_3
2	Daniel	Daniel	12	t_5

S':

t
t_4
t_1
t_2
t_3
t_5

S:

t
t_4

H:

g	c	x
Daniel	12	1
Max	3	1
Lukas	5	1
Julia	1	1
Daniel	24	2

S'':

t
t_4
t_1
t_2
t_3
t_5

Result:

g	c
Max	3
Lukas	5
Julia	1
Daniel	24

Das Ergebnis der Anfrage ist in der Tabelle Result zusammengefasst. □

Schauen wir uns abschließend noch die Chase-Inversentypen sowie die Invertierung von Anfragen mit Aggregatfunktionen und Gruppierung an: Die Chase-Inversentypen mit Gruppierung unterscheiden sich nicht von denen ohne Gruppierung. Sie stimmen daher, wie in Tabelle 6.4 zu sehen, überein. Auch die Invertierung ergibt sich analog zu oben.

Die in Tabelle 10.3 bzw. Tabelle 10.8 vorgestellten Formalisierungen der Aggregatfunktionen MAX, MIN, COUNT, SUM und AVG sind in ProSA umgesetzt und liefern das erwartete Ergebnis (siehe Abbildung 10.4). Zur Zeit ist neben einer Aggregatfunktion jedoch keine weitere Operation zulässig. Dies ist jedoch kein konzeptionelles Problem, sondern eine Frage der Implementierung.

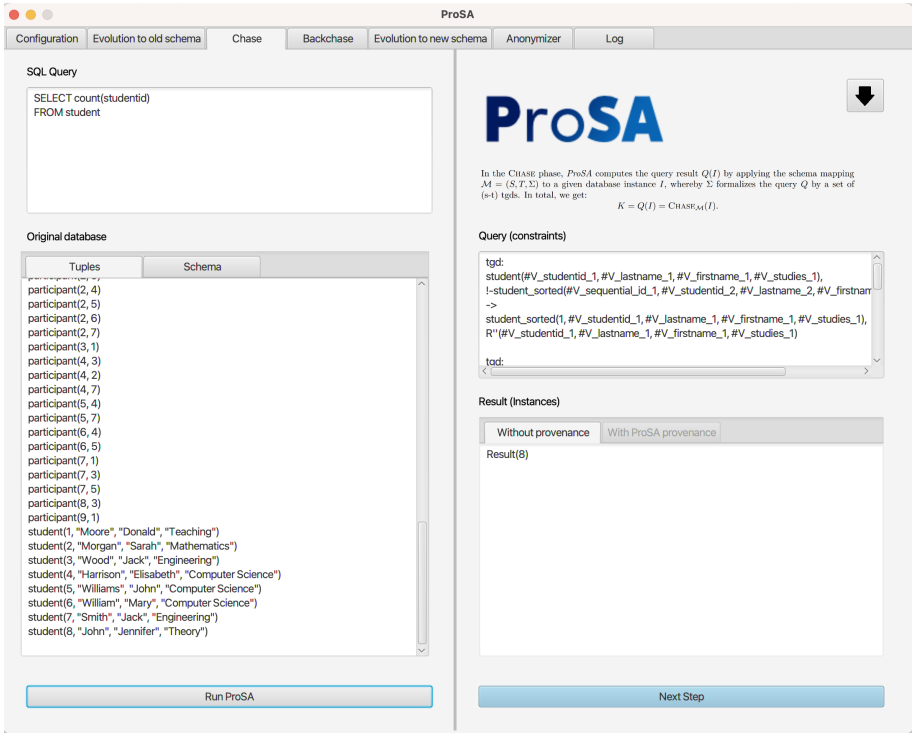


Abbildung 10.4. Auswertung einer Aggregatfunktion in ProSA.

10.5. Einbindung von Provenance (als Erweiterung von ChaTEAU)

Einige SQL-Anfragen können nicht ohne zusätzlichen Informationen als Menge von (s-t) tgds dargestellt werden. Hierzu gehören beispielsweise die in Abschnitt 10.4 vorgestellten SQL-Anfragen mit Aggregatfunktionen sowie Anfragen mit bestimmten Schlüsselwörtern, wie in Abschnitt 10.2 vorgestellt. Für die Berechnung der (minimalen) Teil-Datenbank sind diese Zusatzinformationen somit unabdingbar und wir entscheiden uns bei der Verarbeitung der Auswertungsanfrage Q stets für den Provenance-aware Chase&Backchase nach Definition 6.11. Ein Beispiel hierfür ist zudem in Abschnitt 10.1 unter den Punkten 5, 3 und 6 zu finden.

Anders verhält es sich jedoch bei der Integration der Evolution. Im Kontext des Forschungsdatenmanagements rechnen wir stets mit einem Datenzuwachs sowie einer „positiven“ Schemaändernung (siehe Abschnitt 2.3) wie `ADD Table`, `ADD Column`, `MERGE Column` oder `SPLIT Column` (siehe Abschnitt 7.2). D.h. sowohl Änderungen des Datenbestandes oder des Schemas haben keinen Informationsverlust zur Folge. Die Speicherung zusätzlicher Provenance-Informationen ist somit überflüssig. Der Datenverlust, welcher beim Invertieren der Evolution entsteht, ist in unserem Fall ebenfalls nicht von Bedeutung. Lediglich der Verlust von Informationen über den zugrundeliegenden Datentyp ist denkbar. Die hierfür eingeführte ***what***-Provenance (siehe Abschnitt 7.4) kann jedoch implementierungstechnisch umgesetzt werden und kann an dieser Stelle daher ebenfalls vernachlässigt werden. Für die Verarbeitung der Evolution verwenden wir daher den Chase&Backchase ohne zusätzliche Provenance-Informationen.

Provenance-Informationen speichern Unter Provenance-Informationen verstehen wir in ProSA neben den Zeugenbasen $W_{Q,d}$ der ***why***-Provenance zusätzlich noch die in Abschnitt 6.3 vorgestellten Side Tables. Diese speichern zusätzliche Attributwerte und Tupel-IDs — meist in den Tabellen K', K'', S', S'', usw. — oder Zwischenergebnisse meist in der Tabelle H (vgl. hierzu die Abschnitte 6.3 und 10.4). Ob zusätzliche Side Tables notwendig sind, hängt dabei insbesondere von den Operationen der SQL-Anfrage ab. Ein Überblick über die relevanten Operationen sind in Tabelle 10.12 zusammengefasst.

Wahl des Provenance-Typs Wie in Abschnitt 8.2 beschrieben, verhalten sich die ***why***- und ***how***-Provenance unter Berücksichtigung des Privacy-Aspekts ähnlich. Da die zusätzlichen Informationen der Provenance-Polynome im Laufe des Anonymisierungsprozesses wieder verloren gehen, liegt die Verwendung der Zeugenbasen, d.h. der nächst

Operationen	Zeugenbasis	Side Tables
Identitätsabbildung	✗	✗
Umbenennung β	✗	✗
Projektion π	✓	✗
Natürlicher Verbund $\bowtie$	✓	✓
Selektion σ	✗	✗
Vereinigung $\cup$	✓	✗
Schnittmenge $\cap$	✗	✗
Differenz $-$	✓	✓
Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM`, `AVG`	✓	✓
Gruppierung γ	✓	✓
skalare Operationen $+, -, \cdot, :$	✗	✗

Tabelle 10.12. Operationen, welche für die Verbesserung der (minimalen) Teil-Datenbank zusätzliche Provenance-Informationen wie Zeugenbasen und Side Tables benötigen (✓) bzw. nicht benötigen (✗).

„schwächeren“ Data Provenance, nahe. Die Speicherung und Verarbeitung der Provenance-Polynome ist zudem um einiges komplizierter als die der Zeugenbasen. Speichern wir jedoch die konkreten Attributwerte in zusätzlichen Side Tables — sofern notwendig —, erweitern wir den Informationsgehalt der ***why***-Provenance auf den der ***how***-Provenance. Auf diese Weise können wir Speicherplatz sparen und die Berechnung der Provenance-Informationen „schnell“ und einfach halten.

In ProSA werden bei der Auswertung der Anfrage Q alle drei Provenance-Typen berechnet. Für die weitere Verarbeitung, d.h. für die Bestimmung der (minimalen) Teil-Datenbank beschränken wir uns jedoch auf die ***why***-Provenance. Diese kann, wie in Abschnitt 10.1 beschrieben, als zusätzliches Attribut in die Chase-Anwendung integriert werden. Sei hierzu t ein beliebig gewähltes Tupel der Quell-Instanz I. Dann ergänzen wir zunächst t um ein neues Attribut, die global eindeutige *Provenance-ID*. Diese ergibt sich aus dem Präfix des Relationennamen, dem Wort *id* sowie einer fortlaufenden Nummer. Für das erste Tupel s_1 der Relation `Student` ergibt sich so die ID s_{id_1}. Seien weiter Q und E eine gegebene Auswertungsanfrage bzw. ein Evolutionsschritt. Dann wird auch Q um das zusätzliche Provenance-Attribut ergänzt. Gleiches kann — falls gewünscht — für die Evolution durchgeführt werden. Wir beschränken uns im Folgenden auf die Ergänzung der Auswertungsanfrage. Die Anfrage, welche die Noten des Studierenden names *Jack* ausgibt, lautet entsprechend

$$\texttt{Student}(\texttt{studentID}, \text{'Jack'}, s_{\text{id}}) \wedge \texttt{Grade}(\texttt{studentID}, \texttt{grade}, g_{\text{id}}) \rightarrow \texttt{Result}(\text{'Jack'}, \texttt{grade}, s_{\text{id}}, g_{\text{id}}).$$

Analog wird ihre Inverse zu

$$\texttt{Result}(\text{'Jack'}, \texttt{grade}, s_{\text{id}}, g_{\text{id}}) \rightarrow \exists \texttt{StudentID}: \ \texttt{Student}(\texttt{StudentID}, \text{'Jack'}, s_{\text{id}}) \wedge \texttt{Grade}(\texttt{StudentID}, \texttt{grade}, g_{\text{id}})$$

ergänzt. Weitere Details hierzu sind in Abschnitt 10.1 sowie in [AHH22] zu finden.

10.6. Bestimmung des Chase-Inversentyps

Zur Klassifikation der Anfrage Q bestimmen wir zudem den zugehörigen Chase-Inversentyp. Wie in Abschnitt 6.3 beschrieben, ergibt sich dieser aus den in Q enthaltenen Operationen. Algorithmus 7 beschreibt die hierfür relevante Klassifikation: Liegen beispielsweise ein Vereinigung oder eine der beiden Aggregatfunktionen `SUM` oder `AVG` vor, so hat Q eine ergebnisäquivalente Chase-Inverse. Ist dies nicht der Fall, ist der Chase-Inversentyp relaxt oder besser. So hat Q nur im Falle der Identitätsabbildung, d.h. $Q =$ `SELECT` $*$ `FROM` $<$ `relation` $>$, eine exakte Chase-Inverse.

Algorithm 7 Bestimmung des Chase-Inversentyps (ohne Provenance)

Require: SQL-Anfrage Q
Ensure: invertierte Anfrage Q'

```
1:  if union || sum || avg then
2:      return InverseType.RESULT_EQUIVALENT;
3:  else if max || min || selection || diff || intersect || join || projection then
4:      return InverseType.RELAXED;
5:  else if count then
6:      return InverseType.TP_RELAXED;
7:  else if !projection then
8:      return InverseType.EXACT;
9:  else
10:     return InverseType.NO_INVERSE;
```

Wie zuvor können zusätzliche Provenance-Informationen den Chase-Inversentyp verbessern (siehe Algorithmus 8). So kann in diesem Fall stets mindestens eine relaxte Chase-Inverse garantiert werden. Der Inversentyp der Vereinigung sowie der Aggregation von oben verändert sich beispielsweise zu einer exakten Chase-Inversen. Dies gilt jedoch nicht für alle Inversentypen. So bleibt der Inversentyp von `COUNT` mit und ohne zusätzliche Provenance-Informationen bei tp-relaxed.

Algorithm 8 Bestimmung des Chase-Inversentyps (mit Provenance)

Require: SQL-Anfrage Q
Ensure: invertierte Anfrage Q'

```
1:  if selection || intersect then
2:      return InverseType.RELAXED;
3:  else if count || max || min || projection then
4:      return InverseType.TP_RELAXED;
5:  else if join || union || diff || sum || avg || !projection then
6:      return InverseType.EXACT;
7:  else
8:      return InverseType.NO_INVERSE;
```

Wir setzen die Bestimmung des Chase-Inversentyps in ProSA somit nicht über die Darstellung der Anfrage als Menge von (s-t) tgds um, sondern verbleiben bei der originalen SQL-Anfrage. Der entsprechende Code kann im Git-Repository von ProSA nachgelesen werden.

10.7. Einbindung von Privacy (als Erweiterung von ChaTEAU)

Für uns geht der Begriff des Datenschutzes über den Begriff des (in der Regel personenbezogenen) Datenschutzes hinaus. Vielmehr beziehen wir uns auf den Schutz von Forschungsdaten im Allgemeinen. Gründe für den Schutz von Forschungsdaten können wirtschaftliche (Firmenschutz), persönliche (personenbezogene Daten) oder finanzielle Aspekte haben, da die Erstellung solcher Daten oft sehr zeitaufwendig und teuer ist (siehe Abschnitt 8.1). MOSAiC beispielsweise, ein internationales Forschungsprojekt zur Untersuchung der zentralen Arktis, hat ein fünfjähriges Vorrecht, bevor es seine durch öffentliche Mittel finanzierten Daten veröffentlichen muss.[7] Dies gewährt eine Art „Forschungsvorsprung“, welcher den Aufwand der Datenerzeugung „ausgleichen“ soll. Auch die Identifizierung von persönlichen oder firmen-internen Informationen sollte strikt vermieden werden, ebenso wie die Veröffentlichung militärischer oder geheimdienstlicher Daten.

In diesem Sinne kann es sinnvoll sein, die berechnete (minimale) Teil-Datenbank weiter zu anonymisieren. Gegeben seien hierzu eine Datenbankinstanz I, eine Anfrage Q sowie ein zugehöriges Ergebnis $Q(I)$. Gesucht ist die anonymisierte (minimale) Teil-Datenbank I^*_{anon}, welche nach Anwendung von Q dasselbe Ergebnis $Q(I)$ oder aber ein äquivalentes Ergebnis $Q(I^*)$ oder $Q(I^*_{\text{anon}})$ liefert, welches durch Anwendung eines passenden Homomorphismus intensional auf $Q(I)$ bzw. $Q(I^*)$ abgebildet werden kann. Insgesamt gilt dann:

$$Q(I^*_{\text{anon}}) \preccurlyeq Q(I^*) \preccurlyeq Q(I).$$

[7] MOSAiC: https://spaces.awi.de/display/DM/MOSAiC+Data+Policy

Bei der Rekonstruktion der (minimalen) Teil-Datenbank dürfen nur solche Tupel rekonstruiert werden, welche nicht im Widerspruch zu speziellen Datenschutzaspekten oder allgemeinen Privacy-Aspekten stehen. Je nach gewählter Data Provenance-Frage (***where***, ***why*** oder ***how***) sind unterschiedliche Privacy-Probleme zu berücksichtigen [ASH21a]: (1) Bei der Verwendung von Relationsnamen oder Tupellisten als Nachweis liegen in der Regel zu wenig schützenswerte Daten vor, um ein Privacy-Problem darzustellen. Problematisch wird es erst dann, wenn wir die mittels ***where***-Provenance generierten Informationen zum erneuten Auslesen der Quell-Instanz nutzen würden. (2) Verwenden wir hingegen die ***why***-Provenance, ist eine Verletzung von Privacy-Aspekten nicht auszuschließen. (3) Die zusätzlichen Informationen der ***how***-Provenance liefern in der Regel zu viele Informationen. Diese können jedoch mit Hilfe zusätzlicher Side Tables auch bei Verwendung der ***why***-Provenance erzeugt werden. In diesem Fall sind die ***why***- sowie die ***how***-Provenance also als „gleichwertig“ anzusehen. Wie in Abschnitt 8.2 beschrieben, entscheiden wir uns in ProSA daher für die Zeugenbasen der ***why***-Provenance.

Wählen wir als nächstes ein passendes Anonymisierungsmaß sowie eine Anonymisierungsmethode. Anschließend untersuchen wir die ProSA-Pipeline noch auf den richtigen Zeitpunkt für die Anonymisierung.

10.7.1. Auswahl eines Anonymisierungsmaßes sowie einer Anonymisierungsmethode

Für die Anonymisierung eines Datensatzes existieren nach [GLS14] mehr als ein Dutzend verschiedene Anonymisierungsmaße, welche großteils auf der k-Anonymität basieren. Nach einer ersten groben Selektion haben wir uns für sechs Anonymisierungsmaße entschieden, welche für die Einbindung in ProSA genauer untersucht wurden. Hierzu gehören die k-Anonymität, ihre Erweiterungen l-Diversität, k-Map, t-Closeness und δ-Presence (siehe Abschnitt 3.6). Abschließend betrachten wir noch die häufig verwendete Differential Privacy. Erste Ideen zum Auswahlprozess passender Maße können zudem in [Sch22] nachgelesen werden.

k-Anonymität und ihre Erweiterungen Die „k-Anoynmität ist ein formelles Datenschutzmodell, mit dem Aussagen über anonymisierte Datensätze getroffen werden können“ (Quelle: Wikipedia, 13.09.2022). Sie wurde 2002 erstmals von Latanya Sweeney veröffentlicht und bildet die Grundlage für viele weitere Anonymisierungsmaße wie beispielsweise die l-Diversität oder δ-Presence (siehe Abschnitt 3.6).

- **k-Anonymität** besagt, dass zu jedem Tupel einer Datenbank D mindestens $k-1$ weitere un-unterscheidbare Tupel in D existieren müssen. Sie ist allgemein nicht ohne Informationsverlust möglich, kann aber mittels Generalisierung, Hinzufügen von Rauschen sowie das Hinzufügen oder Löschen von Tupeln in den meisten Datenbanken umgesetzt werden. So auch in ProSA.

- **l-Diversität** kann als Erweiterung der k-Anonymität ebenfalls in den meisten Datenbanken umgesetzt werden. Sie berücksichtigt zudem eine Unterscheidung in sensible und nicht-sensible Attribute. Die l-Diversität liefert somit eine größere Anonymisierung als die k-Anonymität, hat entsprechend aber auch einen höheren Informationsverlust zu verzeichnen.

- **t-Closeness** ist eine Erweiterung der l-Diversität, welche die sensiblen Attribute aller Äquivalenzklassen auf ähnliche Weise, ohne die globale Verteilung dieser sensiblen Attributwerte in der Beziehung zu berücksichtigen. Im Fall realer Datensätze können die Werte sensibler Attribute jedoch verzerrt sein. Da eine globale Verteilung der sensiblen Attribute nicht immer gegeben ist, entscheiden wir uns gegen t-Closeness.

- **k-Map** ist ebenfalls eine Erweiterung der k-Anonymität. Sie sucht die k un-unterscheidbaren Tupel nicht in der Datenbank D, sondern in einer allumfassenden Grundgesamtheit. Dies macht k-Map unempfindlich für Membership Disclosure sowie Attribute Disclosure (unter Umständen). Dieses Maß ist in der Praxis aufgrund der fehlenden Informationen über die Grundgesamtheit jedoch nicht anwendbar. Da ProSA unabhängig von der zu Grunde liegenden Datenmenge arbeiten soll, kommt k-Map für uns an dieser Stelle nicht in Frage.

- **δ-Presence** beschreibt das Verhältnis zwischen k-Anonymität und k-Map, wobei der garantierte Datenschutz zunimmt, je kleiner $\delta = k_{\text{Anon}}/k_{\text{Map}}$ ist. Da wir k-Map für ProSA bereits ausgeschlossen haben, kommt auch δ-Presence hier nicht in Frage.

Da für die Einhaltung von k-Map und δ-Presence zusätzliche Informationen über die Grundgesamtheit aller möglichen Datensätze über die in ProSA gegebenen Datenbank hinaus bekannt sein müssten, was wir in ProSA allerdings nicht garantieren bzw. abfragen können, müssen wir diese beiden Anonymisierungmaße vernachlässigen. Wir stehen damit vor dem selben Problem, welches in der Literatur immer wieder angesprochen wird [NC10], wohingegen die k-Anonymität sowie die l-Diversität für die meisten Datenbanken anwendbar sind.

Differential Privacy Bei der Differential Privacy wird eine Anfrage Q nicht direkt an die Datenbank D gestellt. Stattdessen wird eine Funktion $\mathcal{K}$ zwischengeschaltet, welche die Daten zuvor verrauscht. Sie eignet sich insbesondere für große, statistische Datensätze sowie einem klein gewählten Parameter ϵ (siehe Definition 3.48). Wie in Abschnitt 8.2 beschrieben, widerspricht das für die Erfüllung von Differential Privacy eingeführte Rauschen jedoch der in ProSA genutzten Provenance.

Wahl des Anonymisierungsmaßes Wir fassen zusammen: Die k-Anonymität und l-Diversität zählen zu den gängigsten Anonymisierungsmaßen, welche auf den meisten Datenbanken angewendet werden können. Um die Anwendbarkeit von ProSA möglichst allgemein zu halten, entscheiden wir uns daher für diese beide Maße. Aufgrund der in ProSA verwendeten Provenance-Annotationen kommt Differential Privacy für uns daher ebenfalls nicht in Frage.

Auswahl einer Anonymisierungsmethode Für die Anonymisierung in ProSA haben wir sechs Anonymisierungsmethoden genauer untersucht. Hierzu gehören die Generalisierung, das Unterdrücken von Tupeln oder Spalten, die Permutation, intensionale Antworten, das Verrauschen von Daten sowie das Slicing (siehe Abschnitt 3.6).

- Die **Generalisierung** ist abhängig von der Art des zu generalisierenden Attributes. Wie in Abschnitt 8.3 beschrieben, unterscheiden wir vier verschiedene Varianten von Attributen, welche alle in ProSA-Datenbanken auftreten können. Da sich alle vier Attributtypen direkt oder durch eine zugehörige Konzept- bzw. Domänengeneralisierungshierarchie generalisieren lassen, sind diese in ProSA technisch problemlos umsetzbar. Zudem garantiert die Generalisierung die k-Anonymität.

- Wie zu Abschnitt 3.6 beschrieben, wird die Generalisierung oft mit dem **Unterdrückung von Tupeln und Spalten** kombiniert. Dabei werden „alle Tupel, die sich keiner Äquivalenzklasse hinsichtlich eines Quasi-Identifikators zuordnen lassen, aus dem Datensatz entfernt“ [Sch22]. Dies können wir zur Garantie der k-Anonymität auch in ProSA nutzen. Aus technischen Gründen, ist die Unterdrückung im System ProSA jedoch nur prototypisch implementiert. Da die Unterdrückung nicht effektiv ist [MW04], ziehen wir die Generalisierung stets vor (siehe Abschnitt 10.7).

- **Permutation** kann wie in [PS17] insbesondere Unsorted-Matching-Angriffe verhindern, indem es die zugrundeliegende Struktur der Daten verändert bzw. verschleiert. Dabei wird die Permutation in der Regel mit einer zweiten Anonymisierungsmethode kombiniert, da sie alleinstehend nicht zur Lösung von Privacy-Problemen geeignet ist. Die Permutation kann daher auch in ProSA problemlos ergänzt und mit den ausgewählten Methoden — Generalisierung und Unterdrückung — kombiniert werden. Aktuell wird die Permutation indirekt durch die zufällige Zuordnung der Provenance-IDs unterstützt. Dies gilt insbesondere für die Ausführung von Aggregatfunktionen (siehe Abschnitt 10.4).

- Ziel von ProSA ist die Angabe einer (minimalen) Teil-Datenbank zur Reproduktion bzw. Replikation eines veröffentlichten Ergebnisses. Um eine anschließende Auswertung der Teil-Datenbank I^* durch die ursprüngliche Anfrage Q oder eine umgeschriebene Anfrage gewährleisten zu können, muss die Teil-Datenbank dabei explizit, d.h. durch Angabe konkreter Attribut- oder Nullwerte, angegeben werden. Nach der Anonymisierung ist dies jedoch in der Regel nicht mehr möglich. Hier behelfen wir uns mit einer **intensionale**, d.h. textuelle Beschreibung der Teil-Datenbank. Analog zu den in [Sva16; Lam21; Sch22] vorgestellten Ansätzen zur Kombination von Provenance, Privacy und intensionalen Antworten sowie deren Anwendungsmöglichkeiten basiert unsere Anonymisierung auf den selben Konzepthierarchien (siehe Abschnitt 10.7).

- Differential Privacy basiert auf dem **Verrauschen der Daten**. Dies widerspricht jedoch der in ProSA genutzten Data Provenance, wie in Abschnitt 8.2 beschrieben.

- Da **Slicing** ebenso wie die Permutation auf dem Vertauschen von Tupeln basiert, welches beim Verteilen der Provenance-IDs automatisch integriert wird, verzichten wir auf diese Anonymisierungsmethode. Die Permutation soll uns an dieser Stelle ausreichen.

Zur Garantie der k-Anonymität sowie l-Diversität eignet sich insbesondere die Generalisierung und Intensionalisierung von Attributen. Auch die Permutation sowie das Unterdrücken von Tupeln und Spalten kommen für ProSA in Frage. Wir entscheiden uns daher für das Generalisieren auf Basis von gegebenen Konzept- und Domänengeneralisierungshierarchien (siehe Abschnitt 10.9). Die Permutation ist bei der Generierung der Provenance-IDs sowie der Auswertung von Aggregatfunktionen bereits indirekt enthalten und die Unterdrückung wird aufgrund ihrer Ineffektivität soweit möglich vermieden. Beide Methoden sind jedoch in ProSA integriert.

Provenance und Privacy in ProSA Wie in Abschnitt 8.2 beschrieben, eignet sich die ***why***-Provenance für unsere Zwecke in ProSA am besten. Hierfür ergänzen wir die einzelnen Quell-Tupel durch zusätzliche Tupel-IDs. Diese gehören immer zu den nicht-generalisierbaren Attributen vom Typ 4, da sie nach Abschluss der ProSA-Pipeline gelöscht werden. In der anonymisierten, (minimalen) Teil-Datenbank I^*_{anon} sind diese daher nicht mehr enthalten.

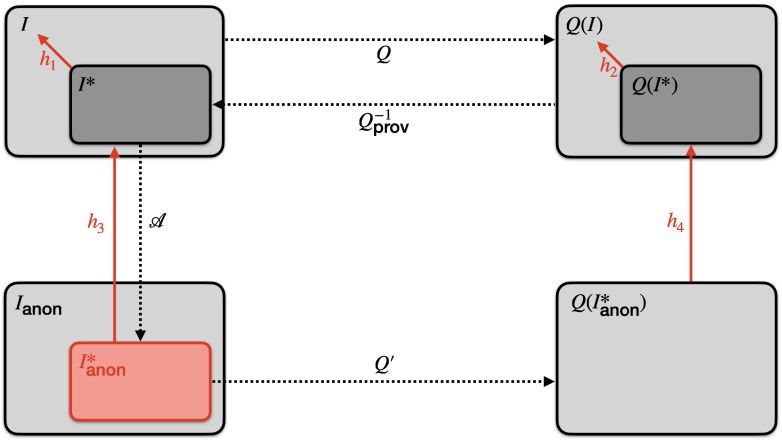


Abbildung 10.5. Schematische Darstellung der Anonymisierung durch Generalisierung und Intensionalisierung.

Wir entscheiden uns zudem für die Generalisierung als Anonymisierungsmethode. Diese liefert eine generalisierte, meist beschreibende anonymisierte Teil-Datenbank I^*_{anon} (in Abbildung 10.5 rot hervorgehoben), weshalb wir auch von einer Kombination aus Generalisierung und Intensionalisierung sprechen können. Bei der Anonymisierung $\mathcal{A}$ ist dabei stets darauf zu achten, dass ein geeigneter Homomorphismus existiert, welcher die anonymisierte Teil-Datenbank I^*_{anon} auf die nicht-anomymisierte Teil-Datenbank I^* abbildet, da wir nur so sicher gehen können, dass die Teil-Datenbank richtig berechnet wurde und die Plausibilität des rekonstruierten Anfrageergebnisses $Q'(I^*_{\text{anon}}$ gewährleistet werden kann. Insgesamt sind folgende Beziehungen gegeben, welche der Abbildung rot hervorgehoben sind:

- die (minimale) Teil-Datenbank I^* ist ein Ausschnitt der Quell-Datenbank I,
  d.h. $\exists$ Homomorphismus $h_1 : I^* \to I$;
- die anonymisierte, (minimale) Teil-Datenbank I^*_{anon} ist ein Ausschnitt der (minimalen) Teil-Datenbank I^*,
  d.h. $\exists$ Homomorphismus $h_3 : I^*_{\text{anon}} \to I^*$;
- das Anfrageergebnis auf der (minimalen) Teil-Datenbank $Q(I^*)$ ist ein Ausschnitt des Anfrageergebnisses auf der Quell-Datenbank $Q(I)$, d.h. $\exists$ Homomorphismus $h_2 : Q(I^*) \to Q(I)$;
- das Anfrageergebnis auf der anonymisierten (minimale) Teil-Datenbank $Q'(I^*_{\text{anon}})$ ist ein Ausschnitt des Anfrageergebnisses auf der (minimalen) Teil-Datenbank $Q(I^*)$, d.h. $\exists$ Homomorphismus $h_4 : Q'(I^*_{\text{anon}}) \to Q(I^*)$.

Fassen wir noch einmal zusammen:

- Liegt keine Anonymisierung vor, wird die (minimale) Teil-Datenbank zusammen mit der originalen Auswertungsanfrage Q veröffentlicht. Wir können in diesem Fall $Q(I^*) = Q(I)$ oder zumindest $Q(I^*) \preccurlyeq Q(I)$ garantieren.
- Liegt die (minimalen) Teil-Datenbank in anonymisierter Form I^*_{anon} vor, so veröffentlichen wir diese zusammen mit der transformierten Auswertungsanfrage Q' (siehe Abschnitt 8.3). In diesem Fall können wir $Q(I^*_{\text{anon}}) \preccurlyeq Q(I)$ garantieren.

10.7.2. Einbindung der Anonymisierung in ProSA

Für die Einbindung der Anonymisierung in ProSA stehen uns verschiedene Zeitpunkte innerhalb der ProSA-Pipeline (siehe Abbildung 10.6) zur Verfügung: Die Quell-Datenbank (**I**), das Anfrageergebnis (**II**), die rekonstruierte (minimale) Teil-Datenbank (**III**) sowie die Provenance-Anfrage (**IV**). Schauen wir uns die Vor- und Nachteile der einzelnen Zeitpunkte einmal im Detail an.

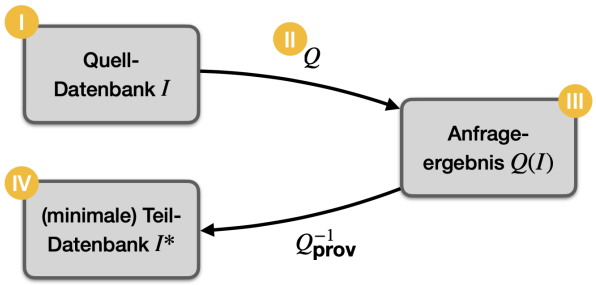


Abbildung 10.6. Mögliche Anonymisierungszeitpunkte innerhalb der ProSA-Pipeline.

I Quell-Datenbank Die Anonymisierung der Quell-Datenbank sowie der (minimalen) Teil-Datenbank hat zum Vorteil, dass in die bisherigen ProSA-Pipeline nicht eingegriffen werden muss. Die Anonymisierung wird vielmehr zu Beginn oder am Ende des ProSA-Ablaufs vorgeschaltet bzw. angehängt. In beiden Fällen sind keine weiteren Schritte notwendig, um die Anonymität konsistent gewährleisten zu müssen.

Eine Anonymisierung der Quell-Datenbank hat jedoch zur Folge, dass in der Regel erheblich mehr Daten anonymisiert werden würden, als für die Rekonstruktion eines veröffentlichten Anfrageergebnisses notwendig wären. Im Falle unseres Studierendenbeispiels bedeutet dies: Wird eine Anfrage an die Relation `Student` gestellt, muss die Relation `Participant` ebenfalls anonymisiert werden, auch wenn diese für die Auswertung der Anfrage nicht relevant ist. Es entsteht somit ein erheblicher Overhead. Um dies zumindest zum Teil zu umgehen, kann die Datenmenge zuvor auf die für die Auswertung der Anfrage relevanten Relationen eingeschränkt werden, beispielsweise durch die relationen-basierte ***where***-Provenance. Dieser Ansatz hat jedoch zum Nachteil, dass die Anfrage mehrfach ausgeführt werden müsste. Die ProSA-Pipeline müsste entsprechend um einen weiteren Chase-Durchlauf zur Bestimmung der ***where***-Provenance erweitert werden.

Auch die Auswertung der Anfrage auf der originalen Quell-Datenbank müsste separat erfolgen. Denn, soll auf einem Quasi-Identifikator selektiert werden, so muss dieser im Original vorliegen. Läge die gesamte Relation anonymisiert vor, wäre dies nicht mehr möglich. Wir benötigen von einigen Relationen also Kopien. Zudem müssen Teile der Anfrage so umgeschrieben werden, dass für die interne Auswertung der Anfrage, beispielsweise bei einer Selektion oder einem natürlichen Verbund, weiterhin die originalen Attributwerte verwendet werden. Andererseits sind Quasi-Identifikatoren „Schwachstellen“ innerhalb der Datenbank, welche zur Einhaltung von Privacy grundsätzlich anonymisiert werden müssen. Der technische Overhead, vor jeder Anfrage eine neue Kopie anzulegen und diese hinreichend zu anonymisieren, wäre auf ProSA übertragen jedoch unverhältnismäßig. ProSA könnte in diesem Fall nur zusätzlich zur Bestimmung der (anonymisierten, minimalen) Teil-Datenbank, nicht jedoch zur allgemeinen Auswertung einer Forschungsfrage genutzt werden.

II Anfrage Eine weiterer Ansatz ist das Anonymisieren der Anfrage. Hierfür kann die Anfrage beispielsweise durch Differential Privacy verrauscht werden. So kann etwa eine Attribut-Konstanten-Selektion durch eine Bereichsselektion ersetzt werden. Die Anfrage liefert dann ein verrauschtes Ergebnis, welches mehr Tupel enthält als das originale Anfrageergebnis. Da in unserem Anwendungsfall sowohl die Anfrage als auch das Anfrageergebnis veröffentlicht werden, ist dieser Ansatz in ProSA nicht anwendbar.

Neben der Anfrage kann auch die zugehörige Provenance-Anfrage anonymisiert werden. Hierfür können wir uns die in Abschnitt 3.4 vorgestellte Provenance-Hierarchie ***where*** $\preccurlyeq$ ***why*** $\preccurlyeq$ ***how*** zu Nutze machen. Statt der ***how***-Provenance könnten wir so die ***why***- oder ***where***-Provenance nutzen. Die „schwächste" Variante ist die relationen-orientierte ***where***-Provenance, welche lediglich die zugehörigen Relationennamen speichert. Im Fall unseres Studierendenbeispiels etwa die Relationen `Student` oder `Participant`. Sie stellt somit im weitesten Sinne eine Anonymisierung der Provenance-Anfragen dar.

Da die relationen-orientierte ***where***-Provenance lediglich die Namen der zugehörigen Relationen, jedoch keine weitere Informationen über die Tupel selbst liefert[8], bietet sie in ProSA keine großen Mehrwert. In ProSA nutzen wir daher die ***why***-Provenance. Diese speichert die Tupel-Indentifikatoren aller an der Berechnung des Ergebnis beteiligten Tupel in einer Zeugenbasis. Eine geeignete Generalisierung existiert hierfür jedoch nicht, da eine Verallgemeinerung der Tupel- bzw. Provenance-IDs auf ihre zugehörigen Relationennamen den Mehrwert der ***why***-Provenance – hierzu gehören insbesondere die Duplikatrekonstruktion sowie die Garantie von natürlichen Verbünden – negieren würden. Eine Anonymisierung der Provenance-Anfrage ist im Falle von ProSA somit nicht zielführend.

III Anfrageergebnis Neben der Quell-Datenbank kann auch das Anfrageergebnis anonymisiert werden. Das Anfrageergebnis bietet sich insbesondere deshalb an, weil es in der ProSA-Pipeline bereits weiter hinten angesiedelt ist. In unserem Anwendungsfall, dem Forschungsdatenmanagement, ist es jedoch zwingend notwendig, dass Anfrageergebnis so original getreu wie möglich zu halten. Unser Ziel ist es schließlich das in einem Konferenz- oder Journal-Artikel veröffentlichte Anfrageergebnis auf Plausibilität zu überprüfen und sofern möglich zu rekonstruieren. Eine Anonymisierung des Anfrageergebnisses ergibt in ProSA somit keinen Sinn.

IV (Minimale) Teil-Datenbank Wie oben beschrieben, muss bei der Anonymisierung der rekonstruierten Teil-Datenbank nicht in die bisherigen ProSA-Pipeline eingegriffen werden. Der entscheidende Vorteil besteht darin, dass alle Berechnungen, die auf Provenance-Annotationen aufbauen, bereits abgeschlossen sind und die Daten selbst nicht weiter verarbeitet werden müssen. Die rekonstruierte Teil-Datenbank kann somit problemlos anonymisiert werden. Der hieraus resultierende Informationsverlust verhindert gegebenenfalls die Reproduzierbarkeit bzw. Replizierbarkeit des Anfrageergebnisses, erlaubt aber immer noch einen Plausibilitätsnachweis insofern, dass $Q(I^*_{\text{anon}}) \preccurlyeq Q(I)$ für eine Datenbankinstanz I und ihre anonymisierte (minimale) Teil-Datenbankinstanz I^*_{anon} ist. Unserem Ziel der Replizierbarkeit bzw. Reproduzierbarkeit eines veröffentlichten Forschungsergebnisses sind wir so unter den gegebenen Privacy-Aspekte ziemlich nahe gekommen.

Auch der technische Aufwand für die Anonymisierung der (minimalen) Teil-Datenbank hält sich in Grenzen. Anders als bei einer Anonymisierung der Quell-Datenbank ist die Teil-Datenbank bereits auf alle relevanten Relationen und Tupel eingeschränkt und alle „überflüssigen" Attributwerte sind bereits durch Nullwerte, generiert durch die existenz-quantifizierten Variablen der inversen Anfrage, ersetzt worden. Die Anonymisierung der (minimalen) Teil-Datenbank ist in ProSA daher die beste Möglichkeit Privacy-Aspekte zu integrieren.

Wahl des Anonymisierungszeitpunktes Schauen wir uns die verschiedenen Anonymisierungszeitpunkte in der ProSA-Pipeline noch einmal am konkreten Beispiel an (siehe Tabelle 10.13). Seien dazu die Anfrage `SELECT AVG(grade) FROM Grade GROUP BY studID` als Q sowie die Relation `Grade` aus Tabelle **(a)** mit den Tupeln $(13, x, 1.0)$, $(27, x, 2.3)$, $(27, x, 1.7)$, $(115, x, 4.0)$, $(115, x, 5.0)$ und $(115, x, 1.5)$ gegeben. Dann liefert die Invertierung des Anfrageergebnisses $Q(I)$ die (minimale) Teil-Datenbank I^* bestehend aus den Tupeln $(13, x, 1.0)$ (1x), $(27, x, 2.0)$ (2x) und $(115, x, 3.5)$ (3x), dargestellt in Tabelle 10.13a. Da eine Anonymisierung der ***why***-Provenance nicht ohne den

[8] Die Definition der relationen-orientierten ***where***-Provenance ist nicht eindeutig. Wir erlauben an dieser Stelle keine Schlussfolgerungen vom Relationennamen auf die originale Datenbankinstanz. Die ***where***-Provenance kann in unserem Sinne also nicht für eine Rekonstruktion konkreter Attributwerte genutzt werden.

studID	moduleID	grade	
13	x	1.0	t_1
27	x	2.3	t_2
27	x	1.7	t_3
115	x	4.0	t_4
115	x	5.0	t_5
115	x	1.5	t_6

⟶

studID	AVG(grade)	
13	1.0	$\{\{t_1\}\}$
27	2.0	$\{\{t_2, t_3\}\}$
115	3.5	$\{\{t_4, t_5, t_6\}\}$

⟶

studID	moduleID	grade
13	x	1.0
27	x	2.0
27	x	2.0
115	x	3.5
115	x	3.5
115	x	3.5

(a) Ohne Anonymisierung

studID	moduleID	grade	
13	x	A	t_1
27	x	B	t_2
27	x	B	t_3
115	x	D	t_4
115	x	E	t_5
115	x	A	t_6

⟶

studID	AVG(grade)	
13	A	$\{\{t_1\}\}$
27	B	$\{\{t_2, t_3\}\}$
115	C	$\{\{t_4, t_5, t_6\}\}$

⟶

studID	moduleID	grade
13	x	A
27	x	B
27	x	B
115	x	C
115	x	C
115	x	C

(b) Anonymisierung der Quell-Datenbank (I)

studID	moduleID	grade	
13	x	1.0	t_1
27	x	2.3	t_2
27	x	1.7	t_3
115	x	4.0	t_4
115	x	5.0	t_5
115	x	1.5	t_6

⟶

studID	AVG(grade)	
13	1.0	$\{\{t_1\}\}$
27	2.0	$\{\{t_2, t_3\}\}$
115	3.5	$\{\{t_4, t_5, t_6\}\}$

⟶

studID	moduleID	grade
13	x	1.0
27	x	2.0
27	x	2.0
115	x	3.5
115	x	3.5
115	x	3.5

(c) Anonymisierung der Provenance-Anfrage (II)

studID	moduleID	grade	
13	x	1.0	t_1
27	x	2.3	t_2
27	x	1.7	t_3
115	x	4.0	t_4
115	x	5.0	t_5
115	x	1.5	t_6

⟶

studID	AVG(grade)	
13	A	$\{\{t_1\}\}$
27	B	$\{\{t_2, t_3\}\}$
115	C	$\{\{t_4, t_5, t_6\}\}$

⟶

studID	moduleID	grade
13	x	A
27	x	B
27	x	B
115	x	C
115	x	C
115	x	C

(d) Anonymisierung des Anfrageergebnisses (III)

studID	moduleID	grade	
13	x	1.0	t_1
27	x	2.3	t_2
27	x	1.7	t_3
115	x	4.0	t_4
115	x	5.0	t_5
115	x	1.5	t_6

⟶

studID	AVG(grade)	
13	1.0	$\{\{t_1\}\}$
27	2.0	$\{\{t_2, t_3\}\}$
115	3.5	$\{\{t_4, t_5, t_6\}\}$

⟶

studID	moduleID	grade
13	x	A
27	x	B
27	x	B
115	x	C
115	x	C
115	x	C

(e) Anonymisierung der rekonstruierten (minimalen) Teil-Datenbank (IV)

Tabelle 10.13. Anonymisierungszeitpunkte am konkreten Beispiel: Gegeben sei die Anfrage `SELECT AVG(grade) FROM Grade GROUP BY studID` sowie die Relation `Grade` aus Tabelle **(a)**. Die zugehörigen Provenance-IDs sind jeweils blau und die Anonymisierung rot hervorgehoben.

Verlust sämtlicher Informationen möglich ist (siehe Abschnitt 8.2), ist die berechnete (minimale) Teil-Datenbank ebenfalls nicht anonymisiert (siehe Abbildung 10.13c). Die Anonymisierung der Quell-Datenbank (Tabelle 10.13b), des Anfrageergebnisses (Tabelle 10.13d) sowie der (minimalen) Teil-Datenbank (siehe Tabelle 10.13e) hingegen liefern jeweils eine anonymisierte (minimale) Teil-Datenbank aus **(a)**.

Fassen wir noch einmal zusammen: Die Anonymisierung der Quell-Datenbank ist möglich, der Aufwand hierfür ist jedoch verhältnismäßig hoch. Da die Anfrage sowie das Anfrageergebnis in der untersuchten wissenschaftlichen Publikation ohnehin veröffentlicht sind, ergibt ihre Anonymisierung im Kontext von ProSA keinen Sinn. Auch die Anonymisierung der Provenance-Annotationen ist, wie in Abschnitt 8.2 beschrieben, insbesondere im Falle der ***why***- und ***how***-Provenance wenig zielführend. Es bleibt somit nur die Anonymisierung der (minimalen) Teil-Datenbank. Da diese am Ende der ProSA-Pipeline steht, hat eine Anonymisierung keinen Einfluss mehr auf die Provenance-Annotationen, sodass sich Provenance- und Privacy-Aspekte hier so wenig wie möglich widersprechen können (siehe Abschnitt 8.2).

10.8. Chase&Backchase in ProSA

Wie in Abschnitt 10.1 angeschnitten, wird in ProSA sowohl die Berechnung der (minimalen) Teil-Datenbank als auch die Einbindung der Evolution vom Chase übernommen. Beide Berechnungen können dabei mit den in Abschnitt 6.4 vorgestellten Chase&Backchase-Verfahren verarbeitet werden.

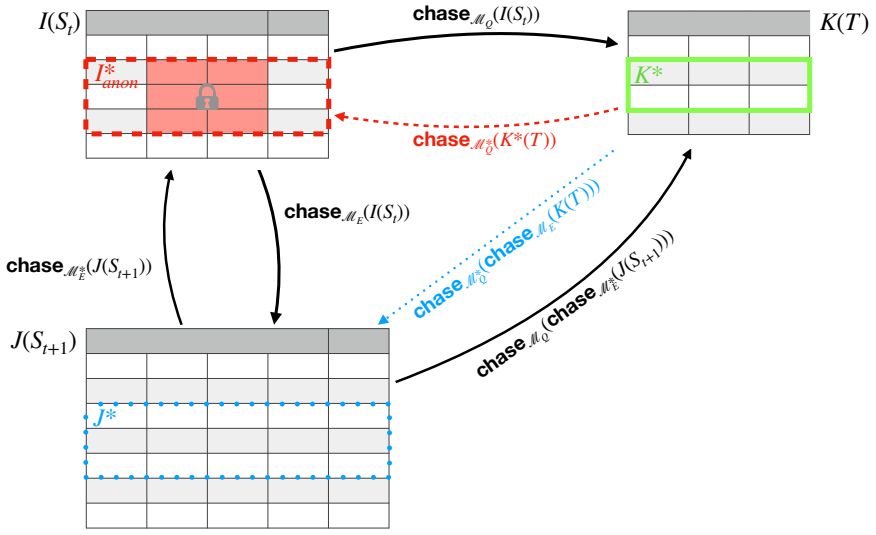


Abbildung 10.7. Chase-Anwendung bei der Berechnung der (minimalen) Teil-Datenbanken I^* und J^* (vgl. Abbildung 5.2).

Berechnung der (minimalen) Teil-Datenbank Für die Berechnung der (minimalen) Teil-Datenbank nutzt ProSA den *Provenance-aware* Chase&Backchase (siehe Definition 6.11). Bei dieser erweiterten Variante des Chase&Backchase werden in der Chase-Phase für jedes Tupel t der Ergebnisinstanz K zusätzliche Provenance-Informationen bestimmt. Hierzu gehören die Zeugenbasis $W_{Q,d}(t)$ (siehe Definition 3.37) sowie zusätzliche Side Tables (siehe Definition 6.10) mit deren Hilfe in der Backchase-Phase die Tupel der (minimalen) Teil-Datenbank I^* möglichst exakt rekonstruiert werden können. Die Zeugenbasis gibt an, welche Quell-Tupel an der Auswertung der Anfrage Q beteiligt sind, und Attributwerte, welche im Laufe der Anfrageauswertung beispielsweise durch Aggregation verloren gehen, werden in den Side Tables abgespeichert. Weitere Details hierzu können in Abschnitt 6.4 nachgelesen werden.

Sei Q eine Anfrage, formalisiert als eine Menge Σ_P von (s-t) tgds (siehe Abschnitt 10.2) und erweitert um zusätzliche Provenance-Informationen P. Seien weiter I eine Datenbankinstanz über dem Quell-Schema S_t und $K = Q(I)$ das Anfrageergebnis über dem Ziel-Schema T. Dann ist die entsprechende Schemaabbildung definiert als $\mathcal{M}_Q = (S_t, T, \Sigma_P)$. Sei Q^{-1} die zu Q inverse Anfrage, dargestellt über die Schemaabbildung $\mathcal{M}^*_Q = (T, S_t, \Sigma_P^{-1})$. Dann liefert der Chase&Backchase (siehe Abbildung 10.7):

$$\text{Chase}_{\mathcal{M}^*_Q}(\text{Chase}_{\mathcal{M}_Q}(I)) = \text{Chase}_{\mathcal{M}^*_Q}(K) = I^*.$$

Neben dieser erweiterten Chase&Backchase-Variante berechnet das ProSA-System auch eine (minimale) Teil-Datenbank ohne zusätzliche Provenance-Informationen (siehe Definition 6.9). Diese dient lediglich zum Vergleich und wird im weiteren Verlauf der ProSA-Pipeline nicht weiter verwendet. Sie ist im vierten Tab unten rechts unter Button Without Provenance zu finden (siehe Abbildung 10.10).

Die Berechnung der ***where***-, ***why***- und ***how***-Provenance erfolgt in der Chase-Phase. Ausgegeben werden sie jedoch im vierten Tab oben rechts (siehe Abbildung 10.10, türkis hervorgehoben). Sie dienen insbesondere der Vollständigkeit. Wie in Abschnitt 10.5 erörtert, genügt für die bestmögliche Rekonstruktion der (minimalen) Teil-Datenbank die Verwendung der ***why***-Provenance.

Verarbeitung der Evolution Für die Verarbeitung der Evolution verzichten wir auf zusätzliche Provenance-Informationen und nutzen erneut den Chase&Backchase nach Definition 6.9. Sei hierfür die Evolution E und ihre Inverse E^*. Diese werden wiederum als Mengen von (s-t) tgds transformiert und als Schemaabbildungen $\mathcal{M}_E$ und $\mathcal{M}_E^*$ interpretiert. Seien hierfür S_t und S_{t+1} zwei Schemata einer temporalen Datenbank. Dann liefert der Chase&Backchase für $\mathcal{M}_E = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}_E^* = (S_{t+1}, S_t, \Sigma^{-1})$:

$$\textsc{Chase}_{\mathcal{M}_E}(\textsc{Chase}_{\mathcal{M}_E^*}(J)) = \textsc{Chase}_{\mathcal{M}_E}(I) = J^*.$$

Chase&Backchase in der ProSA Erweitern wir das Chase&Backchase-Verfahren zur Berechnung der (minimalen) Teil-Datenbank auf temporale Datenbanken, so erhalten wir den in den Abbildung 10.7 aufgezeigten Ablauf. Hierfür chasen wir zunächst E^* mit J, um die Datenbankinstanz auf den alten Zustand zum Zeitpunkt t zurückzuführen. Anschließend berechnen wir wie mit Hilfe des Provenance-aware Chase&Backchase die (minimale) Teil-Datenbank, wie in Kapitel 6 beschrieben, und chasen abschließend die so erhaltene Teil-Datenbank mit der Evolution E. Insgesamt ergibt sich so folgender Ablauf:

$$\begin{aligned}
\textsc{Chase}_{\mathcal{M}_E}(\textsc{Chase}_{\mathcal{M}_Q^*}(\textsc{Chase}_{\mathcal{M}_Q}(\textsc{Chase}_{\mathcal{M}_E^*}(J)))) &= \textsc{Chase}_{\mathcal{M}_E}(\textsc{Chase}_{\mathcal{M}_Q^*}(\textsc{Chase}_{\mathcal{M}_Q}(I))) \\
&= \textsc{Chase}_{\mathcal{M}_E}(\textsc{Chase}_{\mathcal{M}_Q^*}(K)) \\
&= \textsc{Chase}_{\mathcal{M}_E}(I^*) \\
&= J^*.
\end{aligned}$$

Die Berechnung der (minimalen) Teil-Datenbank J^* zum Zeitpunkt $t+1$ wird somit durch doppelte Ausführung des Chase&Backchase realisiert. Wir benötigen hierfür lediglich die Datenbankinstanz J zum Zeitpunkt $t+1$, die Evolution E der Datenbank vom Zustand t nach $t+1$ (dargestellt als SMO) sowie die originale SQL-Anfrage Q. Durch die geschickte Implementierung von ChaTEAU und ProSA (siehe Abschnitt 9.2.1 und Abschnitt 10.9) sind alle durchgeführten Kompositionen von (s-t) tgds stets erster Ordnung.

10.9. Implementierung des ProSA-Systems

ProSA ist als Maven-Projekt in Java 11 implementiert. Es bietet eine Benutzeroberfläche, die die Auswertung einfacher konjunktiver Anfragen sowie SQL-Anfragen über den Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM`, `AVG` ermöglicht. Durch geeignete Erweiterung des ProSA-Parsers ist es zudem möglich beliebige Anfragen zu verarbeiten, sofern diese als Menge von (s-t) tgds dargestellt werden können (siehe Abschnitt 11.2).

Wie in Abschnitt 10.1 beschrieben, wird die Anfrageauswertung in ProSA durch den Chase übernommen, welcher in ChaTEAU separat implementiert ist (siehe Kapitel 9). Das ProSA-System selbst übernimmt das Parsen der Anfrage, das Hinzufügen von Provenance-Informationen, die Vorbereitung der Chase-Ausführungen sowie die Interpretation der Chase-Ergebnisse. Zudem wird die bestimmte minimale Teil-Datenbank auf Korrektheit geprüft und anschließend durch Generalisierung gemäß k-Anonymität und l-Diversität anonymisiert. Schauen wir uns als nächstes die Implementierung des ProSA-Systems an. Für die Software, Beispiele sowie weitere Informationen zu ChaTEAU und ProSA sei auf die zugehörigen Git-Repositories verwiesen:

- ChaTEAU: `https://git.informatik.uni-rostock.de/ta093/chateau-demo`
- ProSA: `https://git.informatik.uni-rostock.de/ta093/prosa-demo`

Als Schnittstelle dient die in den Abbildungen 10.8 bis 10.11 dargestellte ProSA-GUI. Sie besteht aus insgesamt sieben Tabs: einem Konfigurations-Tab, einem Logging-Tab sowie fünf Tabs zur Bestimmung der (minimalen) Teil-Datenbank. Diese fünf verarbeitenden Tabs bestehen stets aus zwei Teilen. Links im Fenster steht jeweils die Eingabe, die in der Regel aus einer Datenbankinstanz, einem Datenbankschema sowie einer Menge von Abhängigkeiten besteht. Das berechnete Ergebnis ist jeweils rechts im Fenster zu sehen. Hierzu gehört etwa die transformierte SQL-Anfrage Q, das Anfrageergebnis $Q(I)$, die rekonstruierte, evolutionierte oder anonymisierte Teil-Datenbank sowie weitere Provenance-Informationen. Nicht nutzbare Tabs und Buttons werden jeweils ausgegraut.

Ein- und Ausgabe Im ersten Tab (siehe Abbildung 10.8, orange hervorgehoben) wird eine Verbindung zur Datenbank hergestellt und — falls notwendig — eine Beschreibung der Schema-Evolution in Form einer separaten XML-Datei eingelesen (siehe XML-Datei D.6). Im dritten Tab (siehe Abbildung 10.10, orange hervorgehoben) kann dann eine SQL-Anfrage gestellt werden, welche in den folgenden Tabs verarbeitet wird. Ist eine Anonymisierung gewünscht, kann im sechsten Tab die Definition einer zugehörige Domänengeneralisierungshierarchie hochgeladen werden (siehe Abbildung 10.11, orange hervorgehoben).

Ausgegeben wird die (minimale) Teil-Datenbank zum einen im Backchase-Tab nach ihrer Berechnung (siehe Abbildung 10.10, rot hervorgehoben), nach der Ausführung der Schema-Evolution (siehe Abbildung 10.9, blau hervorgehoben) sowie nach ihrer Anonymisierung (siehe Abbildung 10.11, türkis hervorgehoben). Die Speicherung des Anfrageergebnisses, der Provenance-Informationen sowie der verschiedenen (minimalen) Teil-Datenbank ist jederzeit über den Button mit dem Pfeil ↓ möglich (siehe Abbildung 10.10).

Konfiguration ProSA ermöglicht die Anbindung an eine PostgreSQL-Datenbank. Mit einer entsprechenden Erweiterung des SQL-Parsers [RSZ21; Kav22] ist aber auch eine Anbindung an andere Datenbankmanagementsysteme wie etwa MySQL oder Oracle denkbar, da die zugrundeliegende Theorie (siehe Abschnitt 10.2) unabhängig von der Wahl des Datenbankmanagementsystems ist.

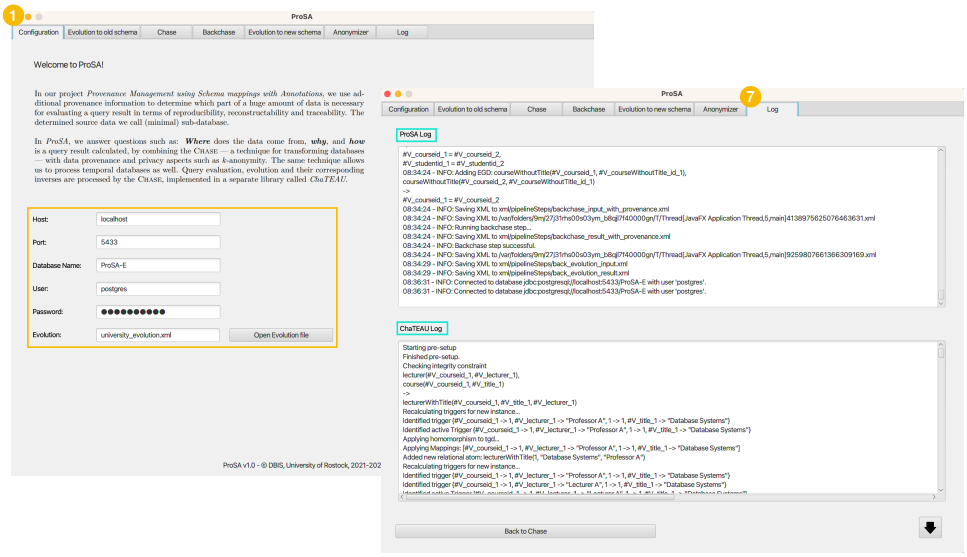


Abbildung 10.8. Konfiguration und Logging in der ProSA-GUI.

Logging ProSA liefert ein separates Logging für den allgemeinen Ablauf von ProSA sowie für die einzelnen Chase-Ausführungen durch ChaTEAU. Die entsprechenden Loggings können im Logging-Tab (siehe Abbildung 10.8, türkis hervorgehoben) eingesehen und heruntergeladen werden.

Schema-Evolution Für die Integration der Schema-Evolution müssen die zugehörigen Operatoren als Menge von (s-t) tgds formalisiert und als XML-Datei bereitgestellt werden. Ein Beispiel für eine solche Eingabe-Datei ist im Anhang D.1 zu finden. Die Darstellungen der Schemaevolutionsoperatoren (SMOs) als Menge von (s-t) tgds kann in Abschnitt 7.2 nachgelesen werden. Die Invertierung der SMOs erfolgt im ProSA-Invertierer (siehe Abschnitt 10.3) und muss nicht extra angegeben werden.

Wie in Abschnitt 10.1 beschrieben, wird die eingelesene Evolutions-Datei zunächst als Menge von (s-t) tgds transformiert und invertiert. Nun wird die originale Datenbank, welche aktuell zum Zeitpunkt $t+1$ vorliegt auf den Zeitpunkt t zurückgesetzt. Hierfür wird zunächst die originale Datenbankinstanz I_{t+1} mit der inversen Evolution (siehe Abbildung 10.9, orange hervorgehoben) gechaset. Die so erhaltene Instanz I_t bildet nun die Grundlage für die in Tab 3 und 4 durchgeführte Anfrageauswertung. Abschließend wird die (minimalen) Teil-Datenbank im fünften Tab noch einmal gechaset, dieses mal mit der inversen Evolution. Das Ergebnis ist in Abbildung 10.9 blau hervorgehoben.

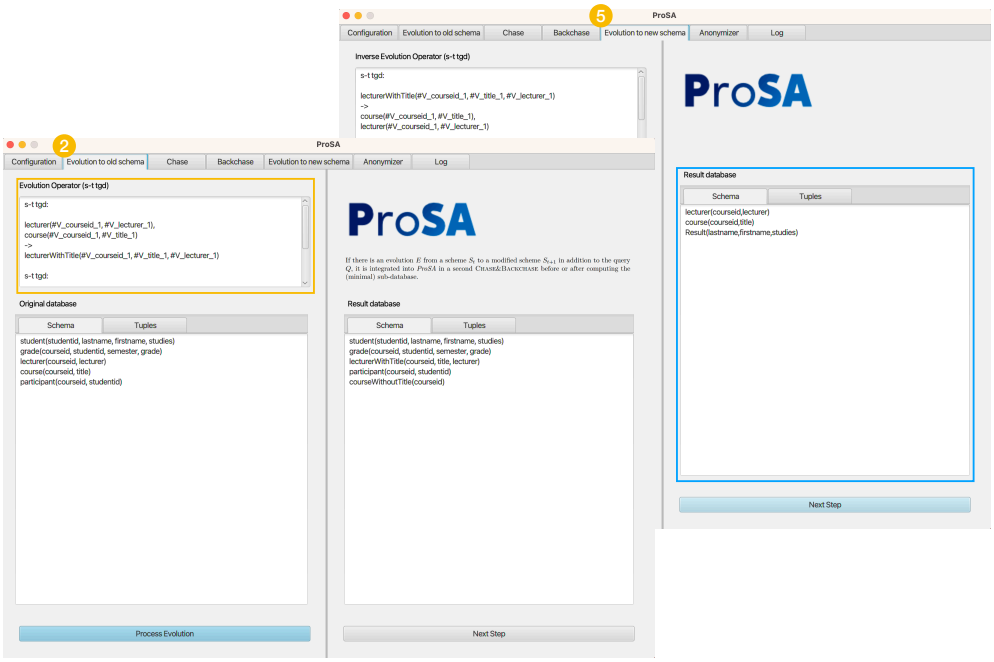


Abbildung 10.9. Evolution in der ProSA-GUI.

Chase&Backchase Die in Abbildung 10.10 zu sehenden Chase- und Backchase-Tabs enthalten neben der SQL-Anfrage als Eingabe (orange hervorgehoben), dem Anfrageergebnis (grün hervorgehoben) und der rekonstruierten (minimalen) Teil-Datenbank (rot hervorgehoben) noch weitere interessante Informationen. Hierzu gehören die Auswertungsanfrage dargestellt als Menge von (s-t) tgds (dritter Tab, oben rechts), die Bestimmung des Chase-Inversentyps sowie die Zeugenlisten, Zeugenbasen und Provenance-Polynome der ***where***-, ***why***- und ***how***-Provenance (vierter Tab, türkis hervorgehoben). Der Chase-Inversentyp wird unabhängig vom Chase&Backchase durch Parsen der SQL-Anfrage ermittelt. Während der linke Typ immer garantiert werden kann, ist der rechte Typ nur mit zusätzlichen Provenance-Informationen möglich.

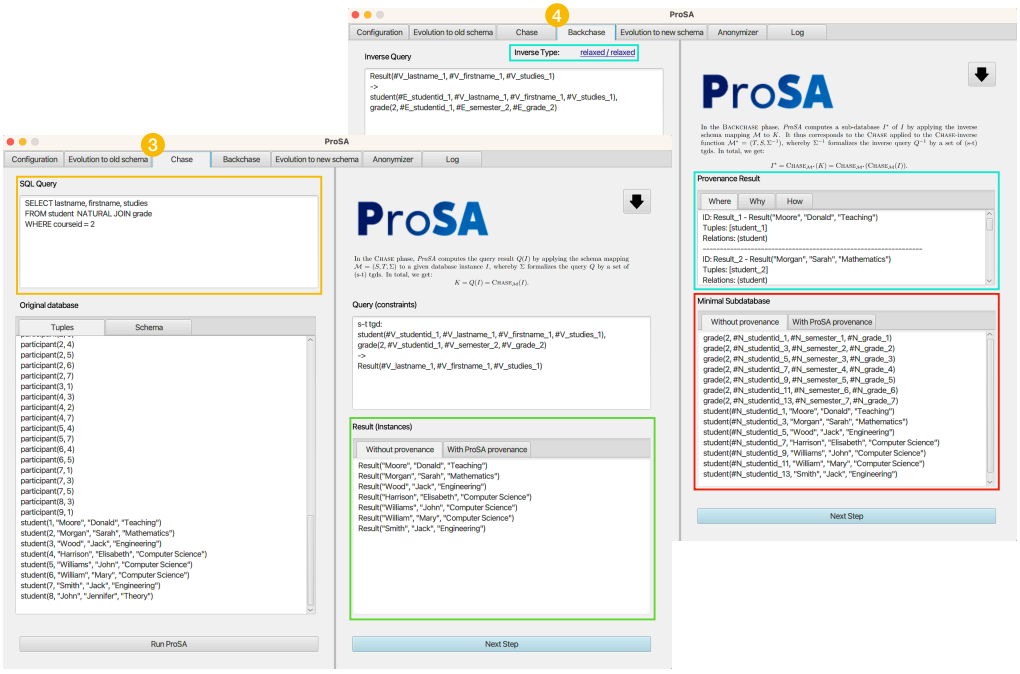


Abbildung 10.10. Chase&Backchase in der ProSA-GUI.

Des Weiteren berechnet das ProSA-System zusätzlich noch die (minimale) Teil-Datenbank ohne zusätzliche Provenance-Informationen (siehe unten rechts im vierten Tab, rot hervorgehoben). Diese dient jedoch lediglich dem Vergleich und wird im weiteren Verlauf der ProSA-Pipeline nicht mehr verwendet. Evolutioniert, anonymisiert und ausgegeben wird die Provenance-basierte (minimale) Teil-Datenbank.

In Anhang D ist jeweils ein Beispiel für das Ergebnis nach dem Parsen der Anfrage (siehe XML-Datei D.7), nach der Berechnung des Anfrageergebnisses inklusive Provenance (siehe XML-Datei D.9) sowie nach Rekonstruktion der (minimalen) Teil-Datenbank (siehe XML-Datei D.10) gegeben. Diese entsprechen jeweils der Eingabe in die Chase-Phase, der Ausgabe der Chase- bzw. Ausgabe der Backchase-Phase sowie dem Ergebnis der Backchase-Phase.

Anonymisierung Die Eingabe im Anonymisierungs-Tab (siehe Abbildung 10.11, orange hervorgehoben) bietet verschiedene Einstellungsmöglichkeiten zur Wahrung der k-Anonymität. So können neben dem gewünschten k-Wert auch die Distinct Ratio — Wert zwischen 0 und 1 — sowie die identifizierenden Attribute pro Relation eingestellt werden. Abschließend kann über den Button Select hierarchies file noch die passende Domänengeneralisierungshierarchie hochgeladen werden. Ein Beispiel hierzu ist in Anhang D unter D.8 zu finden.

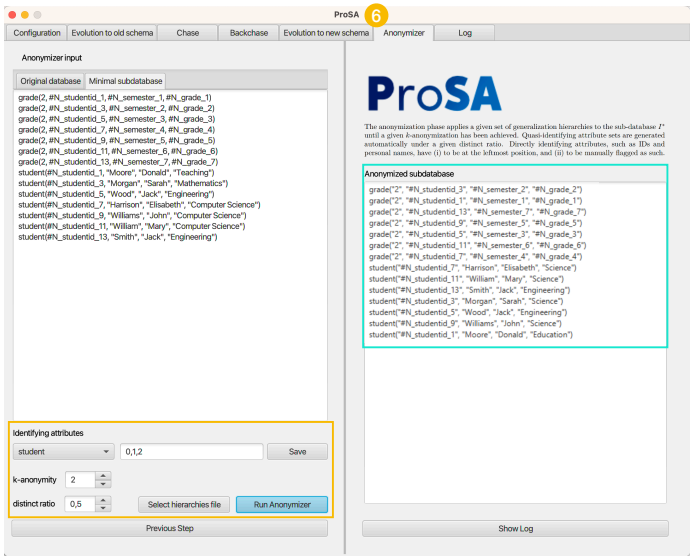


Abbildung 10.11. Anonymisierung in der ProSA-GUI.

Wie in den Abbildungen 10.8 bis 10.11 zu sehen ist, wird die ProSA-Pipeline für folgenden Eingaben fehlerfrei durchlaufen:

- Quell-Instanz I wie in den Tabelle A.1 gegeben
- Auswertungsanfrage Q: `SELECT lastName, firstName, studies FROM Student NATURAL JOIN Grade WHERE coursID` = '2'
- Evolution E: `MOVE title FROM Course INTO Lecturer`
- Anonymisierung: Domänengeneralisierunghierarchie gegeben wie in der XML-Datei D.8

Wir konnten somit sowohl theoretisch als auch praktisch nachweisen, dass das in der vorliegenden Arbeit vorgestellt Konzept ProSA funktioniert.

Eigene sowie betreute Arbeiten

Der konzeptuelle Ablauf von ProSA sowie die zugrundeliegenden Ideen können jedoch in den folgenden beiden Veröffentlichungen nachgelesen werden:

- [AHH22] **Tanja Auge**, Moritz Hanzig, Andreas Heuer: ProSA Pipeline — Provenance Conquers the Chase. In: *Advances in Databases and Information Systems* (ADBIS 2022), pp. 89–98, 2022
- [AH19] **Tanja Auge**, Andreas Heuer: ProSA — Using the Chase for Provenance Management. In: *Advances in Databases and Information Systems* (ADBIS 2019), pp. 357–372, 2019
- [Aug20] **Tanja Auge**: ProSA Pipeline — Provenance Conquers the Chase. In: PhD@VLDB 2020

Die Implementierung von ProSA ist in Kooperation mit verschiedenen studentischen Projekten und Abschlussarbeiten entstanden. Hierzu gehören:

- [För+21] Leonie Förster, Melinda Heuer, Judith-Henrike Overath, Anne-Sophie Waterstradt, Anja Wolpers: Data Provenance. *KSWS*[9], SS2021
- [RSZ21] Ivo Kavisanczki, Tobias Rudolph, Tom Siegl, Marian Zuska: sql2sttgd. *KSWS*[10], SS2021
- [Han22] Moritz Hanzig: Ein Framework für ProSA. *Bachlorarbeit*, 2022
- [Kav22] Ivo Kavisanczki: Erweiterung des ProSA-Parsers. *Bachelorarbeit*, 2022
- [Sch22] Nic Scharlau: Anonymisierung von Data Provenance in ProSA. *Masterarbeit*, 2022
- [Spo22] Dennis Spolwind: Inverse Anfragen in ProSA. *Masterarbeit*, 2022

Der Fokus der einzelnen Arbeiten lag hierbei stets auf der Entwicklung und Implementierung einer konkreten Fragestellung. Die Ideen und Theorie hinter ProSA sind unter anderem in den beiden genannten Veröffentlichungen zu finden.

Einige der in diesem Kapitel vorgestellten Ergebnisse sind bisher noch nicht anderweitig veröffentlicht. Hierzu gehören insbesondere die konkrete Umsetzung der Transformation der SQL-Anfrage, der Invertierung von Anfragen, die Verarbeitung der einfachen Aggregatfunktionen sowie die Erweiterung des Chase&Backchase.

[9] KSWS: *Komplexe Software Systeme*, Modul an der Universität Rostock im Bachelorstudiengang Informatik
[10] KSWS: *Komplexe Software Systeme*, Modul an der Universität Rostock im Bachelorstudiengang Informatik

Teil IV.

Schlussbemerkungen

11. Zusammenfassung und Ausblick

Fassen wir abschließend die wichtigsten Ergebnisse der vorliegenden Dissertation *Provenance Management unter Verwendung von Schemaabbildungen mit Annotationen* (ProSA) noch einmal zusammen (siehe Abschnitt 11.1) und schließen mit einem kurzen Ausblick auf mögliche Erweiterungen und Fortführungen der in ProSA entwickelten Techniken (siehe Abschnitt 11.2).

11.1. Zusammenfassung

Ziel der vorliegenden Arbeit *Provenance Management unter Verwendung von Schemaabbildungen mit Annotationen* ist die Entwicklung und Erweiterung des Chase&Backchase zur Unterstützung von Provenance Management-Strategien im Forschungsdatenmanagement. Konkret haben wir die ***why***-Provenance sowie zusätzliche Annotationen zur Behandlung von verlorenen Attributwerten oder Null-Tupeln eingesetzt. Auch Duplikate, welche während der Anfrageauswertung sowie deren Invertierung auftreten, können so rekonstruiert und Schemaänderungen, welche insbesondere bei mehrjährigen Forschungsprojekten auftreten, integriert werden.

Im Zusammenhang mit einem guten Forschungsdatenmanagement fällt häufig der Begriff der FAIR-Prinzipien. Diese sollen garantieren, dass Forschungsdaten auffindbar (**F**indable), zugänglich (**A**ccessible), kompatibel (**I**nteroperable) sowie wiederverwendbar (**R**eusable) sind. Im weitesten Sinne kann das vierte Prinzip auch so verstanden werden, dass zu einer Publikation auch stets die originalen Daten mit veröffentlicht werden sollen. Eine akkurate Angabe der Originaldaten ist jedoch nicht immer möglich oder gewünscht. Mit Hilfe von Data Provenance lässt sich jedoch feststellen, welcher Teil der primären Forschungsdaten minimal gespeichert werden muss, um die Reproduzierbarkeit, Rekonstruierbarkeit oder Plausibilität einer veröffentlichten Auswertung zu gewährleisten. Die (minimale) Teil-Datenbank soll dabei durch Provenance und zusätzliche Annotationen maximiert und bzgl. Privacy minimiert werden. Zudem soll die (minimale) Teil-Datenbank unter (Schema-)Änderungen konsistent bleiben. Insgesamt ergaben sich hieraus die drei bekannten Fragen:

(I.) Wie lässt sich der minimale Teil der ursprünglichen Forschungsdatenbank berechnen, der für die Replizierbarkeit bzw. Reproduzierbarkeit der Auswertungsergebnisse dauerhaft gespeichert werden muss?
→ siehe Kapitel 6

(II.) Wie lassen sich die Theorien zu Data Provenance und Schema-Entwicklung vereinheitlichen?
→ siehe Kapitel 7

(III.) Wie lassen sich Data Provenance und Privacy-Aspekte, d.h. Aspekte des Datenschutzes, im Falle der Anfrage-Invertierung kombinieren? → siehe Kapitel 8

Grundlage für die Berechnung der (minimalen) Teil-Datenbank ist der Chase, eine Familie von Algorithmen zur Transformation von Datenbanken oder Anfragen. Seine Implementierung ist nicht explizit in ProSA enthalten, sondern als separates System implementiert. Details hierzu können im zugehörigen Git-Repository (`https://git.informatik.uni-rostock.de/ta093/chateau-demo`) oder in Kapitel 9 nachgelesen werden. Details zum konzeptuellen Ablauf von ProSA sowie zur Implementierung des ProSA-Systems können ebenfalls im zugehörigen Git-Repository (`https://git.informatik.uni-rostock.de/ta093/prosa-demo`) oder in Kapitel 10 nachgelesen werden.

11.2. Diskussion der Ergebnisse und weiterführende Fragestellungen

Wir schließen die vorliegende Arbeit mit einer kurzen Diskussion der Ergebnisse, einigen weiterführenden Ideen und offen gebliebenen Problemen.

Gütekriterium für Reproduzierbarkeit bzw. Replizierbarkeit sowie Plausibilität Gütekriterien sind Maßstäbe, mit denen die Qualität von Forschung sichergestellt werden kann. Dabei wird in der Regel zwischen quantitativen und qualitativen Gütekriterien unterschieden. In ProSA wählen wir hierfür die Reproduzierbarkeit bzw. Replizierbarkeit sowie Plausibilität eines veröffentlichten Forschungsergebnisses. Seien hierfür I^* eine rekonstruierte (minimale) Teil-Datenbank mit I^*, eine Anfrage Q sowie das zugehörige Anfrageergebnis $Q(I)$ gegeben. Gilt $Q(I^*) = Q(I)$, so ist das rekonstruierte Anfrageergebnis $Q(I^*)$ reproduzierbar bzw. replizierbar. Gilt hingegen $Q(I^*) \preccurlyeq Q(I)$, d.h. es existiert ein Homomorphismus von I^* nach I, so ist das rekonstruierte Anfrageergebnis $Q(I^*)$ plausibel. Beides kann durch Angabe der zu Q gehörenden Chase-Inversen bereits im Vorfeld überprüft werden. So garantiert das Vorliegen einer exakten Chase-Inversen die Replizierbarkeit bzw. Reproduzierbarkeit des Ergebnisses. Im Falle der anderen drei Inversentypen muss zudem $I^* \preccurlyeq I$ gelten.

Sei alternativ I^*_{anon} eine anonymisierte (minimale) Teil-Datenbank. Dann ist das Anfrageergebnis $Q'(I^*_{\text{anon}})$ plausibel, wenn für die transformierte Anfrage Q' gilt: $Q'(I^*_{\text{anon}}) \preccurlyeq Q(I)$. Da das rekonstruierte Anfrageergebnis in diesem Fall eine intensionale Beschreibung des originalen Anfrageergebnisses ist, kann die Rekonstruierbarkeit bzw. Reproduzierbarkeit des Ergebnisses hingegen nicht garantiert werden.

Erhalt der Tupelanzahl Wir gehen davon aus, dass I bereits auf die für die Auswertung der Anfrage Q relevanten Tupel eingeschränkt ist. Enthält I hingegen Tupel, welche für die Auswertung von Q absehbar nicht benötigt werden, kann $|I^*| = |I|$ niemals garantiert werden.

Daten-Evolution Wir haben uns in Kapitel 7 zunächst auf die Schema-Evolution beschränkt. Für die Integration der Daten-Evolution ist eine gesonderte Untersuchung notwendig. Wir können im Kontext des Forschungsdatenmanagements jedoch davon ausgehen, dass die zugrundeliegende Datenbank im Laufe der Zeit stetig größer wird und Datenlöschungen weitestgehend ausgeschlossen sind. Die Integration der Daten-Evolution erfolgt in ProSA bei der Invertierung der Evolution in Schritt **I.** des in Abschnitt 7.1 vorgestellten Ablaufs. Hierbei werden zunächst alle zum Zeitpunkt t nicht existenten Tupel herausselektiert und anschließend nicht weiter beachtet. Diese können bei der abschließenden Evolution der (minimalen) Teil-Datenbank in Schritt **IV.** nicht rekonstruiert werden. Das ist in unserem konkreten Anwendungsfall jedoch kein Problem.

Anwendungsfälle für ProSA Wir haben uns in der vorliegenden Arbeit insbesondere im Kontext des Forschungsdatenmanagements bewegt. Viele der in ProSA entwickelten Techniken sind jedoch auch auf andere Anwendungsgebiete anwendbar. So können wir, basierend auf der Darstellung des Hidden-Markov-Modells als Folge von SQL-Anweisungen [MH17], auch hier Aussagen zum Thema Provenance sowie zur Invertierbarkeit treffen, ohne das Modell im Detail zu kennen. Wir benötigen lediglich Informationen über die zu Grunde liegenden Operationen. So lange diese als Menge von (s-t) tgds/egds dargestellt werden können, können sie mit den in ProSA entwickelten Techniken verarbeitet werden. Da die notwendigen Operationen — Addition und Subtraktion, skalare Multiplikation und Division sowie die Matrix-Vektor- und Matrix-Matrix-Multiplikation — nach Marten als SQL-Anweisungen [MH17; MMH19] darstellbar sind, können diese anschließend mit Hilfe unseres Parsers als Menge von (s-t) tgds dargestellt werden (siehe Abschnitt 10.2) und über den Chase verarbeiten.

Auch die Untersuchung temporaler Datenbanken, das Answering Queries using Views-Problem oder andere „typische“ Chase-Anwendungen aus Tabelle 9.2 können durch die in ProSA entwickelten Techniken um zusätzliche Provenance-Informationen erweitert werden. Auch Anwendungen, welche bisher noch nicht über den Chase verarbeitet werden sind denkbar. Grundvoraussetzung ist jedoch, dass die einzuarbeitenden Informationen als Menge von (s-t) tgds dargestellt werden können.

Aggregatfunktionen und ihre Chase-Inversen In Abschnitt 6.3 haben wir die geläufigsten Auswertungsoperatoren wie die Projektion, die Selektion oder den natürlichen Verbund auf ihre Chase-Inversentypen untersucht und diese mit Hilfe zusätzlicher Provenance-Informationen „optimiert“. Auch die Aggregatfunktionen `MAX`, `MIN`, `COUNT`, `SUM` und `AVG` sowie die Gruppierung haben wir auf ihre Chase-Inversentyen untersucht (siehe Abschnitt 10.4). Anders als bei den Basisoperatoren (siehe Anhang C) stehen bei den Aggregatfunktionen die formalen Beweise jedoch noch aus.

Implementierung von ChaTEAU ChaTEAU implementiert den Chase auf einem allgemeinen Chase-Objekt $\bigcirc$ sowie einem allgemeinen Chase-Parameter $*$. Es beinhaltet eine Version der Negation, Vergleiche mit $\theta \in \{<, \leq, =, \neq, \geq, >\}$ sowie spezielle Funktionen wie die Addition und Subtraktion von Variablen und Konstanten sowie die (skalare) Multiplikation und Division (siehe Abschnitt 9.2.2). Auch die Berechnung der ***where***-, ***why***- und ***how***-Provenance wird von ChaTEAU übernommen, kann dort jedoch nicht explizit aufgerufen werden. Dennoch sind noch weitere Erweiterungen in ChaTEAU denkbar. Hierzu gehören:

- die Implementierung weiterer Konfluenz- und Terminierungskriterien (siehe Abbildung 9.3, zweiter Tab). Erste Ideen hierzu sind beispielsweise in [Gör22] zu finden;
- die Erweiterung des Funktionsparameters `FUNCTION` (siehe Abschnitt 9.2.2) zur Verarbeitung allgemeiner Funktionen;
- die Verarbeitung von second-order tgds (s-o tgd) ist in ChaTEAU bereits implizit enthalten. Wie in [Kas22] beschrieben, können s-o tgds in der Regel als Menge von mind. zwei (s-t) tgds formalisiert und dann über den Standard-Chase verarbeitet werden. Ein entsprechender Parser liegt aktuell jedoch noch nicht vor. Dieser könnte als Teil in ChaTEAU integriert oder aber als Anwendung — ähnlich zu ProSA — separat auf ChaTEAU „aufgesetzt“ werden;
- die Erweiterung der Negation. Bisher ermöglicht ChaTEAU lediglich die Negation einer einzelnen Relation (siehe Abschnitt 9.2.2). Bei der Implementierung sind wir hierbei aufgrund fehlender Homomorphismen immer wieder auf Probleme gestoßen.

Anwendungsfälle von ChaTEAU Wie in Abschnitt 9.3 beschrieben, bietet ChaTEAU als allgemeine Implementierung des Standard-Chase viele verschiedene Anwendungsmöglichkeiten. ProSA ist somit nur eine mögliche Anwendung, welche ChaTEAU als Komponente integriert.

Implementierung von ProSA Die Implementierung von ProSA ist weitestgehend abgeschlossen. Dennoch sind auch hier noch weitere Ergänzungen denkbar. So ist zur Zeit neben einer Aggregatfunktion keine weitere Operation zulässig, d.h. die Anfrage `SELECT AVG(grade) FROM Grade WHERE courseID = '2'` liefert anders als `SELECT AVG(grade) FROM Grade` kein Ergebnis. Zudem werden Duplikate, welche in der Chase-Phase entstehen können, bei der Auswertung von Aggregatfunktionen noch nicht erkannt. Dies ist jedoch kein konzeptionelles Problem, sondern eine Frage der Implementierung.

12. Danksagungen

Das Projekt *Dissertation* ist eine lange, herausfordernde und faszinierende Reise, die sowohl anspruchsvolle als auch anstrengende Phasen beinhaltet. Ohne die Hilfe so vieler Menschen ist ein solches Projekt kaum zu bewerkstelligen. Auch wenn ich noch nicht am Ziel meiner Reise angekommen bin, möchte ich mich an dieser Stelle ganz herzlich für die bisherige Unterstützung bedanken. Mein besonderer Dank gilt meinem Doktorvater Prof. em. Dr. rer. nat. habil. Andreas Heuer für die intensive Betreuung sowie die Ermöglichung dieser Dissertation. Ich danke auch Prof. Dr. Boris Glavic, Prof. Dr. rer. nat. Bertram Ludäscher sowie Dr. Götz Gräfe für die vielen interessanten Diskussionen.

Ich möchte mich auch bei meinen Kollegen vom Lehrstuhl Datenbanken und Informationssysteme (DBIS) der Universität Rostock bedanken, welche mich in den letzten Jahren täglich begleitet haben. Danke Donald, Florian, Hannes, Holger, Lukas, Meike und Sigrun. Ich hatte eine großartige Zeit am Lehrstuhl DBIS.

Während der letzten Jahre hatte ich die Ehre, viele Studierende zu betreuen, die ihre Bachelor- und Masterarbeiten bei mir verfasst oder in studentischen Projekten erste wissenschaftliche Fragestellungen bearbeitet haben. Vielen Dank an Andreas, Erik, Fabian, Jakob, Martin, Nic sowie alle Studierenden, die motiviert und ehrgeizig an ihren Abschlussarbeiten und Projekten gearbeitet und diese Dissertation mit beeinflusst haben.

Ebenfalls danken möchte ich meiner Familie und meinen Freunden für die mentale Unterstützung der letzten Jahre. Die Promotionszeit umfasst viele Hochphase, welche gefeiert, aber auch Tiefschläge, welche mit tröstenden Worten durchgestanden werden müssen. Das ihr in dieser Zeit für mich da gewesen seid, dafür danke ich insbesondere Tom, meinen Eltern Beate und Arno und Geschwistern Daniel und Julia sowie meinem engsten Freunden innerhalb und außerhalb der Universität Rostock.

Teil V.

Listen und Verzeichnisse

Abkürzungsverzeichnis

(s-t) tgd	(source-to-target) tuple generating dependency
egd	equality generating dependency
FAIR	FAIR-Prinzipien
FD	funktionale Abhängigkeit
GAV	global-as-view dependencies
GLAV	global-and-local-as-view-Abbildung
IND	Inklusionsabhängigkeit
IOW	Leibniz-Instituts für Ostseeforschung Warnemünde
JD	Verbundabhängigkeit
LAV	local-as-view dependencies
QI	Quasi-Identifikator
SMO	Schemamodifikationsoperator
s-o tgd	second-order tuple generating dependency

Symbolverzeichnis

Allgemeine mathematische Symbole

c_i	Konstanten
$\mathbb{D}(A_i)$	Domäne des Attributs A_i
$\emptyset$	leere Menge oder leere Schemaabbildung
η_i	Nullwert
$\mathbb{N}$	natürliche Zahlen
$\mathbb{N}^+$	Menge der natürlichen Zahlen ohne 0
$\mathbb{N}[X]$	Menge der Polynome mit Koeffizienten aus $\mathbb{N}$ und Variablen aus X

Symbole im *Chase*

QA_i	Benchmark-Anfragen im Bereich Provenance
QB_i	Benchmark-Anfragen im Bereich Provenance
$\bigcirc$	***Chase***-Objekt
$*$	***Chase***-Parameter

Aggregatfunktionen

`AVG`	arithmetisches Mittel einer Spalte
`COUNT`	Anzahl der Werte innerhalb einer Spalte
`MIN`/`MAX`	kleinster oder größter Wert einer Spalte
`SUM`	Summe aller Werte innerhalb einer Spalte

Provenance-Symbole

QP_i	Benchmark-Anfragen im Bereich Provenance
τ	Menge von Tupelidentifikatoren
I_w	Zeuge
w_j	Zeugenbasis (Kurzschreibweise)
$W_{Q,d}(t)$	Zeugenbasis

Abhängigkeiten und zugehörige Symbole

$\mathcal{B}$	Menge von Integritätsbedingungen
δ	engebettete Abhängigkeiten wie (s-t) tgds und egds
$dep(\Sigma)$	Abhängigkeitsgraph
$\phi(x, y)$	Body einer eingebetteten Abhängigkeit $\forall x \forall y\ \phi(x, y) \rightarrow \exists z\ \psi(x, z)$
$\psi(x, y)$	Head einer eingebetteten Abhängigkeit $\forall x \forall y\ \phi(x, y) \rightarrow \exists z\ \psi(x, z)$

Σ Abhängigkeitsmenge, d.h. Menge von (s-t) tgds und egds

Spezielle Funktionen und Abbildungen

E Evolution

E^{-1} (quasi-)inverse Evolution

h Homomorphismus

$\mathcal{M}$ Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ mit Quell-Schema S, Ziel-Schema T und Abhängigkeitsmenge Σ

$\mathcal{M}^*$ (quasi-)inverse Schemaabbildung $\mathcal{M}^* = (T, S, \Sigma^{-1})$ mit Quell-Schema T, Ziel-Schema S und Abhängigkeitsmenge Σ^{-1}

Q Auswertungsanfrage

Q^{-1} (quasi-)inverse Auswertungsanfrage

Spezielle Variablen und Variablenmengen

a_j ausgezeichnete Variablen

b_k nicht-ausgezeichnete Variablen

$\underline{\text{Const}}$ Menge von Konstanten

$\underline{\text{Null}}$ Menge von (markierten) Nullwerten

$\underline{\text{Var}}$ Menge von (nicht-)ausgezeichneten Variablen

x, y Variable

Relationenalgebra

A_i Attribute

β Umbenennung

$\bowtie$ natürlicher Verbund

$\cap$ Schnittmenge

$\cup$ Vereinigung

I, J, K Datenbankinstanzen

I^*_{anon} anonymisierte, minimale Teil-Datenbanken mit $I^* \preccurlyeq I$

I^* (minimale) Teil-Datenbanken mit $I^* \preccurlyeq I$

J^* evolutionierte Teil-Datenbanken mit $J^* \preccurlyeq J$

$\mathcal{K}$ Schlüsselmenge

K^* zu rekonstruierendes Anfrageergebnis $K^* \preccurlyeq K$

$-$ Differenz

ω skalare Operationen $\omega \in \{+, -, \cdot, :\}$

π Projektion

R_i Relationssysmbole

S_t Datenbankschema zum Zeitpunkt t

σ	Selektion
T	Ziel-Datenbankschema
t	Tupel
θ	Vergleichsoperator mit $\theta \in \{<, \leq, =, \neq, \geq, >\}$
$\times$	kartesisches Produkt
$\mathcal{U}$	Universum
X, Y	Attributmengen

Weitere Symbole

$Q(I)$	Anfrageergebnis $K = Q(I)$
H, S', S''	Side Tables
LLUNS	spezielle Variablenmenge in Llunatic
ρ	K-Relation
$\mathcal{T}$	Tableau
U	Universalplan
$\mathcal{V}$	Menge von Sichten

Tabellenverzeichnis

3.1. Relationenenalgebra-Operationen . . . 32
3.2. Komposition verschiedener Arten von Abbildungen . . . 34
3.3. Chase-Varianten und ihre Implementierungen . . . 40
3.4. Provenance-Anfragen . . . 50
3.5. Provenance-Antworten . . . 54
3.6. Liste bekannter SMOs . . . 55
3.7. Beispiel für die Unterscheidung von Privacy-Attributen (Teil I) . . . 56
3.8. Beispiel für die Unterscheidung von Privacy-Attributen (Teil II) . . . 60

4.1. Analyse-Ergebnisse der Tool-Tests . . . 64
4.2. Ergebnisse der Benchmark-Anfragen . . . 65
4.3. Klassifikation der untersuchten Provenance-Systeme nach Literatur . . . 72
4.4. Ergebnisse der Benchmark-Anfragen . . . 73
4.5. Bestimmung des unterstützten Provenance-Typs . . . 74

6.1. Hinreichende und notwendige Bedingungen für die Existenz von Chase-Inversen . . . 97
6.2. Basis-Auswertungsoperatoren und ihre Inversen . . . 97
6.3. Auswertungsanfragen mit ihren zugehörigen Chase-Inversen und Provenance-Klassen . . . 102
6.4. Auswertungsanfragen mit verschiedenen Chase-Inversen . . . 106

7.1. Beispiel zur Berechnung der (minimalen) Teil-Datenbank bei sich ändernden Datenbanken . . . 114
7.2. Schemamodificationsoperatoren . . . 115
7.3. Klassifizierte Schemaänderungen für das Sauerstoff-Beispiel am IOW . . . 115
7.4. SMOs für `MERGE Column` und `SPLIT Column` . . . 116
7.5. Schemamodifikationsoperatoren mit zugehörigen Chase-Inversen und Provenance-Klassen . . . 119
7.6. SMOs und ihre Chase-Inversen (Teil I) . . . 125
7.7. SMOs und ihre Chase-Inversen (Teil II) . . . 126

9.1. Flags für die ChaTEAU-Ausführung . . . 147
9.2. Chase-Varianten . . . 152

10.1. Vorüberlegungen zur Übersetzung von SQL-Schlüsselwörtern als Menge von (s-t) tgds . . . 161
10.2. Darstellung von SQL-Schlüsselwörtern als Menge von (s-t) tgds . . . 162
10.3. Aggregatfunktionen, dargestellt als Menge von (s-t) tgds . . . 166
10.6. Darstellung der invertierten Aggregatfunktionen als Menge von (s-t) tgds . . . 168
10.7. Chase-Inversentypen mit und ohne zusätzlichen Provenance-Informationen . . . 168
10.8. Darstellung von `SUM`(b), umgesetzt in ProSA . . . 169
10.9. Darstellung der Aggregatfunktionen erweitert um `GROUP BY` . . . 170
10.12Operationen, welche zusätzliche Provenance-Informationen benötigen . . . 172
10.13Anonymisierungszeitpunkte am konkreten Beispiel . . . 179

A.1. Relationen des Studierenden-Datenbank (Teil I) . . . xxv
A.1. Relationen des Studierenden-Datenbank (Teil II) . . . xxvi

B.1. Ergebnisse der Provenance-Anfragen QP_1 bis QP_4 . . . xxviii

E.1. Auszug eines fiktiven medizinischen Datensatzes . . . lv

Abbildungsverzeichnis

1.1. Reproduzierbarkeit, Replizierbarkeit und Plausibilität am Beispiel . . . 10
1.2. Berechnung einer (minimalen) Teil-Datenbank am konkreten Beispiel . . . 11
1.3. Datenzuwachs (pro Jahr) in IOWDB . . . 12
1.4. Auswertungsanfrage, Provenance-Anfrage und Schema-Evolution graphisch dargestellt. . . . 14
1.5. Aufbau der Dissertation . . . 17

2.1. Berechnung der (minimalen) Teil-Datenbank . . . 19
2.2. Kombination von Auswertungsanfrage, Evolution und Provenance . . . 20
2.3. Privacy im Falle von Anfrage-Invertierung . . . 21
2.4. Forschungsdaten-Lifecycle. . . . 22
2.5. Provenance im Forschungsdatenmanagement . . . 23
2.6. Berechnung einer (minimalen) Teil-Datenbank unter Berücksichtigung von zusätzlichen Provenance-Informationen und Privacy-Aspekten bei sich ändernden Datenbanken . . . 25

3.1. Relation graphisch dargestellt . . . 31
3.2. Beispiel für ein Tableau $\mathcal{T}$ . . . 38
3.3. Abhängigkeitsgraphen für verschiedene Terminierungskriterien . . . 47
3.4. Terminierungskriterien . . . 48
3.5. Provenance-Hierarchie . . . 49
3.6. Reduktion der ***where***-, ***why***- und ***how***-Provenance am konkreten Beispiel . . . 50
3.7. Beispiel für einen Provenance-Graphen . . . 54
3.8. Beispiel für Pseudnoymisierung und Anonymisierung von Daten . . . 56
3.9. Beispiel für eine Domänenegeneralisierungshierarchie . . . 60

4.1. Auswahl der untersuchten Chase-Systeme . . . 64
4.2. Chase-Ergebnisse in verschiedenen Systemen . . . 67
4.2. Chase-Ergebnisse in verschiedenen Systemen . . . 68
4.3. Auswahl der untersuchten Provenance-Systeme . . . 72
4.4. Provenance-Darstellung der Anfrage QP_3 in verschiedenen Systemen . . . 75

5.1. Rekonstruktion der (minimalen) Teil-Datenbanken I_i^* . . . 84
5.2. Chase-Anwendungen bei der Berechnung der (minimalen) Teil-Datenbanken I^* und J^* . . . 85

6.1. Chase&Backchase auf Datenbanken . . . 94
6.2. Invertierung der Aggregatfunktion `AVG` mit und ohne zusätzlicher Provenance . . . 101
6.3. Chase-Inversentyp der Projektion erweitert um Provenance-Informationen . . . 104

7.1. Kombination von Auswertungsanfrage, Evolution und Provenance . . . 112
7.2. `JOIN Table` mit Dangling Tuples . . . 121
7.3. `MERGE Column` mit verschiedenen Chase-Inversen . . . 122
7.4. `MERGE Table` erweitert um zusätzliche Provenance-Informationen . . . 123

8.1. Ergebnisse der Interview-Studie (Teil I) . . . 133
8.2. Ergebnisse der Interview-Studie (Teil II) . . . 134
8.3. Ergebnisse der Interview-Studie (Teil III) . . . 134
8.4. Mögliche Provenance-basierte Datenbankrekonstruktion einschließlich Verallgemeinerung . . . 137
8.5. Anonymisiserungsmethoden . . . 138
8.6. Konzepthierarchien für die drei Attributtypen 1 bis 3 . . . 140

9.1. Hierarchien von Chase-Objekt und Chase-Parameter . . . 143
9.2. Konzeptueller Ablauf von ChaTEAU . . . 146
9.3. Überblick über die ChaTEAU-GUI . . . 148
9.4. ChaTEAU-Warnmeldung . . . 149

10.1. Einbindung von ChaTEAU in ProSA . . . 155
10.2. Konzeptueller Ablauf in ProSA . . . 156
10.3. Beispielbaum für die Verarbeitung im Parser . . . 163
10.4. Auswertung einer Aggregatfunktion in ProSA . . . 171
10.5. Schematische Darstellung der Anonymisierung durch Generalisierung und Intensionalisierung . . . 176

10.6. Mögliche Anonymisierungszeitpunkte innerhalb der ProSA-Pipeline 177
10.7. Chase-Anwendung bei der Berechnung der (minimalen) Teil-Datenbanken I^* und J^* 180
10.8. Konfiguration und Logging in der ProSA-GUI 182
10.9. Evolution in der ProSA-GUI 183
10.10Chase&Backchase in der ProSA-GUI 183
10.11Anonymisierung in der ProSA-GUI 184

Anfrageverzeichnis

4.1. Verbundtreue-Kriterium mit einer FD (QA_2) . . . 65
4.2. Multi-Attribut-Fremdschlüssel mit interner FD (QB_4) . . . 65
4.3. Verbund dreier Relationen (QP_3) . . . 73
4.4. Aggregation und Gruppierung (QP_4), umgeschrieben für Verarbeitung in Trio . . . 74

9.1. Beispiel für Negation in ChaTEAU . . . 150
9.2. Beispiel für Vergleiche in ChaTEAU . . . 151
9.3. Beispiel für Funktionen in ChaTEAU . . . 151

B.1. Test auf Duplikateliminierung (QP_1) . . . xxvii
B.2. Selektion und Verbund (QP_2) . . . xxvii
B.3. Verbund dreier Relationen (QP_3) . . . xxvii
B.4. Aggregation und Gruppierung (QP_4) . . . xxvii
B.5. Optimierung von Schlüssel-Fremdschlüssel-Verbunden (QA_1) . . . xxix
B.6. Verbundtreue-Kriterium mit einer FD (QA_2) . . . xxix
B.7. Verbundtreue-Kriterium mit einer JD (QA_3) . . . xxix

D.1. XML-Datei für die Eingabe in ChaTEAU (am Studierenden-Beispiel) . . . xlvi
D.2. XML-Datei als Ausgabe aus ChaTEAU (am Studierenden-Beispiel) . . . xlvii
D.3. Eingabe-Datei für ChaTEAU (Anfrage QA1) . . . xlviii
D.4. Eingabe-Datei für ChaTEAU (Anfrage QA2) . . . xlix
D.5. Eingabe-Datei für ChaTEAU (Anfrage QA3) . . . l
D.6. Evolution als Eingabe für ProSA (am Studierenden-Beispiel) . . . li
D.7. provenance-erweiterter Parser-Output (am Studierenden-Beispiel) . . . lii
D.8. Domänengeneralisierungshierarchie als Eingabe für ProSA (am Studierenden-Beispiel) . . . lii
D.9. provenance-erweiterter Chase-Output (am Studierenden-Beispiel) . . . liii
D.10.(minimale) Teil-Datenbank als Ergebnis von ProSA (am Studierenden-Beispiel) . . . liv

Projektverzeichnis

Benchmarking the Chase

Projektwebsite	https://dbunibas.github.io/chasebench/
GitHub	https://github.com/dbunibas/chasebench

Chase-Systeme

ChaseFUN	https://github.com/dbunibas/chasebench/tree/master/tools/chasefun
ChaseTEQ	http://wwwinfo.deis.unical.it/chaseteq/index.htm
Graal	https://graphik-team.github.io/graal/
Llunatic	https://www.db.unibas.it/projects/llunatic/
PDQ	http://www.cs.ox.ac.uk/projects/pdq/home.html
Pegasus	https://sourceforge.net/projects/chase-backchase/
$\text{Prov}_{\text{C\&B}}$	https://github.com/IoanaIleana/ProvCB

Provenance-Systeme

GProM	https://github.com/IITDBGroup/gprom
Orchestra	https://www.cis.upenn.edu/~zives/orchestra/
Perm	https://github.com/IITDBGroup/perm
$\text{Prov}_{\text{C\&B}}$	https://github.com/IoanaIleana/ProvCB
ProvSQL	https://github.com/PierreSenellart/provsql/
Tioga/Datasplash	http://datasplash.cs.berkeley.edu
Trio	http://infolab.stanford.edu/trio/

Während der Dissertation entwickelte Projekte

ChaTEAU	https://git.informatik.uni-rostock.de/ta093/chateau-demo
ProSA	https://git.informatik.uni-rostock.de/ta093/prosa-demo

Literatur

[ABJ06] Ilkay Altintas, Oscar Barney und Efrat Jaeger-Frank. "Provenance Collection Support in the Kepler Scientific Workflow System". In: *IPAW*. Bd. 4145. LNCS. Springer, 2006, S. 118–132.

[ABS15] Antoine Amarilli, Pierre Bourhis und Pierre Senellart. *Provenance Circuits for Trees and Treelike Instances (Extended Version)*. http://arxiv.org/abs/1511.08723, 2015.

[ABU79] Alfred V. Aho, Catriel Beeri und Jeffrey D. Ullman. "The Theory of Joins in Relational Databases". In: *ACM Trans. Database Syst.* 4.3 (1979), S. 297–314.

[Aca+10] Umut A. Acar, Peter Buneman, James Cheney, Jan Van den Bussche, Natalia Kwasnikowska und Stijn Vansummeren. "A Graph Model of Data and Workflow Provenance". In: *TaPP*. USENIX Association, 2010.

[ADT11] Yael Amsterdamer, Daniel Deutch und Val Tannen. "Provenance for Aggregate Queries". In: *PODS*. ACM, 2011, S. 153–164.

[Agr+06] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara und Jennifer Widom. "Trio — A System for Data, Uncertainty, and Lineage". In: *VLDB*. ACM, 2006, S. 1151–1154.

[Agr+09] Parag Agrawal, Robert Ikeda, Hyunjung Park und Jennifer Widom. *Trio-ER — The Trio System as a Workbench for Entity-Resolution*. Technical Report, Stanford InfoLab, http://ilpubs.stanford.edu:8090/912/, 2009.

[AH18a] Tanja Auge und Andreas Heuer. "Combining Provenance Management and Schema Evolution". In: *IPAW*. Bd. 11017. LNCS. Springer, 2018, S. 222–225.

[AH18b] Tanja Auge und Andreas Heuer. "Inverse im Forschungsdatenmanagement". In: *30th Workshop Grundlagen von Datenbanken* (2018), S. 108–113.

[AH18c] Tanja Auge und Andreas Heuer. "The Theory behind Minimizing Research Data — Result equivalent Chase-inverse Mappings". In: *LWDA*. Bd. 2191. CEUR Workshop Proceedings. CEUR-WS.org, 2018, S. 1–12.

[AH19] Tanja Auge und Andreas Heuer. "ProSA — Using the Chase for Provenance Management". In: *ADBIS*. Bd. 11695. LNCS. Springer, 2019, S. 357–372.

[AH21] Tanja Auge und Andreas Heuer. "Tracing the History of the Baltic Sea Oxygen Level". In: *BTW*. Bd. P-311. LNI. Gesellschaft für Informatik, Bonn, 2021, S. 337–348.

[AH22a] Tanja Auge und Andreas Heuer. *Enhanced Inversion of Schema Evolution with Provenance*. Veröffentlichung auf https://arxiv.org am 28.11. 2022.

[AH22b] Tanja Auge und Andreas Heuer. *Testing Provenance Systems*. Techn. Ber. CS 01-22. University of Rostock, DBIS, 2022. URL: https://eprints.dbis.informatik.uni-rostock.de/1076/.

[AHH22] Tanja Auge, Moritz Hanzig und Andreas Heuer. "ProSA Pipeline — Provenance Conquers the Chase". In: *ADBIS*. Bd. 1652. CCIS. Springer, 2022, S. 89–98.

[Aik+96] Alexander Aiken, Jolly Chen, Michael Stonebraker und Allison Woodruff. "Tioga-2 — A Direct Manipulation Database Visualization Environment". In: *ICDE*. IEEE Computer Society, 1996, S. 208–217.

[AL12] Khalid Nawaf Alharbi und Xiaodong Lin. "PDP — A Privacy-Preserving Data Provenance Scheme". In: *ICDCS Workshops*. IEEE Computer Society, 2012, S. 500–505.

[Alb+21] Michael Albus, Eduard Buch, Lukas Görtz, Moritz Hanzig und Eric Maier. *Qualitätssicherung für ChaTEAU*. Abschlussbericht KSWS, Universität Rostock, DBIS. 2021.

[Ams+11] Yael Amsterdamer, Susan B. Davidson, Daniel Deutch, Tova Milo, Julia Stoyanovich und Val Tannen. "Putting Lipstick on Pig — Enabling Database-style Workflow Provenance". In: *Proc. VLDB Endow.* 5.4 (2011), S. 346–357.

[Ara+18] Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu und Qitian Zeng. "GProM — A Swiss Army Knife for Your Provenance Needs". In: *IEEE Data Eng. Bull.* 41.1 (2018), S. 51–62.

[ASH21a] Tanja Auge, Nic Scharlau und Andreas Heuer. "Privacy Aspects of Provenance Queries". In: *IPAW*. Bd. 12839. LNCS. Springer, 2021, S. 218–221.

[ASH21b] Tanja Auge, Nic Scharlau und Andreas Heuer. “Provenance and Privacy in ProSA — A Guided Interview on Privacy-Aware Provenance”. In: *DEXA Workshops*. Bd. 1479. CCIS. Springer, 2021, S. 52–62.

[Aug+20] Tanja Auge, Erik Manthey, Susanne Jürgensmann, Susanne Feistel und Andreas Heuer. “Schema Evolution and Reproducibility of Long-term Hydrographic Data Sets at the IOW”. In: *LWDA*. Bd. 2738. CEUR Workshop Proceedings. CEUR-WS.org, 2020, S. 258–269.

[Aug+22] Tanja Auge, Nic Scharlau, Andreas Görres, Jakob Zimmer und Andreas Heuer. *ChaTEAU — A Universal Toolkit for Applying the* Chase. https://arxiv.org/abs/2206.01643, 2022.

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

[Aug18] Tanja Auge. *Exposé eines Promotionsprojektes — Provenance Management für Data-Science-Anwendungen unter Berücksichtigung von Daten- und Schema-Evolution*. Exposé, Universität Rostock, DBIS. 2018.

[Aug20] Tanja Auge. “Extended Provenance Management for Data Science Applications”. In: *PhD@VLDB*. Bd. 2652. CEUR Workshop Proceedings. CEUR-WS.org, 2020.

[Bag+15] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher und Clément Sipieter. “Graal — A Toolkit for Query Answering with Existential Rules”. In: *RuleML*. Bd. 9202. LNCS. Springer, 2015, S. 328–344.

[BCC06] Peter Buneman, Adriane Chapman und James Cheney. “Provenance management in curated databases”. In: *SIGMOD Conference*. ACM, 2006, S. 539–550.

[Ben+06] Omar Benjelloun, Anish Das Sarma, Chris Hayworth und Jennifer Widom. *An Introduction to ULDBs and the Trio System*. Technical Report, Stanford InfoLab, http://ilpubs.stanford.edu:8090/793/, 2006.

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

[Ber+14] Elisa Bertino, Gabriel Ghinita, Murat Kantarcioglu, Dang Nguyen, Jae Park, Ravi S. Sandhu, Salmin Sultana, Bhavani Thuraisingham und Shouhuai Xu. “A roadmap for privacy-enhanced secure data provenance”. In: *J. Intell. Inf. Syst.* 43.3 (2014), S. 481–501.

[BFJ14] Steffen Bock, Susanne Feistel und Susanne Jürgensmann. *Data Management at IOW*. Poster. 2014.

[Bha+05] Deepavali Bhagwat, Laura Chiticariu, Wang Chiew Tan und Gaurav Vijayvargiya. “An annotation management system for relational databases”. In: *VLDB J.* 14.4 (2005), S. 373–396.

[BIL16] Angela Bonifati, Ioana Ileana und Michele Linardi. “Functional Dependencies Unleashed for Scalable Data Exchange”. In: *SSDBM*. ACM, 2016, 2:1–2:12.

[BIL17] Angela Bonifati, Ioana Ileana und Michele Linardi. “ChaseFUN — A Data Exchange Engine for Functional Dependencies at Scale”. In: *EDBT*. OpenProceedings.org, 2017, S. 534–537.

[BKT01] Peter Buneman, Sanjeev Khanna und Wang Chiew Tan. “Why and Where — A Characterization of Data Provenance”. In: *ICDT*. Bd. 1973. Springer, 2001, S. 316–330.

[BLT14] Michael Benedikt, Julien Leblay und Efthymia Tsamoura. “PDQ — Proof-driven Query Answering over Web-based Data”. In: *Proc. VLDB Endow.* 7.13 (2014), S. 1553–1556.

[BLT15] Michael Benedikt, Julien Leblay und Efthymia Tsamoura. “Querying with Access Patterns and Integrity Constraints”. In: *Proc. VLDB Endow.* 8.6 (2015), S. 690–701.

[BÓ81] Ronald V. Book und Colm Ó’Dúnlaing. “Testing for the Church-Rosser Property”. In: *Theor. Comput. Sci.* 16 (1981), S. 223–229.

[Bra+06] Uri Braun, Simson L. Garfinkel, David A. Holland, Kiran-Kumar Muniswamy-Reddy und Margo I. Seltzer. “Issues in Automatic Provenance Collection”. In: *IPAW*. Bd. 4145. LNCS. Springer, 2006, S. 171–183.

[Bre+19] Willi Brekenfelder, Ruven Kronenberg, Sebastian Rutofski, Sergej Schimanski und Jakob Zimmer. Chase-*Tools (Teil II)*. Abschlussbericht Projekt, Universität Rostock, DBIS. 2019.

[Bre+20] Willi Brekenfelder, Ruven Kronenberg, Maximilan Lamster, Erik Manthey und Jakob Zimmer. Chase-*Tools (Teil III)*. Abschlussbericht NEidI, Universität Rostock, DBIS. 2020.

[Bru+17] Ilvio Bruder, Meike Klettke, Mark Lukas Möller, Frank Meyer, Andreas Heuer, Susanne Jürgensmann und Susanne Feistel. “Daten wie Sand am Meer — Datenerhebung, -strukturierung, -management und Data Provenance für die Ostseeforschung”. In: *Datenbank-Spektrum* 17.2 (2017), S. 183–196.

[BT07] Peter Buneman und Wang Chiew Tan. “Provenance in databases”. In: *SIGMOD Conference*. ACM, 2007, S. 1171–1173.

[BV84] Catriel Beeri und Moshe Y. Vardi. “A Proof Procedure for Data Dependencies”. In: *J. ACM* 31.4 (1984), S. 718–741.

[Cat+09] Balder ten Cate, Laura Chiticariu, Phokion G. Kolaitis und Wang Chiew Tan. “Laconic Schema Mappings — Computing the Core with SQL Queries”. In: *Proc. VLDB Endow.* 2.1 (2009), S. 1006–1017.

[CCT09] James Cheney, Laura Chiticariu und Wang Chiew Tan. “Provenance in Databases — Why, How, and Where”. In: *Foundations and Trends in Databases* 1.4 (2009), S. 379–474.

[CGK08] Andrea Cali, Georg Gottlob und Michael Kifer. “Taming the Infinite Chase — Query Answering under Expressive Relational Constraints”. In: *Description Logics*. Bd. 353. CEUR Workshop Proceedings. CEUR-WS.org, 2008.

[Che11] James Cheney. “A Formal Framework for Provenance Security”. In: *CSF*. IEEE Computer Society, 2011, S. 281–293.

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

[CMZ08] Carlo A. Curino, Hyun J. Moon und Carlo Zaniolo. “Graceful Database Schema Evolution — The PRISM Workbench”. In: *PVLDB* 1.1 (2008), S. 761–772.

[Cod70] E. F. Codd. “A Relational Model of Data for Large Shared Data Banks”. In: *Commun. ACM* 13.6 (1970), S. 377–387.

[Cor11] Graham Cormode. “Personal privacy vs population privacy — Learning to attack anonymization”. In: *KDD*. ACM, 2011, S. 1253–1261.

[CR36] Alonzo Church und J. Barkley Rosser. “Some properties of conversion”. In: *Transactions of the American Mathematical Society* 39.3 (1936), S. 472–482.

[CTV05] Laura Chiticariu, Wang Chiew Tan und Gaurav Vijayvargiya. “DBNotes — A post-it system for relational databases based on provenance”. In: *SIGMOD Conference*. ACM, 2005, S. 942–944.

[Cur+08] Carlo A. Curino, Letizia Tanca, Hyun J. Moon und Carlo Zaniolo. “Schema Evolution in Wikipedia — Toward a Web Information System Benchmark”. In: *ICEIS (1)*. 2008, S. 323–332.

[Cur+10] Carlo Curino, Hyun Jin Moon, Alin Deutsch und Carlo Zaniolo. “Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System — PRISM++”. In: *Proc. VLDB Endow.* 4.2 (2010), S. 117–128.

[CW00] Yingwei Cui und Jennifer Widom. “Practical Lineage Tracing in Data Warehouses”. In: *ICDE*. IEEE Computer Society, 2000, S. 367–378.

[CWW00] Yingwei Cui, Jennifer Widom und Janet L. Wiener. “Tracing the lineage of view data in a warehousing environment”. In: *ACM Trans. Database Syst.* 25.2 (2000), S. 179–227.

[CY20] Özgü Can und Dilek Yilmazer. “A novel approach to provenance management for privacy preservation”. In: *J. Inf. Sci.* 46.2 (2020).

[Dav+11] Susan B. Davidson, Sanjeev Khanna, Sudeepa Roy, Julia Stoyanovich, Val Tannen und Yi Chen. “On provenance and privacy”. In: *ICDT*. ACM, 2011, S. 3–10.

[DF08] Susan B. Davidson und Juliana Freire. “Provenance and scientific workflows — Challenges and opportunities”. In: *SIGMOD Conference*. ACM, 2008, S. 1345–1350.

[DH13] Alin Deutsch und Richard Hull. “Provenance-Directed Chase&Backchase”. In: *In Search of Elegance in the Theory and Practice of Computation*. Bd. 8000. LNCS. Springer, 2013, S. 227–236.

[DM02] Josep Domingo-Ferrer und Josep Maria Mateo-Sanz. “Practical Data-Oriented Microaggregation for Statistical Disclosure Control”. In: *IEEE Trans. Knowl. Data Eng.* 14.1 (2002), S. 189–201.

[DNR08] Alin Deutsch, Alan Nash und Jeffrey B. Remmel. “The Chase revisited”. In: *PODS*. ACM, 2008, S. 149–158.

[DPT06] Alin Deutsch, Lucian Popa und Val Tannen. “Query Reformulation with Constraints”. In: *SIGMOD Record* 35.1 (2006), S. 65–73.

[DPT99] Alin Deutsch, Lucian Popa und Val Tannen. “Physical Data Independence, Constraints, and Optimization with Universal Plans”. In: *VLDB*. Morgan Kaufmann, 1999, S. 459–470.

[DT01] Alin Deutsch und Val Tannen. “Optimization Properties for Classes of Conjunctive Regular Path Queries”. In: *DBPL*. Bd. 2397. LNCS. Springer, 2001, S. 21–39.

[Dwo06] Cynthia Dwork. “Differential Privacy”. In: *ICALP (2)*. Bd. 4052. LNCS. Springer, 2006, S. 1–12.

[Dwo08] Cynthia Dwork. “Differential Privacy — A Survey of Results”. In: *TAMC*. Bd. 4978. LNCS. Springer, 2008, S. 1–19.

[ED08] Khaled El Emam und Fida Kamal Dankar. "Research Paper — Protecting Privacy Using k-Anonymity". In: *J. Am. Medical Informatics Assoc.* 15.5 (2008), S. 627–637.

[EFT07] Heinz-Dieter Ebbinghaus, Jörg Flum und Wolfgang Thomas. *Einführung in die mathematische Logik, 5. Auflage*. Spektrum Akademischer Verlag, 2007.

[Fag+05a] Ronad Fagin, Phokion G. Kolaitis, Renée J. Miller und Lucian Popa. "Data Exchange — Semantics and Query Answering". In: *Theor. Comput. Sci.* 336.1 (2005), S. 89–124.

[Fag+05b] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa und Wang Chiew Tan. "Composing schema mappings — Second-order dependencies to the rescue". In: *ACM Trans. Database Syst.* 30.4 (2005), S. 994–1055.

[Fag+08] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa und Wang Chiew Tan. "Quasi-Inverses of Schema Mappings". In: *ACM Transaction of Database Systems* 33.2 (2008), 11:1–11:52. DOI: 10.1145/1366102.1366108.

[Fag+09] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa und Wang Chiew Tan. "Reverse data exchange — Coping with nulls". In: *PODS*. ACM, 2009, S. 23–32.

[Fag+11a] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa und Wang Chiew Tan. "Reverse data exchange — Coping with nulls". In: *ACM Trans. Database Syst.* 36.2 (2011), 11:1–11:42.

[Fag+11b] Ronald Fagin, 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. Springer, 2011, S. 191–222.

[Fag07] Ronald Fagin. "Inverting Schema Mappings". In: *ACM Transaction Database Systems* 32.4 (2007).

[Fla+20a] Rocco Flach, Leon Herrmann, Max Kaseler, Chris Röhrs und Artur Strelnikov. *Provenance Tools (Teil I)*. Abschlussbericht KSWS, Universität Rostock, DBIS. 2020.

[Fla+20b] Rocco Flach, Maximilian Lamster, Chris Röhrs, Nic Scharlau und Tanja Auge. *Provenance Tools (Teil II)*. Abschlussbereicht, Universtität Rostock, DBIS. 2020.

[För+21] Leonie Förster, Melinda Heuser, Judith-Henrike Overath, Anne-Sophie Waterstradt und Anja Wolpers. *Data Provenance*. Abschlussbericht KSWS, Universität Rostock, DBIS. 2021.

[Fre+08] Juliana Freire, David Koop, Emanuele Santos und Cláudio T. Silva. "Provenance for Computational Tasks — A Survey". In: *Comput. Sci. Eng.* 10.3 (2008), S. 11–21.

[Fro22] Jördis Frommhold. *LongCovid — Die neue Volkskrankheit*. C.H.Beck, 2022. ISBN: 9783406783579.

[FRS20] Rocco Flach, Chris Röhrs und Nic Scharlau. *Data Provenance-Tools (Teil II)*. Abschlussbericht Projekt und NEidI, Universität Rostock, DBIS. 2020.

[FST11] Andrea De Francesco, Francesca Spezzano und Irina Trubitsyna. "ChaseT — A Tool For Checking Chase Termination". In: *SEBD*. 2011, S. 163–174.

[GA09] Boris Glavic und Gustavo Alonso. "Perm — Processing Provenance and Data on the Same Data Model through Query Rewriting". In: *ICDE*. IEEE Computer Society, 2009, S. 174–185.

[Gee+13] Floris Geerts, Giansalvatore Mecca, Paolo Papotti und Donatello Santoro. "The LLUNATIC Data-Cleaning Framework". In: *PVLDB* 6.9 (2013), S. 625–636.

[Gee+14a] Floris Geerts, Giansalvatore Mecca, Paolo Papotti und Donatello Santoro. *Mapping and Cleaning — The LLUNATIC Way*. Technical Report, University of Basilicata, https://www.db.unibas.it/projects/llunatic/, 2014.

[Gee+14b] Floris Geerts, Giansalvatore Mecca, Paolo Papotti und Donatello Santoro. "That's All Folks! — LLUNATIC Goes Open Source". In: *PLVDB* 7.13 (2014), S. 1565–1568.

[Gee+20] Floris Geerts, Giansalvatore Mecca, Paolo Papotti und Donatello Santoro. "Cleaning data with Llunatic". In: *VLDB J.* 29.4 (2020), S. 867–892.

[GH15] Hannes Grunert und Andreas Heuer. "Slicing in Assistenzsystemen — Wie trotz Anonymisierung von Daten wertvolle Analyseergebnisse gewonnen werden können". In: *GvD*. Bd. 1366. CEUR Workshop Proceedings. CEUR-WS.org, 2015, S. 24–29.

[GH16a] Hannes Grunert und Andreas Heuer. "Datenschutz im PArADISE". In: *Datenbank-Spektrum* 16.2 (2016), S. 107–117.

[GH16b] Hannes Grunert und Andreas Heuer. "Privacy Protection through Query Rewriting in Smart Environments". In: *EDBT*. OpenProceedings.org, 2016, S. 708–709.

[GH17] Hannes Grunert und Andreas Heuer. "Rewriting Complex Queries from Cloud to Fog under Capability Constraints to Protect the Users' Privacy". In: *Open J. Internet Things* 3.1 (2017), S. 31–45.

[GH18] Hannes Grunert und Andreas Heuer. “Query Rewriting by Contract under Privacy Constraints”. In: *Open J. Internet Things* 4.1 (2018), S. 54–69.

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

[GLS14] Aris Gkoulalas-Divanis, Grigorios Loukides und Jimeng Sun. “Publishing data from electronic health records while preserving privacy — A survey of algorithms”. In: *J. Biomed. Informatics* 50 (2014), S. 4–19.

[GM15] Sergio Greco und Cristian Molinaro. *Datalog and Logic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

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

[GN08] Georg Gottlob und Alan Nash. “Efficient core computation in data exchange”. In: *J. ACM* 55.2 (2008), 9:1–9:49.

[Gör20] Andreas Oliver Görres. *Erweiterung des* Chase-*Werkzeugs ChaTEAU um ein Terminierungskriterium*. Masterarbeit, Universität Rostock, DBIS. 2020.

[Gör22] Andreas Görres. “Termination and Confluence of an Extended CHASE Algorithm”. In: *ADBIS*. Bd. 1652. Communications in Computer and Information Science. Springer, 2022, S. 631–638.

[Gra80] Marc H. Graham. “A New Proof that the Chase is a Church-Rosser Replacement System”. In: *XP1 Workshop on Database Theory*. 1980.

[Gre+07] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives und Val Tannen. “Update Exchange with Mappings and Provenance”. In: *VLDB*. ACM, 2007, S. 675–686.

[Gru22] Hannes Grunert. “Vertrauenswürdige, adaptive Anfrageverarbeitung in dynamischen Sensornetzwerken zur Unterstützung assistiver Systeme”. Diss. Universität Rostock, IEF, DBIS, 2022.

[GST11] Sergio Greco, Francesca Spezzano und Irina Trubitsyna. “Stratification Criteria and Rewriting Techniques for Checking Chase Termination”. In: *Proc. VLDB Endow.* 4.11 (2011), S. 1158–1168.

[GT17] Todd J. Green und Val Tannen. “The Semiring Framework for Database Provenance”. In: *PODS*. ACM, 2017, S. 93–99.

[GZ12] Shi Gao und Carlo Zaniolo. “Provenance Management in Databases Under Schema Evolution”. In: *TaPP*. USENIX Association, 2012.

[Haa99] Laura M. Haas. “Review — Physical Data Independence, Constraints, and Optimization with Universal Plans”. In: *ACM SIGMOD Digit. Rev.* 1 (1999).

[Hal01] Alon Y. Halevy. “Answering Queries Using Views — A Survey”. In: *VLDB J.* 10.4 (2001), S. 270–294. DOI: 10.1007/s007780100054.

[Han22] Moritz Hanzig. *Ein Framework für ProSA*. Bachelorarbeit, Universität Rostock, DBIS. 2022.

[HDL17] Melanie Herschel, Ralf Diestelkämper und Houssem Ben Lahmar. “A survey on provenance — What for? What form? What from?” In: *VLDB J.* 26.6 (2017), S. 881–906.

[Heb06] Udo Hebisch. *Ontologien*. abgerufen am 03.03.2022. 2006.

[Her15] Melanie Herschel. “A Hybrid Approach to Answering Why-Not Questions on Relational Query Results”. In: *ACM J. Data Inf. Qual.* 5.3 (2015), 10:1–10:29.

[Heu20] Andreas Heuer. *Theorie relationaler Datenbanken*. Vorlesungsskript, Universität Rostock, DBIS. 2020.

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

[Hul97] Richard Hull. “Managing Semantic Heterogeneity in Databases — A Theoretical Perspective”. In: *PODS*. ACM Press, 1997, S. 51–61.

[Ile+14] Ioana Ileana, Bogdan Cautis, Alin Deutsch und Yannis Katsis. “Complete yet practical search for minimal query reformulations under constraints”. In: *SIGMOD*. ACM, 2014, S. 1015–1026.

[Ile14] Ioana Ileana. “Query rewriting using views — A theoretical and practical perspective”. Diss. ELECOM ParisTech Spécialité, Informatique et Réseaux, 2014.

[Ins20] Johner Insitut. *Anonymisierung und Pseudonymisierung*. https://www.johner-institut.de/blog/medizinische-informatik/anonymisierung-und-pseudonymisierung/. 2020.

[Ive+05] Zachary G. Ives, Nitin Khandelwal, Aneesh Kapur und Murat Cakir. “ORCHESTRA — Rapid, Collaborative Sharing of Dynamic Data”. In: *CIDR*. www.cidrdb.org, 2005, S. 107–118.

[Ive+08] Zachary G. Ives, Todd J. Green, Grigoris Karvounarakis, Nicholas E. Taylor, Val Tannen, Partha Pratim Talukdar, Marie Jacob und Fernando C. N. Pereira. "The ORCHESTRA Collaborative Data Sharing System". In: *SIGMOD Rec.* 37.3 (2008), S. 26–32.

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

[Kas22] Max Tilman Kaseler. *So-tgds und Skolemisierung für ChaTEAU*. Masterarbeit, Universität Rostock, DBIS. 2022.

[Kav22] Ivo Kavisanczki. *Erweiterung des ProSA-Parsers*. Bachelorarbeit, Universität Rostock, DBIS. 2022.

[KLZ13] Sven Köhler, Bertram Ludäscher und Daniel Zinn. *First-Order Provenance Games*. https://arxiv.org/abs/1309.2655. 2013.

[Lam21] Maximilian Lamster. *Provenance-unterstützte Datenanalyse in Kombination mit intensionalen Antworten zur Steigerung der Privatsphäre*. Masterarbeit, Universität Rostock, DBIS. 2021.

[Lee+17] Seokki Lee, Sven Köhler, Bertram Ludäscher und Boris Glavic. "A SQL-Middleware Unifying Why and Why-Not Provenance for First-Order Queries". In: *ICDE*. IEEE Computer Society, 2017, S. 485–496.

[Leo+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri und Francesco Scarcello. "The DLV system for knowledge representation and reasoning". In: *ACM Trans. Comput. Log.* 7.3 (2006), S. 499–562.

[Li+12] Tiancheng Li, Ninghui Li, Jian Zhang und Ian M. Molloy. "Slicing — A New Approach for Privacy Preserving Data Publishing". In: *IEEE Trans. Knowl. Data Eng.* 24.3 (2012), S. 561–574.

[LLV07] Ninghui Li, Tiancheng Li und Suresh Venkatasubramanian. "t-Closeness — Privacy Beyond k-Anonymity and l-Diversity". In: *ICDE*. IEEE Computer Society, 2007, S. 106–115.

[Mac+06] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer und Muthuramakrishnan Venkitasubramaniam. "l-Diversity — Privacy Beyond k-Anonymity". In: *ICDE*. IEEE Computer Society, 2006, S. 24.

[Mai83] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[Man20] Erik Manthey. *Beschreibung der Veränderungen von Schemata und Daten am IOW mit Schema-Evolutions-Operatoren*. Bachelorarbeit, Universität Rostock, DBIS. 2020.

[Mar09] Bruno Marnette. "Generalized schema-mappings — From termination to tractability". In: *PODS*. ACM, 2009, S. 13–22.

[Mei14] Michael Meier. "The BACKCHASE revisited". In: *VLDB J.* 23.3 (2014), S. 495–516.

[MG12] Abhijeet Mohapatra und Michael Genesereth. *Aggregation in Datalog Under Set Semantics*. Technial Report, Stanford University, https://www.yumpu.com/en/document/view/24665763/aggregation-in-datalog-under-set-semantics-stanford-logic-group-. 2012.

[MH17] Dennis Marten und Andreas Heuer. "Machine Learning on Large Databases — Transforming Hidden Markov Models to SQL Statements". In: *Open J. Databases* 4.1 (2017), S. 22–42.

[MMH19] Dennis Marten, Holger Meyer und Andreas Heuer. "Calculating Fourier Transforms in SQL". In: *ADBIS*. Bd. 11695. LNCS. Springer, 2019, S. 151–166.

[MMS79] David Maier, Alberto O. Mendelzon und Yehoshua Sagiv. "Testing Implications of Data Dependencies". In: *ACM Trans. Database Syst.* 4.4 (1979), S. 455–469.

[Moo+08] Hyun Jin Moon, Carlo Curino, Alin Deutsch, Chien-Yi Hou und Carlo Zaniolo. "Managing and Querying Transaction-time Databases under Schema Evolution". In: *PVLDB* 1.1 (2008), S. 882–895.

[Mot94] Amihai Motro. "Intensional Answers to Database Queries". In: *IEEE Trans. Knowl. Data Eng.* 6.3 (1994), S. 444–454.

[MSL09a] Michael Meier, Michael Schmidt und Georg Lausen. *On Chase Termination Beyond Stratification*. https://arxiv.org/abs/0906.4228. 2009.

[MSL09b] Michael Meier, Michael Schmidt und Georg Lausen. "On Chase Termination Beyond Stratification". In: *PVLDB* 2.1 (2009), S. 970–981.

[MW04] Adam Meyerson und Ryan Williams. "On the Complexity of Optimal K-Anonymity". In: *PODS*. ACM, 2004, S. 223–228.

[NAC07] Mehmet Ercan Nergiz, Maurizio Atzori und Chris Clifton. "Hiding the presence of individuals from shared databases". In: *SIGMOD Conference*. ACM, 2007, S. 665–676.

[NC10] Mehmet Ercan Nergiz und Christopher W. Clifton. "d-Presence without Complete World Knowledge". In: *IEEE Trans. Knowl. Data Eng.* 22.6 (2010), S. 868–883.

[One12] Adrian Constantin Onet. "The Chase procedure and its applications". Diss. Concordia University, Department of Computer Science und Software Engineering, 2012.

[PRS17] Beatriz Pérez, Julio Rubio und Carlos Sáenz-Adán. *Provenance Systems Roadmap*. https://www.unirioja.es/cu/beperev/ProvenanceSystemsRoadmap.pdf, abgerufen am 30.12.2021. 2017.

[PRS18] Beatriz Pérez, Julio Rubio und Carlos Sáenz-Adán. "A systematic review of provenance systems". In: *Knowl. Inf. Syst.* 57.3 (2018), S. 495–543.

[PS09] Reinhard Pichler und Vadim Savenkov. "DEMo — Data Exchange Modeling Tool". In: *Proc. VLDB Endow.* 2.2 (2009), S. 1606–1609.

[PS17] Ronald Petrlic und Christoph Sorge. *Datenschutz — Einführung in technischen Datenschutz, Datenschutzrecht und angewandte Kryptographie*. Springer-Verlag, 2017.

[PY99] E. K. Park und Suk-Chung Yoon. "An Approach to Intensional Query Answering at Multiple Abstraction Levels using Data Mining Approaches". In: *HICSS*. IEEE Computer Society, 1999.

[QLS13] Dong Qiu, Bixin Li und Zhendong Su. "An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications". In: *ESEC/SIGSOFT FSE*. ACM, 2013, S. 125–135.

[Rau+21] Andreas Rauber u. a. "Precisely and Persistently Identifying and Citing Arbitrary Subsets of Dynamic Data". In: *Harvard Data Science Review* 3.4 (2021).

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

[Ros20] Florian Rose. *Erweiterung des* Chase-*Werkzeugs ChaTEAU um eine* Backchase-*Phase*. Masterarbeit, Universität Rostock, DBIS. 2020.

[RR17] Fabian Renn und Frank Röger. Chase-*Tools (Teil I)*. Abschlussbericht KSWS, Unviersität Rostockl, DBIS. 2017.

[RSZ21] Ivo Kavisanczki Tobias Rudolph, Tom Siegl und Marian Zuska. *sql2sttgd*. Abschlussbericht KSWS, Universität Rostock, DBIS. 2021.

[Sam01] Pierangela Samarati. "Protecting Respondents' Identities in Microdata Release". In: *IEEE Trans. Knowl. Data Eng.* 13.6 (2001), S. 1010–1027.

[San14] Donatello Santoro. *Advanced Techniques for Mapping and Cleaning*. Dissertation, Roma Tre University, Department of Engineering. 2014.

[SBS18] Jose Luis Canovas Sanchez, Jorge Bernal Bernabé und Antonio F. Skarmeta. "Towards privacy preserving data provenance for the Internet of Things". In: *WF-IoT*. IEEE, 2018, S. 41–46.

[Sch20] Nic Scharlau. *Provenance und Privacy in ProSA*. Bachelorarbeit, Universität Rostock, DBIS. 2020.

[Sch22] Nic Scharlau. *Anonymisierung von Data Provenance in ProSA*. Masterarbeit, Universität Rostock, DBIS. 2022.

[Sen+18] Pierre Senellart, Louis Jachiet, Silviu Maniu und Yann Ramusat. "ProvSQL — Provenance and Probability Management in PostgreSQL". In: *Proc. VLDB Endow.* 11.12 (2018), S. 2034–2037.

[Sen17] Pierre Senellart. "Provenance and Probabilities in Relational Databases". In: *SIGMOD Rec.* 46.4 (2017), S. 5–15.

[Set74] Ravi Sethi. "Testing for the Church-Rosser Property". In: *J. ACM* 21.4 (1974), S. 671–679.

[SG10] Francesca Spezzano und Sergio Greco. "ChaseTermination — A Constraints Rewriting Approach". In: *Proc. VLDB Endow.* 3.1 (2010), S. 93–104. DOI: 10.14778/1920841.1920858.

[Sjø93] D. Sjøberg. "Quantifying schema evolution". In: *Inf. Softw. Technol.* 35.1 (1993), S. 35–44.

[Spe11] Francesca Spezzano. "On the Problem of Checking Chase Termination". Diss. Università della Calabria, 2011.

[Spo22] Dennis Spolwind. *Inverse Anfragen in ProSA*. Masterarbeit, Universität Rostock, DBIS. 2022.

[SS83] Schweizer und Sklar. *Probabilistic Metric Spaces*. https://www.planetmath.org/quasiinverseofafunction. 1983.

[Sto+93] Michael Stonebraker, Jolly Chen, Nobuko Nathan, Caroline Paxson und Jiang Wu. "Tioga — Providing Data Management Support for Scientific Visualization Applications". In: *VLDB*. Morgan Kaufmann, 1993, S. 25–38.

[STW09] Anish Das Sarma, Martin Theobald und Jennifer Widom. *LIVE — A Lineage-Supported Versioned DBMS*. Technical Report, Stanford InfoLab, http://ilpubs.stanford.edu:8090/926/, 2009.

[Sva16] Jan Svacina. *Intensional Answers for Provenance Queries in Big Data Analytics*. Bachelorarbeit, Universität Rostock, DBIS. 2016.

[Swe00] Latanya Sweeney. *Simple Demographics Often Identify People Uniquely*. Carnegie Mellon University, School of Computer Science, Data Privacy Lab White Paper SeriesLIDAP-WP4. Pittsburgh, PA. 2000.

[Swe01] Latanya Sweeney. "Computational disclosure control — A primer on data privacy protection". Diss. Massachusetts Institute of Technology, Cambridge, 2001.

[Swe02a] Latanya Sweeney. "Achieving k-Anonymity Privacy Protection Using Generalization and Suppression". In: *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 10.5 (2002), S. 571–588.

[Swe02b] Latanya Sweeney. "k-Anonymity — A Model for Protecting Privacy". In: *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 10.5 (2002), S. 557–570.

[Tan07] Wang Chiew Tan. "Provenance in Databases — Past, Current, and Future". In: *IEEE Data Eng. Bull.* 30.4 (2007), S. 3–12.

[Tau19] Anja Tausch. "Nachhaltigkeit von Forschungsdaten". In: *SRH Fernhochschule (eds) Nachhaltigkeit im interdisziplinären Kontext*. Springer, 2019. DOI: https://doi.org/10.1007/978-3-658-24288-6_9.

[Tor+17] Vicenç Torra, Guillermo Navarro-Arribas, David Sanchez-Charles und Victor Muntés-Mulero. "Provenance and Privacy". In: *MDAI*. Bd. 10571. LNCS. Springer, 2017, S. 3–11.

[Tre22] Nico Trebbin. *Nutzung von ChaTEAU für das Answering-Queries-using-Views-Problem (AQuV)*. Bachelorarbeit, Universität Rostock, DBIS. 2022.

[Ull97] Jeffrey D. Ullman. "Information Integration Using Logical Views". In: *ICDT*. Bd. 1186. LNCS. Springer, 1997, S. 19–40.

[Var03] Achille Varzi. *The Stanford Encyclopedia of Philosophy — Mereology*, *https://plato.stanford.edu/entries/mereology*. Hrsg. von Edward N. Zalta. abgerufen am 13.09.2021. 2003.

[Vim+12] Sabrina De Capitani di Vimercati, Sara Foresti, Giovanni Livraga und Pierangela Samarati. "Data Privacy — Definitions and Techniques". In: *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 20.6 (2012), S. 793–818.

[Wid04] Jennifer Widom. *Trio — A System for Integrated Management of Data, Accuracy, and Lineage*. Techn. Ber. 2004.

[Wid08] Jennifer Widom. "Trio — A System for Data, Uncertainty, and Lineage". In: *Managing and Mining Uncertain Data*. Springer, 2008.

[Wie+96] Janet L. Wiener, Himanshu Gupta, Wilburt Labio, Yue Zhuge, Hector Garcia-Molina und Jennifer Widom. "A System Prototype for Warehouse View Maintenance". In: *VIEWS*. 1996, S. 26–33.

[WN11] Shengfeng Wu und Iulian Neamtiu. "Schema Evolution Analysis for Embedded Databases". In: *ICDE Workshops*. IEEE Computer Society, 2011, S. 151–156.

[WS97] Allison Woodruff und Michael Stonebraker. "Supporting Fine-grained Data Lineage in a Database Visualization Environment". In: *ICDE*. IEEE Computer Society, 1997, S. 91–102.

[Wuc99] Kurt Wuchterl. *Methoden der Gegenwartsphilosophie, 3. erweiterte Auflage*. UTB, 1999.

[XT06] Xiaokui Xiao und Yufei Tao. "Personalized privacy preservation". In: *SIGMOD Conference*. ACM, 2006, S. 229–240.

[Yan18] Jiawei Yan. *Konzeption und Umsetzung einer Schemaextraktion von langjährigen In-Situ-Forschungsdaten aus der Ostsee*. Masterarbeit, Universität Rostock, DBIS. 2018.

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

[Zim21] Jakob Zimmer. *Datenintegration durch inverse Schemaabbildungen — Erweiterung der Rostocker GaLVE-Technik*. Masterarbeit, Universität Rostock, DBIS. 2021.

Teil VI.

Anhänge

A. Das Praxis-Beispiel

Wann immer an Sachverhalt an einem konkreten Beispiel erläutert wird, wählen wir hierzu das Studierenden-Beispiel. Dies betrifft insbesondere Kapitel 3 (Grundlagen) und 4 (Analyse verschiedener Tools) sowie die ab Kapitel 6 folgenden Ergebniskapitel. Der Vergleich der verschiedenen Chase- und Provenance-Tools erfolgt auf Effektivität, nicht auf Effizienz. Hierfür verwenden wir insgesamt elf Benchmark-Anfragen, welche in Abschnitt B vorgestellt werden.

Die Studierenden-Datenbank besteht aus den fünf Relationen `Student`(`studentID`, `lastName`, `firstName`, `studies`) (siehe Tabelle A.1a), `Course`(`courseID`, `title`) (siehe Tabelle A.1b), `Lecturer`(`courseID`, `lecturer`) (siehe Tabelle A.1c), `Grade`(`courseID`, `studentID`, `semester`, `grade`) (siehe Tabelle A.1d) und `Participant`(`courseID`, `studentID`) (siehe Tabelle A.1e). Alle Relationen besitzen einen einfach oder zusammengesetzten Schlüssel, welcher im Folgenden durch Unterstreichung hervorgehoben ist. Zusätzlich ist jedes Tupel einer Relation durch eine Provenance-ID eindeutig gekennzeichnet. Die ID setzt sich aus dem ersten Buchstaben des Relationennamens sowie einer fortlaufenden Nummer zusammen. So ist C_7 Provenance-ID des siebten Tupels (007, Law and Science) der Relation `Course`. Eine einmal vergeben Provenance-ID wird nach Löschung des zugehörigen Tupels nicht erneut vergeben. Für einen einfacheren Überblick notieren wir die Provenance-ID stets als letztes Attribut im Schema bzw. rechts der Relation.

studentID	lastName	firstName	studies	
1	Moore	Donald	Teaching	S_1
2	Morgan	Sarah	Mathematics	S_2
3	Wood	Jack	Engineering	S_3
4	Harrison	Elisabeth	Computer Science	S_4
5	Williams	John	Computer Science	S_5
6	William	Mary	Computer Science	S_6
7	Smith	Jack	Engineering	S_7
8	John	Jennifer	Theory	S_8

(a) `Student`

courseID	title	
001	Database Systems	C_1
002	Operating Systems	C_2
003	Artificial Intelligence	C_3
004	Probability Theory	C_4
005	Algorithms	C_5
006	Data Science	C_6
007	Law and Science	C_7
008	Data Warehouses	C_8
009	Cybersecurity	C_9

(b) `Course`

courseID	lecturer	
001	Professor A	L_1
001	Lecturer A	L_2
002	Professor B	L_3
003	Professor C	L_4
004	Professor D	L_5
005	Lecturer B	L_6
006	Professor D	L_7
007	Professor A	L_8
008	Professor E	L_9
009	Professor A	L_{10}

(c) `Lecturer`

Tabelle A.1. Relationen des Studierenden-Datenbank (Teil I).

courseID	studentID	semester	grade	
001	1	SS 16	2.0	G_1
001	2	SS 16	1.7	G_2
001	4	SS 16	1.7	G_3
001	5	SS 16	3.0	G_4
002	1	WS 15/16	3.7	G_5
002	2	WS 14/15	1.3	G_6
002	3	WS 14/15	2.3	G_7
002	4	WS 15/16	1.0	G_8
002	5	WS 15/16	1.3	G_9
002	6	WS 15/16	2.0	G_{10}
002	7	WS 15/16	3.3	G_{11}
003	1	WS 16/17	1.0	G_{12}
004	3	WS 16/17	1.3	G_{13}
004	5	WS 16/17	3.0	G_{14}
005	4	SS 17	2.7	G_{15}
005	7	SS 17	1.7	G_{16}
006	4	SS 17	2.7	G_{17}
006	5	SS 17	4.0	G_{18}
007	1	SS 16	2.3	G_{19}
007	3	SS 16	1.7	G_{20}
009	1	SS 16	3.3	G_{21}
009	5	SS 15	5.0	G_{22}
009	5	SS 16	2.7	G_{23}

(d) `Grade`

courseID	studentID	
001	1	P_1
001	2	P_2
001	4	P_3
001	5	P_4
002	1	P_5
002	2	P_6
002	3	P_7
002	4	P_8
002	5	P_9
002	6	P_{10}
002	7	P_{11}
003	1	P_{12}
004	3	P_{13}
004	5	P_{14}
004	7	P_{15}
005	4	P_{16}
005	7	P_{17}
006	4	P_{18}
006	5	P_{19}
007	1	P_{20}
007	3	P_{21}
007	5	P_{22}
008	3	P_{23}
009	1	P_{24}

(e) `Participant`

Tabelle A.1. Relationen des Studierenden-Datenbank (Teil II).

B. Die Benchmark-Anfragen

In Kapitel 4 haben wir die bekanntesten CHASE- und Provenance-Tools auf ihre Effektivität untersucht. Basis für diese Untersuchung sind insgesamt elf Benchmark-Anfragen. Sieben Anfragen (QA_1 bis QB_4) beschreiben verschiedene Spezialfälle bei der Anwendung des CHASE-Algorithmus. Die anderen vier Anfragen (QP_1 bis QP_4) decken die häufigsten SQL-Anfragen ab, welche im Zusammenhang mit Provenance interessant sind.

B.1. Provenance-Anfragen

```
SELECT DISTINCT studies
FROM student;
```

XML-Datei B.1 Test auf Duplikateliminierung (QP_1)

```
SELECT firstName
FROM student s INNER JOIN participant p
ON s.studentID = p.studentID
WHERE p.courseID = '005';
```

XML-Datei B.2 Selektion und Verbund (QP_2)

```
SELECT s.firstName, l.lecturer
FROM student s, participant p, lecturer l
WHERE s.studentID = p.studentID
AND p.courseID = l.courseID;
```

XML-Datei B.3 Verbund dreier Relationen (QP_3)

```
SELECT grade, count(*) AS gcount
FROM grade
WHERE courseID = '002'
GROUP BY grade
ORDER BY grade;
```

XML-Datei B.4 Aggregation und Gruppierung (QP_4)

Anfrage-Ergebnisse Es folgen die Ergebnisse der Provenance-Anfragen in Tabelle B.1. Zu beachten ist, dass Tabelle B.1c aus Platzgründen lediglich einen Ausschnitt des tatsächlichen Anfrageergebnisses darstellt. Zu sehen sind die Tupellisten und Relationennamen der ***where***-, die Zeugenbasen der ***why***- und die Polynome der ***how***-Provenance sowie die Data Lineage. Die Ergebnisse der CHASE-Anfragen sind unter „optimized query“ in der jeweiligen Anfrage (Anfrage B.5 bis B.11) bereits enthalten.

	where				Data
`studies`	relationen-basiert	tupel-basiert	***why***	***how***	`Lineage`
Teaching	`Student`	$\{S_1\}$	$\{\{S_1\}\}$	S_1	S_1
Mathematics	`Student`	$\{S_2\}$	$\{\{S_2\}\}$	S_2	S_2
Engineering	`Student`	$\{S_3, S_7\}$	$\{\{S_3, S_7\}\}$	$S_3 + S_7$	S_3, S_7
Computer Science	`Student`	$\{S_4, S_5, S_6\}$	$\{\{S_4, S_5, S_6\}\}$	$S_4 + S_5 + S_6$	S_4, S_5, S_6
Theory	`Student`	$\{S_8\}$	$\{\{S_8\}\}$	S_8	S_8

(a) Ergebnis der Anfrage QP_1

	where				Data
`studies`	relationen-basiert	tupel-basiert	***why***	***how***	`Lineage`
Elisabeth	`Student`, `Participant`	$\{S_4, P_{16}\}$	$\{\{S_4, P_{16}\}\}$	$S_4 \cdot P_{16}$	S_4
Jack	`Student`, `Participant`	$\{S_7, P_{17}\}$	$\{\{S_7, P_{17}\}\}$	$S_7 \cdot P_{17}$	S_7

(b) Ergebnis der Anfrage QP_2

		where				Data
`firstName`	`lecturer`	relationen-basiert	tupel-basiert	***why***	***how***	`Lineage`
Donald	Professor A	`Student`, `Lecturer`	$\{S_1, P_1, L_1\}$	$\{\{S_1, P_1, L_1\}\}$	$S_1 \cdot P_1 \cdot L_1$	S_1, L_1
Donald	Lecturer A	`Student`, `Lecturer`	$\{S_1, P_1, L_2\}$	$\{\{S_1, P_1, L_2\}\}$	$S_1 \cdot P_1 \cdot L_2$	S_1, L_2
Donald	Lecturer B	`Student`, `Lecturer`	$\{S_1, P_5, L_6\}$	$\{\{S_1, P_5, L_6\}\}$	$S_1 \cdot P_5 \cdot L_6$	S_1, L_6
Sarah	Professor A	`Student`, `Lecturer`	$\{S_2, P_2, L_1\}$	$\{\{S_2, P_2, L_1\}\}$	$S_2 \cdot P_2 \cdot L_1$	S_2, L_1
Sarah	Lecturer A	`Student`, `Lecturer`	$\{S_2, P_2, L_2\}$	$\{\{S_2, P_2, L_2\}\}$	$S_2 \cdot P_2 \cdot L_2$	S_2, L_2
Elisabeth	Professor A	`Student`, `Lecturer`	$\{S_3, P_7, L_3\}$	$\{\{S_3, P_7, L_3\}\}$	$S_3 \cdot P_7 \cdot L_3$	S_3, L_3
...	...	...	...	...	...	...

(c) Ergebnis der Anfrage QP_3

		where				Data
`grade`	`gcount`	relationen-basiert	tupel-basiert	***why***	***how***	`Lineage`
3.7	1	`Grade`	$\{G_5\}$	$\{\{G_5\}\}$	G_5	G_5
1.3	2	`Grade`	$\{G_6, G_9\}$	$\{\{G_6\}, \{G_9\}\}$	$G_6 + G_9$	G_6, G_9
2.3	1	`Grade`	$\{G_7\}$	$\{\{G_7\}\}$	G_7	G_7
1.0	1	`Grade`	$\{G_8\}$	$\{\{G_8\}\}$	G_8	G_8
2.0	1	`Grade`	$\{G_{10}\}$	$\{\{G_{10}\}\}$	G_{10}	G_{10}
3.3	1	`Grade`	$\{G_{11}\}$	$\{\{G_{11}\}\}$	G_{11}	G_{11}

(d) Ergebnis der Anfrage QP_4

Tabelle B.1. Ergebnisse der Provenance-Anfragen QP_1 bis QP_4.

B.2. Chase-Anfragen

schema: $R_1(A_1, A_2)$, $R_2(A_2, A_3)$
tgd: $\forall a_1, a_2 : R_1(a_1, a_2) \rightarrow \exists a_3 : R_2(a_2, a_3)$
query: $\pi_{a_1}(r(R_1) \bowtie r(R_2))$
optimized query: $\pi_{a_1}(r(R_1))$

XML-Datei B.5 Optimierung von Schlüssel-Fremdschlüssel-Verbunden (QA_1)

schema: $R(A_1, A_2, A_3)$
egd: $\forall a_1, ..., a_5 : R(a_1, a_2, a_3) \wedge R(a_4, a_2, a_5) \rightarrow (a_3 = a_5)$
query: $\pi_{A_1,A_2}(r(R)) \bowtie \pi_{A_2,A_3}(r(R))$
optimized query: $r(R)$

XML-Datei B.6 Verbundtreue-Kriterium mit einer FD (QA_2)

schema: $R(A_1, A_2, A_3)$
tgd: $\forall a_1, ..., a_5 : R(a_1, a_2, a_3) \wedge R(a_4, a_2, a_5) \rightarrow R(a_1, a_2, a_5)$
query: $\pi_{A_1,A_2}(r(R)) \bowtie \pi_{A_2,A_3}(r(R))$
optimized query: $r(R)$

XML-Datei B.7 Verbundtreue-Kriterium mit einer JD (QA_3)

schema: $R_1(A_1, A_2), R_2(A_1, A_3), R_3(A_1, A_4), R_4(A_1, A_5), R_5(A_1, A_6)$
IND: $R_5(A_1) \subseteq R_4(A_1) \subseteq R_3(A_1) \subseteq R_2(A_1) \subseteq R_1(A_1)$
query: $\pi_{A_1,A_2}(\sigma_{A_6=10}(R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5))$
optimized query $\pi_{A_1,A_2}(r(R_1) \bowtie r(R_5))$

XML-Datei B.8 Kette von Schlüssel-Fremdschlüssel-Verbunden (QB_1)

schema: $R_1(A_1, A_2), R_2(A_1, A_3), R_3(A_1, A_4), R_4(A_4, A_5), R_5(A_4, A_6)$
IND: $R_5(A_4) \subseteq R_4(A_4) \subseteq R_3(A_4)$, $R_3(A_1) \subseteq R_2(A_1) \subseteq R_1(A_1)$
query: $\pi_{A_1,A_2}(\sigma_{A_6=10}(R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5))$
optimized query: $\pi_{A_1,A_2}(\sigma_{A_6=10}(r(R_1) \bowtie r(R_3) \bowtie r(R_5)))$

XML-Datei B.9 Kette von Schlüssel-Fremdschlüssel-Verbunden mit wechselnden Schlüsseln (QB_2)

schema: $R_1(A_1, A_2), R_2(A_1, A_3), R_3(A_1, A_4), R_4(A_1, A_4, A_5), R_5(A_1, A_4, A_6)$
IND: $R_5(A_4) \subseteq R_4(A_4) \subseteq R_3(A_4)$, $R_5(A_1) \subseteq R_4(A_1) \subseteq R_3(A_1) \subseteq R_2(A_1) \subseteq R_1(A_1)$
query: $\pi_{A_1,A_2}(\sigma_{A_6=10}(R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5))$
optimized query: $\pi_{A_1,A_2}(\sigma_{A_6=10}(R_1 \bowtie R_5))$

XML-Datei B.10 konkurrierende Ketten von Schlüssel-Fremdschlüssel-Verbindungen mit wechselnden Schlüsseln (QB_3)

schema: $R_1(A_1, A_2, A_3), R_2(A_4, A_5, A_6)$
IND: $R_2(A_4, A_5) \subseteq R_1(A_1, A_2)$
FD: $A_1 \rightarrow A_2$
query: $\pi_{A_4,A_5}(\pi_{A_4,A_5}(R_2) \bowtie \sigma_{A_6=5}(\pi_{A_4,A_6}(R_2)))$
optimized query: $\pi_{A_4,A_5}(\sigma_{A_6=5}(r(R_2)))$

XML-Datei B.11 Multi-Attribut-Fremdschlüssel mit interner FD (QB_4)

C. Klassifikation der Chase-Inversen (Beweise)

In diesem Abschnitt werden die Beweise für alle elf Auswertungsoperatoren und 15 Schemaänderungsoperatoren aufgeführt. Die Lemmata werden pro Klasse formuliert und bewiesen.

Klasse 1: Provenance Invariant Zusätzliche Provenance-Informationen führen nicht immer zu einer Verbesserung des Chase-Inversentyps. Dies betrifft etwa die Hälfte der klassifizierten Schemaevolutionsoperatoren wie `COPY Table`, `CREATE Table`, `PARTITION Table` und `RENAME Table` sowie `ADD Column`, `COPY Column` und `RENAME Column`. Ebenso wie bei den meisten Auswertungsoperatoren wie der Identitätsabbildung, der Umbenennung, der Selektion, der Schnittmenge oder den vier skalare Operationen $\theta \in \{+, -, \cdot, :\}$ ändert sich der Inversentyp trotz der zusätzlichen Informationen nicht.

Lemma C.1. *Die Identitätsabbildung hat eine exakte* Chase*-Inverse.*

Beweis. Seien S_t und S_{t+1} zwei Datenbankschemata einer sich ändernden Datenbank und $Q : S_t \to S_{t+1}$ die Anfrage `SELECT` $*$ `FROM` R. Sei $\mathcal{M} = (S, S', \Sigma)$ die s-t tgd, welche die Selektion beschreibt und $\mathcal{M}^* = (S', S, \Sigma^{-1})$ die zugehörige Inverse, formalisiert über

$$\Sigma = \{R(a,b,c) \to R(a,b,c)\} \quad \text{und} \quad \Sigma^{-1} = \{R(a,b,c) \to R(a,b,c)\}.$$

O.B.d.A. sei R auf drei Attribute beschränkt. Ferner sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz mit verschiedenen drei-elementigen Tupeln. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^{-1}}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^{-1}}(\textsc{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\ &= \textsc{Chase}_{\mathcal{M}^{-1}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Es existiert somit eine exakte Chase-Inverse. Zusätzliche Provenance-Informationen sind notwendig. □

Lemma C.2. *Die Schnittmenge hat eine relaxte* Chase*-Inverse.*

Beweis. Seien S_t und S_{t+1} zwei Datenbankschemata einer sich ändernden Datenbank und $Q : S_t \to S_{t+1}$ die SQL-Anfrage `SELECT` $*$ `FROM` R `INTERSECT` V. Sei weiter $\mathcal{M} = (S, S', \Sigma)$ die Schemaabbildung, welche Q beschreibt und $\mathcal{M}^* = (S', S, \Sigma^{-1})$ die zugehörige inverse Funktion, formalisiert über

$$\Sigma = \{R(a,b) \wedge V(a,b) \to T(a,b)\} \quad \text{und} \quad \Sigma^{-1} = \{T(a,b) \to R(a,b) \wedge V(a,b)\}.$$

O.B.d.A. seien R, V und T auf jeweils zwei Attribute beschränkt. Ferner sei $I = \{R(x_{i_1}, x_{i_2}), R(x_{i_3}, x_{i_4}), V(x_{i_1}, x_{i_2}) \mid i_1, i_2, i_3, i_4 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz mit verschiedenen zwei-elementigen Tupeln. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^{-1}}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^{-1}}(\textsc{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}), R(x_{i_3}, x_{i_4}), V(x_{i_1}, x_{i_2}) \mid i_1, i_2, i_3, i_4 \in \mathbb{N}\})) \\ &= \textsc{Chase}_{\mathcal{M}^{-1}}(\{T(x_{i_1}, x_{i_2}) \mid i_1, i_2 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}), V(x_{i_1}, x_{i_2}) \mid i_1, i_2 \in \mathbb{N}\} \\ &\preccurlyeq I. \end{aligned}$$

Es existiert somit eine relaxte Chase-Inverse. Zusätzliche Provenance-Informationen ändern hieran nichts. □

Lemma C.3. *Die Selektion hat eine relaxte* Chase*-Inverse.*

Beweis. Seien S und S' zwei gegebene Schemata. Seien weiter $\mathcal{M} = (S, S', \Sigma)$ die Schemaabbildung, welche die Selektion beschreibt und $\mathcal{M}^* = (S', S, \Sigma^{-1})$ die zugehörige inverse Funktion, formalisiert über

$$\Sigma = \{\exists \alpha \in \mathbb{D}(b) : R(a,b,c) \wedge b\theta\alpha \to T(a,b,c)\} \quad \text{und} \quad \Sigma^{-1} = \{T(a,b,c) \to R(a,b,c)\}.$$

O.B.d.A. sei R auf drei Attribute beschränkt. Ferner sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz mit verschiedenen drei-elementigen Tupeln. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^{-1}}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^{-1}}(\text{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\ &= \text{Chase}_{\mathcal{M}^{-1}}(\{T(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge a_{i_2}\theta\alpha\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge a_{i_2}\theta\alpha\} \\ &\preccurlyeq I. \end{aligned}$$

Es existiert somit eine relaxte Chase-Inverse. Zusätzliche Provenance-Informationen ändern hieran nichts. □

Lemma C.4. *Die skalare Operationen haben jeweils eine exakte* Chase-*Inverse.*

Beweis. Seien S_t und S_{t+1} zwei Datenbankschemata einer sich ändernden Datenbank und $Q : S_t \to S_{t+1}$ eine Abbildung mit skalarem Operator $\theta \in \{+, -, \cdot, :\}$. Sei weiter $\mathcal{M} = (S, S', \Sigma)$ die Schemaabbildung, welche Q beschreibt und $\mathcal{M}^* = (S', S, \Sigma^{-1})$ die zugehörige inverse Funktion, formalisiert über

$$\begin{aligned} \Sigma &= \{\exists\alpha \in \mathbb{R} : R(a, b, c) \wedge d = b\theta\alpha \to T(a, d, c)\}, \\ \Sigma^{-1} &= \{\exists\alpha \in \mathbb{R} : T(a, b, c) \wedge b = d\theta^{-1}\alpha \to R(a, b, c)\}. \end{aligned}$$

O.B.d.A. seien R und T auf jeweils drei Attribute beschränkt und θ die Addition $+$. Ferner sei $I = \{R(x_{i_1}, x_i, x_{i_2}) \mid i, i_1, i_2 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz mit verschiedenen drei-elementigen Tupeln und $c \in \mathbb{R}$ eine beliebige Konstante. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^{-1}}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^{-1}}(\text{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_i, x_{i_2}) \mid i, i_1, i_2 \in \mathbb{N}\})) \\ &= \text{Chase}_{\mathcal{M}^{-1}}(\{T(x_{i_1}, x, x_{i_2}) \mid i_1, i_2 \in \mathbb{N} \wedge x = x_i + c\}) \\ &= \{R(x_{i_1}, x_i, x_{i_2}) \mid i_1, i_2 \in \mathbb{N} \wedge x_i = x - c\} \\ &= I. \end{aligned}$$

Es existiert somit eine relaxte Chase-Inverse. Zusätzliche Provenance-Informationen sind notwendig. □

Lemma C.5. `COPY Table` *hat eine exakte* Chase-*Inverse.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `COPY Table` ein Evolutionsschritt von S_t zu S_{t+1}. `Copy Table` kann auf zwei verschiedene Arten formalisiert werden. So kann der Operator als eine s-t tgd oder als Menge von zwei s-t tgds dargestellt werden.

Seien zunächst $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{R(a, b, c) \to R'(a, b, c) \wedge V(a, b, c)\} \quad \text{und} \quad \Sigma^{-1} = \{R'(a, b, c) \wedge V(a, b, c) \to R(a, b, c)\},$$

die `COPY Table` und die entsprechende inverse Abbildung formalisieren. O.B.d.A. seien R, R' und V jeweils auf drei Attribute beschränkt. Sei ferner $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\ &= \text{Chase}_{\mathcal{M}^*}(\{R'(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Es ist kein Informationsverlust zu verzeichnen. Insgesamt ergibt sich so eine exakte Chase-Inverse. Da wir bereits eine exakte Inverse garantieren können, ist das Hinzufügen zusätzlicher Provenance-Informationen wie einer Zeugenbasis oder eines Provenance-Polynoms nicht erforderlich.

Seien alternativ $\mathcal{M}$ und $\mathcal{M}^*$ definiert über

$$\begin{aligned} \Sigma &= \{R(a, b, c) \to R'(a, b, c), R(a, b, c) \to V(a, b, c)\}, \\ \Sigma^{-1} &= \{R'(a, b, c) \to R(a, b, c), V(a, b, c) \to R(a, b, c)\}. \end{aligned}$$

Seien wiederum R, R' und V jeweils auf drei Attribute beschränkt und $I^* = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz. Dann liefert die Anwendung des CHASE&BACKCHASE:

$$\begin{aligned} I^* &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)) \\ &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\ &= \text{CHASE}_{\mathcal{M}^*}(\{R(x_{i_1}, x_{i_2}, x_{i_3}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Auch hier erhalten wir für `COPY Column` eine exakte CHASE-Inverse. Zusätzliche Provenance-Informationen sind wiederum nicht notwendig. □

Lemma C.6. *`CREATE Table` hat eine exakte* CHASE*-Inverse.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `CREATE Table` ein Evolutionsschritt von S_t zu S_{t+1}. Sei $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{\emptyset \to \exists A, B, C : R(A, B, C)\} \quad \text{und} \quad \Sigma^{-1} = \{R(a, b, c) \to \emptyset\},$$

welche `COPY Column` und seine entsprechende inverse Abbildung formalisieren. O.b.d.A. sei R auf drei Attribute beschränkt. Ferner sei $I = \emptyset$ eine gegebenen Instanz. Die liefert die Anwendung des CHASE&BACKCHASE:

$$\begin{aligned} I^* &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)) \\ &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(\emptyset)) \\ &= \text{CHASE}_{\mathcal{M}^*}(\{R(\eta_1, \eta_2, \eta_3)\}) \\ &= \emptyset \\ &= I. \end{aligned}$$

Wegen $I^* = I$ existiert für `CREATE Table` somit stets eine exakte CHASE-Inverse. Zusätzliche Provenance-Informationen ändern hieran nichts. □

Lemma C.7. *`PARTITION Table` bzw. `DISTRIBUTE Table` hat eine exakte* CHASE*-Inverse.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `PARTITION Table` bzw. `DISTRIBUTE Table` ein Evolutionsschritt von S_t zu S_{t+1}. Seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\begin{aligned} \Sigma &= \{R(a, b, c) \wedge \texttt{cond}_A \to V(a, b, c),\ R(a, b, c) \wedge \neg\texttt{cond}_A \to T(a, b, c)\} \\ \Sigma^{-1} &= \{V(a, b, c) \to R(a, b, c), T(a, b, c) \to R(a, b, c)\}, \end{aligned}$$

welche `PARTITION Table` und seine zugehörige inverse Abbildung formalisieren. O.B.d.A. seien R, V und T jeweils auf drei Attribute beschränkt. Seien ferner $\texttt{cond}_A$ eine gegebene Bedingung und $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}), R(x_{j_1}, x_{j_2}, x_{j_3}) \mid i_1, i_2, i_3, j_1, j_2, j_3 \in \mathbb{N}\}$ eine beliebig gewählte Instanz. Dann liefert die Anwendung des CHASE&BACKCHASE:

$$\begin{aligned} I^* &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)) \\ &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}), R(x_{j_1}, x_{j_2}, x_{j_3}) \mid i_1, i_2, i_3, j_1, j_2, j_3 \in \mathbb{N}\})) \\ &= \text{CHASE}_{\mathcal{M}^*}(\{V(x_{i_1}, x_{i_2}, x_{i_3}), T(x_{j_1}, x_{j_2}, x_{j_3}) \mid i_1, i_2, i_3, j_1, j_2, j_3 \in \mathbb{N} \\ &\qquad \wedge\ V(x_{i_1}, x_{i_2}, x_{i_3}) \text{ erfüllt } \texttt{cond}_A \wedge\ T(x_{j_1}, x_{j_2}, x_{j_3}) \text{ erfüllt nicht } \texttt{cond}_A\} \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}), R(x_{j_1}, x_{j_2}, x_{j_3}) \mid i_1, i_2, i_3, j_1, j_2, j_3 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Mit $\text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)) = I$ ergibt dies eine exakte CHASE-Inverse für `PARTITION Table`. Zusätzliche Provenance-Informationen verbessern den inversen Typ nicht weiter. □

Lemma C.8. *`RENAME Table` hat eine exakte* CHASE*-Inverse.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich ändernden Datenbank und `RENAME Table` ein Evolutionsschritt von S_t zu S_{t+1}. Seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{R(a, b, c) \to V(a, b, c)\} \quad \text{und} \quad \Sigma^{-1} = \{V(a, b, c) \to R(a, b, c)\},$$

welche `RENAME Table` und seine zugehörige inverse Abbildung formalisieren. O.b.d.A. seien R und V jeweils auf drei Attribute beschränkt. Ferner sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gegebene Instanz.

Dann liefert die Anwendung von Chase&Backchase

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\ &= \text{Chase}_{\mathcal{M}^*}(\{V(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Insgesamt ergibt dies eine exakte Chase-Inverse für `RENAME Table`. Auch hier verbessert das Hinzufügen von Provenance den Inversentyp nicht. □

Lemma C.9. *`ADD Column` hat eine exakte Chase-Inverse.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `ADD COLUMN` ein Evolutionsschritt von S_t zu S_{t+1}. Seien weiter $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen

$$\Sigma = \{R(a,b,c) \to V(a,b,c, \textbf{const} \mid f(a,b,c))\} \quad \text{und} \quad \Sigma^{-1} = \{V(a,b,c, \textbf{const} \mid f(a,b,c)) \to R(a,b,c)\},$$

welche `ADD Column` und seine entsprechende Inverse formalisieren. Die vierte Stelle kann eine beliebige Konstante **const** sein oder durch eine mehrwertige, nicht unbedingt invertierbare Funktion f berechnet werden. O.B.d.A. seien R, V und f jeweils auf drei Attribute beschränkt. Sei weiter $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebige Instanz. Dann liefert der Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\ &= \text{Chase}_{\mathcal{M}^*}(\{V(x_{i_1}, x_{i_2}, x_{i_3}, \eta \mid f(x_{i_1}, x_{i_2}, x_{i_3})) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Die Instanz $J = \text{Chase}_{\mathcal{M}}(I)$ kann in ihrer vierten Position sowohl einen Nullwert η als auch einen mittels f berechneten Wert enthalten. Dieser wird jedoch in der Backchase-Phase entfernt. Aufgrund von $I^* = I$ erhalten wir eine exakte Chase-Inverse für `ADD Column`. Durch zusätzliche Provenance-Informationen ändert sich der inverse Typ nicht.

Ist hingegen der addierte Wert unbekannt, d.h. $\mathcal{M}$ und $\mathcal{M}^{-1}$ sind definiert über

$$\Sigma = \{R(a,b,c) \to \exists D : V(a,b,c,D)\} \quad \text{und} \quad \Sigma^{-1} = \{V(a,b,c,d) \to R(a,b,c)\},$$

dann liefert die Anwendung des Chase&Backchase $I^* = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$. Dies garantiert ebenfalls eine exakte Chase-Inverse für `ADD Column`. □

Lemma C.10. *`COPY Column` hat eine exakte Chase-Inverse.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei weiter `COPY Column` ein Evolutionsschritt von S_t nach S_{t+1}. `Copy Column` kann auf zwei verschiedene Arten formalisiert werden. So kann der Operator zum einen als eine Menge von zwei s-t tgds dargestellt werden – *ein Attribut c wird von einer Tabelle R in eine zweite Tabelle T kopiert* – oder als eine einzige s-t tgd – *ein Attribut c wird innerhalb einer Tabelle R kopiert.*

Seien zunächst $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\begin{aligned} \Sigma &= \{R(a,b,c) \wedge V(d,e) \wedge \texttt{cond}_A \to R'(a,b,c) \wedge T(c,d,e), \\ &\quad\ R(a,b,c) \wedge V(d,e) \wedge \neg\texttt{cond}_A \to \exists C : R'(a,b,c) \wedge T(C,d,e)\} \\ \Sigma^{-1} &= \{R(a,b,c) \to R(a,b,c), T(c,d,e) \to V(d,e)\}, \end{aligned}$$

welche `COPY Column` und ihre entsprechende inverse Abbildung formalisieren. O.B.d.A. seien R, R' und T jeweils auf drei Attribute beschränkt und V auf zwei Attribute beschränkt. Ferner sei $\texttt{cond}_A : c = d$ die Bedingung für eine Attributauswahl. Sei weiter $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{j_1}, x_{j_2}), V(x_{j_3}, x_{j_4}) \mid i_1, i_2 i, i_3, j_1, j_2, j_3, j_4 \in \mathbb{N} \wedge x_{i_3} = x_{j_1} \wedge x_{i_3} \neq x_{j_3}\}$ eine beliebig gewählte Instanz. D.h. $V(x_{j_1}, x_{j_2})$ erfüllt $\texttt{cond}_A$, $V(x_{j_3}, x_{j_4})$ jedoch nicht.

Dann ergibt die Anwendung des Chase&Backchase:

$$\begin{aligned}
I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\
&= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{j_1}, x_{j_2}), V(x_{j_3}, x_{j_4}) \mid i_1, i_2 i, i_3, j_1, j_2, j_3, j_4 \in \mathbb{N} \\
&\quad \wedge x_{i_3} = x_{j_1} \wedge x_{i_3} \neq x_{j_3}\})) \\
&= \textsc{Chase}_{\mathcal{M}^*}(\{R'(x_{i_1}, x_{i_2}, x_{i_3}), T(x_{i_3}, x_{j_1}, x_{j_2}), T(\eta, x_{j_3}, x_{j_4}) \mid i_1, i_2, i_3, j_1, j_2, j_3, j_4 \in \mathbb{N} \\
&\quad \wedge x_{i_3} = x_{j_1} \wedge x_{i_3} \neq x_{j_3}\}) \\
&= \{R(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{j_1}, x_{j_2}), V(x_{j_3}, x_{j_4}) \mid i_1, i_2, i_3, j_1, j_2, j_3, j_4 \in \mathbb{N} \wedge x_{i_3} = x_{j_1} \wedge x_{i_3} \neq x_{j_3}\} \\
&= I.
\end{aligned}$$

Dies liefert eine exakte Chase-Inverse, die nicht durch zusätzliche Provenance-Informationen beeinflusst wird.

Seien alternativ $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ definiert über

$$\Sigma = \{R(a, b, c) \to T(a, b, c, c)\} \text{ und } \Sigma^{-1} = \{T(a, b, c, c) \to R(a, b, c)\}.$$

Seien nun R auf drei und T auf vier Attribute beschränkt. Sei wiederum $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Instanz. Dann liefert die Anwendung der Chase&Backchase:

$$\begin{aligned}
I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\
&= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\
&= \textsc{Chase}_{\mathcal{M}^*}(\{R(x_{i_1}, x_{i_2}, x_{i_3}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\
&= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\
&= I.
\end{aligned}$$

Auch hier erhalten wir eine exakte Chase-Inverse für `COPY Column`. Zusätzliche Provenance-Informationen sind wiederum nicht notwendig. □

Lemma C.11. *Die Umbenennung bzw.* ***`RENAME Column`*** *haben eine exakte* Chase*-Inverse.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich ändernden Datenbank und `RENAME Column` ein Evolutionsschritt von S_t zu S_{t+1}. Seien alternativ S_t und S_{t+1} zwei Datenbankschemata einer sich ändernden Datenbank und $Q : S_t \to S_{t+1}$ eine gegeben Umbenennungs-Anfrage. Sei weiter $\mathcal{M} = (S, S', \Sigma)$ die s-t tgd, welche die Umbennennung bzw. `RENAME Coluumn` beschreibt und $\mathcal{M}^* = (S', S, \Sigma^{-1})$ die zugehörige inverse Funktion, formalisiert über

$$\Sigma = \{R(a, b, c) \to T(a, b, c)\} \quad \text{und} \quad \Sigma^{-1} = \{T(a, b, c) \to R(a, b, c)\}.$$

O.B.d.A. seien R und T auf drei Attribute beschränkt. Ferner sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Quell-Instanz mit verschiedenen drei-elementigen Tupeln. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned}
I^* &= \textsc{Chase}_{\mathcal{M}^{-1}}(\textsc{Chase}_{\mathcal{M}})(I)) \\
&= \textsc{Chase}_{\mathcal{M}^{-1}}(\textsc{Chase}_{\mathcal{M}}(\{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\})) \\
&= \textsc{Chase}_{\mathcal{M}^{-1}}(\{T(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\
&= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\
&= I.
\end{aligned}$$

Für die Umbenennung sowie `RENAME Column` existiert somit eine exakte Chase-Inverse. Zusätzliche Provenance-Informationen sind hier nicht notwendig. □

Klasse 2: Dangling Tuples Je nach Quell-Instanz können „Dangling Tuples“ auftreten. In diesem Fall kann eine exakte Chase-Inverse nicht garantiert werden. Auch das Hinzufügen von Zeugenbasen und Provenance-Polynomen ist möglicherweise nicht ausreichend. Durch die Speicherung der verlorenen Tupel in so genannten Side Tables können wir jedoch exakte Chase-Inversen angeben.

Lemma C.12. *Der natürliche Verbund zweier Relationen sowie die SMO* `JOIN Table` *haben eine exakte* Chase*-Inverse. Wenn Dangling Tuples vorliegen, kann eine relaxte* Chase*-Inverse garantiert werden. Zusätzliche Provenance-Informationen wie Zeugenbasen und Side Tables ermöglichen die Angabe einer exakten* Chase*-Inversen.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `JOIN Table` ein Evolutionsschritt von S_t zu S_{t+1}. Seien alternativ S_t und S_{t+1} zwei Datenbankschemata einer sich ändernden Datenbank und $Q : S_t \to S_{t+1}$ die SQL-Anfrage `SELECT` $*$ `FROM` R `NATURAL JOIN` V. Dann seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{R(a,b) \wedge V(a,c) \to T(a,b,c)\} \quad \text{und} \quad \Sigma^{-1} = \{T(a,b,c) \to R(a,b) \wedge V(a,c)\},$$

welche den Verbund zweier Relationen bzw. `JOIN Table` und ihre zugehörige inverse Abbildung formalisieren. O.B.d.A. haben R und V jeweils zwei Attribute mit einem gemeinsamen Verbund-Attribut a. Seien weiter T auf drei Attribute beschränkt und $I = \{R(x_{i_1}, x_{i_2}), V(x_{i_1}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}$ eine beliebig gewählte Instanz. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}), V(x_{i_1}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Insgesamt erhalten wir eine exakte Chase-Inverse. Zusätzliche Provenance ändert den Chase-Inversentyp nicht.

Liegen Dangling-Tupeln vor, kann eine relaxte Chase-Inverse garantiert werden. O.B.d.A. sei $t_k = R(x_{k_1}, x_{k_2})$ ein Dangling-Tupel in R und $I = \{R(x_{i_1}, x_{i_2})_{r_i}, V(x_{j_1}, x_{j_2})_{v_j} \mid i, i_1, i_2, j, j_1, j_2 \in \mathbb{N} \wedge x_{i_1} = x_{j_1}$ für fast alle i_1, j_1 $\wedge \exists k : t_k$ Dangling Tuple in $R \wedge r_i, v_j$ Tupelidentifikator des i-ten oder j-ten Tupels in R bzw. $V\}$ eine beliebig gewählte Instanz. Weiter sei w Zeugenbasis eines Tupel $t \in T$. Wird keine Provenance verwendet, können r_i und v_j unterdrückt werden. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2}, x_{j_2})_w \mid i_1, i_2, j_2 \in \mathbb{N} \wedge w = \{\{r_i, v_j\}\} \text{ Zeugenbasis}\} \\ &\qquad \cup \{R(x_{k_1}, x_{k_2})_{r_k} \mid k, k_1, k_2 \in \mathbb{N} \wedge \ r_k \text{ Tupelidentifikator} \wedge \ t_k = V(x_{k_1}, x_{k_2}) \text{ Dangling Tuple}\}) \\ &= \{R(x_{i_1}, x_{i_2}), V(x_{j_1}, x_{j_2}) \mid i \in \mathbb{N}\} \\ &= I \end{aligned}$$

Ohne zusätzliche Informationen kann das Dangling-Tupel $t_k = R(x_{k_1}, x_{k_2})$ nicht rekonstruiert werden, d.h. es gilt $I^* \preccurlyeq I$. In diesem Fall kann eine relaxte Chase-Inverse angegeben werden. Da der Tupel-Identifikator des Dangling-Tupels t_k nicht in w enthalten ist, erhalten wir durch die Hinzunahme von Provenance keine zusätzlichen Informationen.

Addieren wir zusätzlich Zeugenbasen und Side Tables $\{R(x_{k_1}, x_{k_2})_{r_k} \mid k, k_1, k_2 \in \mathbb{N} \wedge\ r_k$ Tupelidentifikator$\}$ (blau hervorgehoben) verhindert dies den Informationsverlust. Die Side Tables — eine Tabelle pro Quell-Relation — speichern die Dangling Tuples, welche andernfalls verlorenen gehen würden. In der Backchase-Phase, nachdem alle inversen s-t tgds verarbeitet wurden, werden die in den Side Tables gespeicherten Tupel zurück in die Relationen R bzw. V geschrieben. So kann eine exakte Chase-Inverse garantiert werden. Die Chase-Inverse vom natürlichen Verbund bzw. `JOIN Table` hängt somit von der Existenz von Duplikaten ab. Sie ist entweder exakt oder relaxt. □

Lemma C.13. `MOVE Column` *hat eine exakte* Chase*-Inverse. Wenn Dangling Tuples und/oder Duplikate vorhanden sind, kann dies durch Hinzufügen von zusätzlicher Provenance-Informationen wie Zeugenbasen und Side Tables gewährleistet werden. In einigen dieser Fälle kann nur eine ergebnisäquivalente, in anderen Fällen eine relaxte* Chase*-Inverse angegeben werden.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `MOVE Column` ein Evolutionsschritt von S_t zu S_{t+1}. Seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\begin{aligned} \Sigma &= \{R(a,b,c) \wedge V(d,e) \wedge \texttt{cond}_A \to W(a,b) \wedge T(c,d,e)\} \\ \Sigma^{-1} &= \{W(a,b) \wedge T(c,d,e) \wedge \texttt{cond}_A \to R(a,b,c) \wedge V(d,e)\}, \end{aligned}$$

welche sowohl `MOVE Column` als auch die entsprechende inverse Abbildung formalisieren. O.B.d.A. seien R und T auf drei Attribute und V sowie W auf zwei Attribute beschränkt. Seien ferner $\texttt{cond}_A$ eine Bedingung, welche die

Verschiebung des Attributes c beschreibt, und $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{j_1}, x_{j_2}) \mid i_1, i_2, i_3, j_1, j_2 \in \mathbb{N}\}$ eine beliebig gewählte Instanz. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{W(x_{i_1}, x_{i_2}), T(x_{i_3}, x_{j_1}, x_{j_2}) \mid i_1, i_2, i_3, j_1, j_2 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{j_1}, x_{j_2}) \mid i_1, i_2, i_3, j_1, j_2 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Wir erhalten eine exakte Chase-Inverse. Das Hinzufügen der zusätzlichr Provenance-Informationen ändert den Typ der Inversen nicht.

Dies ändert sich jedoch mit dem Vorhandensein von Dangling Tuples und Duplikaten. Seien hierfür o.B.d.A. $t_k = V(x_{j_1}, x_{j_2})$ ein Dangling Tuple in V und r_m, r_n Tupelidentifikatoren zweier Duplikate in W, die durch Anwendung es Chase erzeugt wurden. Sei ferner w Zeugenbasis und I eine beliebig gewählte Instanz mit $I = \{R(x_{i_1}, x_{i_2}, x_{i_3})_{r_i}, V(x_{j_1}, x_{j_2})_{v_j} \mid i, i_1, i_2, i_3, j, j_1, j_2 \in \mathbb{N} \wedge x_{i_1} = x_{j_1}$ für beinahe alle $i, j \wedge \exists k : t_k$ Dangling Tuple in $V \wedge r_i, v_j$ Tupelidentifikatoren des i-ten und j-ten Tupels in R und $V\}$ eine Instanz. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{W(x_{i_1}, x_{i_2})_{w_i}, T(x_{i_3}, x_{j_1}, x_{j_2})_{w_j} \mid i, i_1, i_2, i_3, j, j_1, j_2 \in \mathbb{N} \\ &\qquad \wedge W(x_{m_1}, x_{m_2}) = W(x_{n_1}, x_{n_2}) \wedge w_i, w_j \text{ Zeugenbasis mit } w_m = w_n = \{\{r_m\}, \{r_n\}\}\} \\ &\qquad \cup \{V(x_{k_1}, x_{k_2})_{r_k} \mid k, k_1, k_2 \in \mathbb{N} \wedge r_k \text{ Tupelidentifikator} \wedge t_k = V(x_{k_1}, x_{k_2}) \text{ Dangling Tuple}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{j_1}, x_{j_2}) \mid i_1, i_2, i_3, j_1, j_2 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Wegen der Duplikate und Dangling Tuples enthält die rekonstruierte Instanz I^* weniger Tupel als die Instanz I, d.h. es gilt $I^* \preccurlyeq I$, und wir erhalten eine relaxte Chase-Inverse. Die Verwendung zusätzlicher Zeugenbasen und Side Tables wie $\{V(x_{k_1}, x_{k_2})_{r_k} \mid k, k_1, k_2 \in \mathbb{N} \wedge r_k$ Tupelidentifikator $\wedge\, t_k = V(x_{k_1}, x_{k_2})$ Dangling Tuple$\}$ (blau hervorgehoben) ermöglicht jedoch die Rekonstruktion der Dangling Tuples. Zusätzlich können mit Hilfe der Zeugenbasen die Duplikate rekonstruiert werden und so eine exakte Chase-Inverse garantiert werden.

Seien alternativ $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ definiert über

$$\begin{aligned} \Sigma &= \{R(a, b, c) \wedge V(d, e) \wedge \texttt{cond}_A \rightarrow W(a, b) \wedge T(c, d, e), \\ &\quad\; R(a, b, c) \wedge V(d, e) \wedge \neg\texttt{cond}_A \rightarrow \exists C : W(a, b) \wedge T(C, d, e)\}, \\ \Sigma^{-1} &= \{W(a, b) \wedge T(c, d, e) \rightarrow R(a, b, c),\ T(c, d, e) \rightarrow V(d, e)\}. \end{aligned}$$

Seien zudem alle Bedingungen wie oben gegeben. Dann kann die Anwendung des Chase&Backchase ohne zusätzliche Provenance-Informationen zu Tupeln führen, die ursprünglich nicht in R vorhanden sind, analog zu `MERGE Table`. Grund dafür ist die Definition der inversen Schemaabbildung $\mathcal{M}^*$. Die Einschränkung auf R kann durch Integration der Zeugenbase realisiert werden. Zusätzliche Side Tables sind hier nicht notwendig. Dies garantiert eine exakte Chase-Inverse. Ohne zusätzliche Provenance-Informationen kann lediglich eine ergebnisäquivalente Chase-Inverse angegeben werden, denn für I wie oben liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(W(x_{i_1}, x_{i_2}), T(x_{i_3}, x_{j_1}, x_{j_2}), T(\eta, x_{k_1}, x_{k_2}) \mid \\ &\qquad i_1, i_2, i_3, j_1, k_1, k_2 \in \mathbb{N} \wedge t_k \text{ Dangling Tuple} \wedge W(x_{m_1}, x_{m_2}) = W(x_{n_1}, x_{n_2})\} \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}), R(x_{i_1}, x_{i_2}, \eta), V(x_{j_1}, x_{j_2}), V(x_{k_1}, x_{k_2}) \mid i_1, i_2, i_3, j_1, j_2, k_1, k_2 \in \mathbb{N}\} \\ &\succcurlyeq I. \end{aligned}$$

□

Klasse 3: Duplikate Das Vorhandensein von Duplikaten erlaubt in der Regel nur die Angabe einer relaxten CHASE-Inversen. Zu dieser Klasse gehören die SMOs `DECOMPOSE Table`, `DROP Column`, `MERGE Column`, `MOVE Column` und `SPLIT Column` sowie der Auswertungsoperator für den natürlichen Verbund. Eine Verbesserung des Inversentyps ist nur durch zusätzliche Provenance-Informationen möglich. In einigen Fällen kann sogar eine exakte CHASE-Inverse garantiert werden, indem Dangling Tuples und Duplikate rekonstruiert werden.

Lemma C.14. *Die Projektion hat eine tp-relaxte* CHASE*-Inverse. Liegen Duplikate vor, so ist die* CHASE*-Inverse relaxt ohne zusätzliche Provenance-Informationen und tp-relaxt mit zusätzlichen Provenance-Informationen.*

Beweis. Seien S_t und S_{t+1} zwei Datenbankschemata einer sich ändernden Datenbank und $Q : S \to S_{t+1}$ eine Abbildung, welche die Projektion beschreibt. Seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^{-1} = (S_{t+1}, S_t, \Sigma^{-1})$ mit

$$\Sigma = \{R(a,b,c) \to T(a,b)\} \quad \text{und} \quad \Sigma^{-1} = \{T(a,b) \to \exists B : R(a,B,c)\}$$

zwei Schemaabbildungen, welche die Projektion und ihre zugehörige Inverse formalisieren. O.b.d.A. sei R auf drei und T auf zwei Attribute beschränkt. Sei ferner $I = \{R(x_i, x_i, x_i)_{r_i} \mid i_1, i_2, i_3 \in \mathbb{N} \wedge r_i \text{ Tupelidentifikator}\}$ eine beliebige Quell-Instanz, wobei r_i Tupelidentifikator des i-ten Tupels ist. Seien weiter $R(x_{j_1}, x_{j_2}, x_{j_3})$ und $R(x_{k_1}, x_{k_2}, x_{k_3})$ zwei Tupel, welche nach der Projektion ein Duplikat liefern. Sei $w_j = \{\{r_j\}, \{r_k\}\}$ Zeugenbasis von r_j und $w_k = \{\{r_j\}, \{r_k\}\}$ Zeugenbasis von r_k. Dann liefert die Anwendung des CHASE&BACKCHASE:

$$\begin{aligned} I^* &= \text{CHASE}_{\mathcal{M}^{-1}}(\text{CHASE}_{\mathcal{M}}(I)) \\ &= \text{CHASE}_{\mathcal{M}^{-1}}(\{T(x_{i_1}, x_{i_2})_{w_i} \mid i \in \mathbb{N} \wedge w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\}\}) \\ &= \{R(x_{i_1}, \eta_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge \eta_{j_2} = \eta_{k_2}\} \\ &\preccurlyeq_{(\text{tp})} I. \end{aligned}$$

Aufgrund des Existenzquantors erzeugt die inverse s-t tgd immer einen neuen Nullwert η_{i_2}. Wegen $T(x_{j_1}, x_{j_2}) = T(x_{k_1}, x_{k_2})$ enthält die rekonstruierte Instanz I^* jedoch weniger Tupel als die ursprüngliche Instanz I. Die Inverse $\mathcal{M}^*$ ist relaxt. Zusätzliche Provenance (blau hervorgehoben) ermöglicht die Rekonstruktion der Duplikate und garantiert so eine tp-relaxte CHASE-Inverse. Liegen keine Duplikate vor, kann immer eine exakte CHASE-Inverse garantiert werden. □

Lemma C.15. *`DECOMPOSE Table` hat zwei mögliche Formalisierungen. Die erste garantiert eine ergebnisäquivalente* CHASE*-Inverse ohne und eine exakte* CHASE*-Inverse mit zusätzlichen Provenance-Informationen. Die zweite Formalisierung liefert eine (tp-)relaxte* CHASE*-Inverse.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `DECOMPOSE Table` ein Evolutionsschritt von S_t zu S_{t+1}. `DECOMPOSE Table` kann auf zwei verschiedene Arten formalisiert werden. Sie kann als eine s-t tgd oder als Menge von zwei s-t tgds dargestellt werden.

Wir betrachten zunächst den ersten Fall, in dem wir ein Tupel einer gegebenen Tabelle R in zwei Tupel in T und V zerlegen. Dazu seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{R(a,b,c) \to T(a,b) \wedge V(b,c)\} \quad \text{und} \quad \Sigma^{-1} = \{V(a,b) \wedge T(b,c) \to R(a,b,c)\},$$

welche den Operator `DECOMPOSE Table` und seine zugehörige inverse Abbildung formalisieren. O.B.d.A. ist R auf drei Attribute beschränkt. Sei weiter $I = \{R(x_{i_1}, x_{i_2}, x_{i_3})_{r_i} \mid i_1, i_2, i_3 \in \mathbb{N} \wedge r_i \text{ Tupelidentifikator des } i\text{-ten Tupels in } R\}$ eine beliebige Instanz. Sei w Zeugenbasis zu einem Tupel $t \in T$ bzw. $t \in V$. Wird keine Provenance verwendet, kann r_i unterdrückt werden. Dann liefert die Anwendung des CHASE&BACKCHASE:

$$\begin{aligned} I^* &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)) \\ &= \text{CHASE}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2}), V(x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge \ x_{i_2} \not\to x_{i_3}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &\succcurlyeq I. \end{aligned}$$

Wenn x_{i_2} kein Schlüssel in V ist, erzeugt der natürliche Verbund im zweiten CHASE-Schritt Tupel, die nicht in I enthalten waren. Somit gilt $I^* \succcurlyeq I$ und wir erhalten eine ergebnisäquivalente CHASE-inverse. Ist x_{i_2} hingegen ein Schlüssel, so ergibt diese Formalisierung eine exakte CHASE-Inverse. Das Hinzufügen von Provenance-Informationen (blau hervorgehoben) liefert dann:

$$\begin{aligned} I^* &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)) \\ &= \text{CHASE}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2})_{w_{i_1}}, V(x_{i_2}, x_{i_3})_{w_{i_2}} \mid i, i_1, i_2, i_3 \in \mathbb{N} \\ &\qquad \wedge\ x_{i_2} \text{ kein Schlüssel in } V \wedge\ w_{i_1}, w_{i_2} \text{ Zeugenbasen}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &= I \end{aligned}$$

Durch die zusätzlichen Zeugenbasen können alle Quell-Tupel vollständig rekonstruiert werden. Die zusätzlichen Tupel, die durch die Nicht-Schlüsseleigenschaft von x_{i_2} entstehen, werden durch die Zeugen negiert. Somit gilt $I^* = I$ und die Chase-Inverse kann als ergebnisäquivalent (ohne Provenance) bzw. exakt (mit Provenance) angegeben werden.

Sei alternativ $\mathcal{M}$ als Menge zweier s-t tgds formalisiert. In diesem Fall werden alle Tupel einer gegebenen Tabelle R in zwei neue Tabellen T und V kopiert. Dazu seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen definiert über

$$\begin{aligned} \Sigma &= \{R(a,b,c) \to T(a,b),\ R(a,b,c) \to S(b,c)\} \\ \Sigma^{-1} &= \{T(a,b) \to \exists C : R(a,b,C),\ S(b,c) \to \exists A : R(a,b,c)\}. \end{aligned}$$

Weiter sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3})_{r_i} \mid i, i_1, i_2, i_3 \in \mathbb{N} \wedge r_i$ Tupelidentifikator des i-ten Tupels in $R\}$ eine Instanz und w Zeugenbasis eines Tupels $t \in T$ bzw. $t \in V$. Wird keine Provenance verwendet, kann r_i wie zuvor unterdrückt werden. Seien r_j und r_k die Tupelidentifikatoren von zwei Duplikaten, die durch Anwendung der Evolution auf `DECOMPOSE Table` erzeugt wurden. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2})_{w_{i_1}}, V(x_{i_2}, x_{i_3})_{w_{i_2}} \mid i_1, i_2, i_3 \in \mathbb{N} \\ &\qquad \wedge \exists j,k : T(x_{j_1}, x_{j_2}) = T(x_{k_1}, x_{k_2}) \wedge w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\}\}) \\ &= \{R(x_{i_1}, x_{i_2}, \eta_{i_3}), R(\eta_{i_1}, x_{i_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &\preccurlyeq_{(\text{tp})} I. \end{aligned}$$

Aufgrund von Duplikaten enthält die rekonstruierte Instanz I^* weniger Tupel als die Quell-Instanz I, denn $\eta_{j_3} = \eta_{k_3}$. In diesem Fall erhalten wir eine relaxte Chase-Inverse. Zusätzliche Provenance-Informationen ermöglichen die Rekonstruktion dieser Dangling Tuples und wir erhalten eine tp-relaxed Chase-Inverse. Liegen keine Duplikate vor, kann auch ohne Provenance eine tp-relaxed Chase-Inverse für `DECOMPOSE Table` angegeben werden. □

Lemma C.16. *`DROP Column` hat eine tp-relaxed Chase-Inverse. Wenn Duplikate vorhanden sind, kann dies durch zusätzliche Provenance-Informationen garantiert werden. Ohne Provenance ist die Inverse relaxt.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich ändernden Datenbank. Sei `DROP Column` ein Evolutionsschritt von S_t zu S_{t+1}. Seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{R(a,b,c,d) \to T(a,b,c)\} \quad \text{und} \quad \Sigma^{-1} = \{T(a,b,c) \to \exists D : R(a,b,c,D)\},$$

welche `DROP Column` und seine zugehörige inverse Abbildung formalisieren. O.B.d.A. sei R beschränkt auf vier Attribute, T beschränkt auf drei Attribute und $I = \{R(x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4})_{r_i} \mid i, i_1, i_2, i_3, i_4 \in \mathbb{N} \wedge r_i$ Tupelidentifikator des i-ten Tupels in $R\}$ eine beliebig gewählte Instanz. Ferner sei w Zeugenbasis eines Tupels $t \in T$. Wird Provenance nicht benötigt, kann r_i unterdrückt werden. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2}, i_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}\eta) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &\preccurlyeq_{\text{tp}} I. \end{aligned}$$

Dies garantiert eine tp-relaxte Chase-Inverse. Zusätzliche Provenance-Informationen ändern daran nichts.

Seien nun r_j und r_k die Tupel-Identifikatoren zweier Duplikate, die durch Anwendung der Evolution erzeugt wurden. Dann ergibt Chase&Backchase

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2})_{w_i} \mid i, i_1, i_2 \in \mathbb{N} \wedge\ \exists j,k : T(x_{j_1}, x_{j_2}) = T(x_{k_1}, x_{k_2}) \\ &\qquad \wedge w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\}\}) \\ &= \{T(x_{i_1}, x_{i_2}, \eta) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &\preccurlyeq_{(\text{tp})} I. \end{aligned}$$

Aufgrund der Duplikate enthält die rekonstruierte Instanz I^* weniger Tupel als die ursprüngliche Instanz I, d.h. $I^* \preccurlyeq I$. Alles in allem hat $\mathcal{M}$ eine relaxte Chase-Inverse. Durch zusätzliche Provenance-Informationen (blau hervorgehoben) können diese Dangling Tuples rekonstruiert werden und eine tp-relaxed Chase-Inverse kann garantiert werden. □

Lemma C.17. *`MERGE Column` hat eine tp-relaxed Chase-Inverse. Bei Vorliegen von Duplikaten kann dies nur durch zusätzliche Provenance-Informationen erreicht werden. Außerdem kann, wenn die Inverse der Merge-Funktion*

f bekannt ist, eine exakte CHASE*-Inverse garantiert werden. In diesem Fall sind neben den Zeugenbasen bzw. Provenance-Polynomen noch zusätzliche Side Tables erforderlich.*

Beweis. Sei `MERGE Column` ein Evolutionsschritt von S_t zu S_{t+1}. Seien $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ mit

$$\Sigma = \{R(a,b,c) \to T(a, f(b,c))\} \quad , \quad \Sigma^{-1} = \{T(a,g) \to \exists D, E : R(a,D,E)\}$$

zwei Schemaabbildungen, welche den Operator `MERGE Column` und seine zugehörige inverse Funktion formalisieren. O.b.d.A. sei R auf drei und T auf zwei Attribute beschränkt. Sei ferner f eine beliebige Merging-Funktion und $I = \{R(x_{i_1}, x_{i_2}, x_{i_3})_{r_i} \mid i_1, i_2, i_3 \in \mathbb{N} \wedge f(x_{j_2}, x_{j_3}) = f(x_{k_2}, x_{k_3}) \wedge\ r_i \text{ Tupelidentifikator}\}$ eine beliebige Quell-Instanz. Sei weiter r_i Tupelidentifikator des i-ten Tupels und seien $R(x_{j_1}, x_{j_2}, x_{j_3})$ sowie $R(x_{k_1}, x_{k_2}, x_{k_3})$ zwei Tupel, welche nach Anwendung von f ein Duplikat liefern. Sei $w_j = \{\{r_j\}, \{r_k\}\}$ Zeugenbasis von r_j und $w_k = \{\{r_j\}, \{r_k\}\}$ Zeugenbasis von r_k. Ein Zeuge, hier $\{r_j\}$ und $\{r_k\}$, enthält alle Tupel-IDs, die für die Rekonstruktion eines Tupels benötigt werden. Zwei Zeugen stellen innerhalb einer Zeugenbasis ein Duplikat dar (blau hervorgehoben). Für diese Erweiterungen liefert die Anwendung des CHASE&BACKCHASE:

$$\begin{aligned}
I^* &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)) \\
&= \text{CHASE}_{\mathcal{M}^*}(\{T(x_{i_1}, f(x_{i_2}, x_{i_3}))_{w_i} \mid i \in \mathbb{N} \wedge\ f(x_{j_2}, x_{j_3}) = f(x_{k_2}, x_{k_3}) \\
&\qquad \wedge\ w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\})\} \\
&= \{R(x_{i_1}, \eta_{i_2}, \eta_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge\ \eta_{j_2} = \eta_{k_2}, \eta_{j_3} = \eta_{k_3}\} \\
&\preccurlyeq_{(\text{tp})} I.
\end{aligned}$$

Aufgrund der $\exists$-Quantoren erzeugt die inverse s-t tgd immer zwei neue Nullwerte η_{i_2} und η_{i_3}. Somit können die in f verarbeiteten Attributwerte nicht wiederhergestellt werden. Wegen des Duplikats enthält die rekonstruierte Instanz I^* jedoch weniger Tupel als die ursprüngliche Instanz I. Die Inverse $\mathcal{M}^*$ wird also ohne und tp-relaxed mit vorhandenen Duplikaten relaxt. Das Hinzufügen von Zeugenbasen garantiert eine tp-relaxierte CHASE-Inverse. In diesem Fall gilt sowohl $\eta_{j_2} \neq \eta_{j_3}$ als auch $\eta_{k_2} \neq \eta_{k_3}$ und das verloren gegangene Duplikat kann rekonstruiert werden.

Seien alternativ $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ definiert über

$$\begin{aligned}
\Sigma &= \{R(a,b,c) \to T(a, f(b,c)) \wedge S(c)\} \\
\Sigma^{-1} &= \{T(a,g) \wedge S(c) \to R(a, f^{-1}(g,c), c\}.
\end{aligned}$$

Diese Formalisierung kann nur in Kombination mit der Provenance verwendet werden, da hier die Zeugenbasen w_i, Side Tables S_i und die Kenntnis von f^{-1} notwendig sind (blau hervorgehoben). So liefert $f^{-1}(g,c)$ den rekonstruierten Attributwert zu b und die Anwendung des CHASE&BACKCHASE:

$$\begin{aligned}
I^* &= \text{CHASE}_{\mathcal{M}^*}(\text{CHASE}_{\mathcal{M}}(I)) \\
&= \text{CHASE}_{\mathcal{M}^*}(\{T(x_{i_1}, f(x_{i_2}, x_{i_3}))_{w_i} \mid i, i_1, i_2, i_3 \in \mathbb{N} \wedge\ \exists j,k : f(x_{j_2}, x_{j_3}) = f(x_{k_2}, x_{k_3}) \\
&\qquad \wedge\ w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\} \wedge\ f \text{ invertierbare Funktion}\} \\
&\qquad \cup\ \{(x_{i_3})_{r_i} \mid i, i_3 \in \mathbb{N} \wedge\ r_i \text{ Tupelidentifikator}\}) \\
&= \{R(x_{i_1}, f^{-1}(f(x_{i_2}, x_{i_3}), x_{i_3}), x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N} \wedge\ f(x_{j_2}, x_{j_3}) = f(x_{k_2}, x_{k_3})\} \\
&= \{R(x_{i_1}, x_{x_2}, x_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\
&= I.
\end{aligned}$$

Sei w_i Zeugenbasis und x_{i_3} Element der Side Table $\{(x_{i_3})_{r_i} \mid i, i_3 \in \mathbb{N} \wedge\ r_i \text{ Tupelidentifikator}\}$. Dann kann x_{i_2} aus $f^{-1}(g_i, x_{i_3})$ wie folgt berechnet werden:

- die Attributwerte g_i und x_{i_1} sind in $J = \text{CHASE}_{\mathcal{M}}(I)$ explizit enthalten;
- die Umkehrfunktion f^{-1} ist per Definition gegeben;
- der Attributwert x_{i_3} kann mit Hilfe von w_i aus der Side Table ausgelesen werden.

Auch Duplikate können mit Hilfe dieser zusätzlichen Informationen rekonstruiert werden. Insgesamt kann eine exakte CHASE-Inverse garantiert werden. □

Lemma C.18. *`SPLIT Column` hat eine tp-relaxed* CHASE*-Inverse. Beim Vorliegen von Duplikaten kann dies nur durch zusätzliche Provenance-Informationen erreicht werden. Wenn die Inverse der Split-Funktion f bekannt ist, kann sogar eine exakte* CHASE*-Inverse garantiert werden.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `SPLIT Column` ein Evolutionsschritt von S_t zu S_{t+1}. Seien $\mathcal{M} = (S_t, S_{i+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{R(a,b) \to T(f_1(a), b, f_2(a))\} \quad \text{und} \quad \Sigma = \{T(g_1, b, g_2) \to \exists C : R(C,b)\},$$

welche `SPLIT Column` und ihre zugehörige inverse Abbildung formalisiert. O.B.d.A. sei R auf zwei Attribute und T auf drei Attribute beschränkt. Ferner seien f_1 und f_2 zwei beliebige Split-Funktionen und $I = \{R(x_{i_1}, x_{i_2})_{r_i} \mid i, i_1, i_2 \in \mathbb{N} \wedge r_i$ Tupelidentifikator des i-ten Tupels in $R\}$ eine beliebige Instanz. Dann ist w Zeugenbasis eines Tupels $t \in T$. Wird keine Provenance verwendet, kann r_i ignoriert werden. Die Anwendung des Chase&Backchase liefert dann:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{T(f_1(x_{i_1}), x_{i_2}, f_2(x_{i_1})) \mid i_1, i_2 \in \mathbb{N}\}) \\ &= \{R(\eta, x_{i_2}) \mid i_2 \in \mathbb{N}\} \\ &\preccurlyeq_{\text{tp}} I. \end{aligned}$$

Die inverse Schemaabbildung $T(g_1, b, g_2) \rightarrow \exists C : R(C, b)$ erzeugt immer einen neuen Nullwert. Somit kann das in f_1 und f_2 verarbeitete Attribut nicht wiederhergestellt werden. Die Schemaabbildung $\mathcal{M}$ ist also tp-relaxt. Auch die zusätzliche Zeugenbasis kann den eingeführten Nullwert nicht weiter spezifizieren.

Im Falle von Duplikaten ist $\mathcal{M}^*$ nur relaxt. Dazu seien r_j und r_k Tupel-Identifikatoren zweier Duplikate, die durch die Anwendung der Evolution entstanden sind. Die Anwendung des Chase&Backchase liefert dann:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{T(f_1(x_{i_1}), x_{i_2}, f_2(x_{i_2}))_{w_i} \mid i, i_1, i_2 \in \mathbb{N} \ \wedge \ \exists j, k : f_1(x_{j_1}) = f_1(x_{k_1}) \\ &\qquad \wedge \ f_2(x_{j_1}) = f_2(x_{k_1}) \wedge w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\}\}) \\ &= \{R(\eta, x_{i_2}) \mid i_2 \in \mathbb{N}\} \\ &\preccurlyeq_{(\text{tp})} I. \end{aligned}$$

Aufgrund der Duplikate enthält die rekonstruierte Instanz I^* weniger Tupel als die Quell-Instanz I, d.h. $I^* = I$. Sei w_j die Zeugenbasis des Duplikats r_j. Dann ist $w_j = w_k = \{\{r_j\}, \{r_k\}\}$, d.h. w_j enthält zwei Tupelidentifikatoren. Daraus können wir zwei Tupel rekonstruieren. Insgesamt erhalten wir somit eine tp-relaxte Chase-Inverse.

Sei alternativ $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ definiert über

$$\Sigma^{-1} = \{T(g_1, b, g_2) \rightarrow R(f^{-1}(g_1, g_2), b)\}.$$

Diese Formalisierung kann nur in Kombination mit Provenance verwendet werden, da hier die Zeugenbasen sowie die Kenntnis von f_1^{-1} und f_2^{-1} notwendig sind (blau hervorgehoben). Dann liefert Chase&Backchase:

$$\begin{aligned} I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\ &= \textsc{Chase}_{\mathcal{M}^*}(\{T(f_1(x_{i_1}), x_{i_2}, f_2(x_{i_1}))_{w_i} \mid i, i_1, i_2 \in \mathbb{N} \wedge \ \exists j, k : f_1(x_{j_1}) = f_1(x_{k_1}) \wedge \ f_2(x_{j_1}) = f_2(x_{k_1}) \\ &\qquad \wedge \ f_1, f_2 \text{ invertierbare Funktion } \wedge \ w_i \text{ Zeugenbasis mit } w_j = w_k = \{\{r_j\}, \{r_k\}\}\}) \\ &= \{R(f^{-1}(f_1(x_{i_1}), f_2(x_{i_1})), x_{i_2}) \mid i_1, i_2 \in \mathbb{N}\} \\ &= \{R(x_{i_1}, x_{i_2} \mid i_1, i_2 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

In diesem Fall können auch Duplikate rekonstruiert werden. Insgesamt ergibt sich eine exakte Chase-Inverse für $\mathcal{M}$ in beiden Fällen mit und ohne Duplikate in J. □

Klasse 4: Wiederherstellung durch Provenance Fast alle Operatoren können den Klassen 1 bis 3 zugeordnet werden, nur die Vereinigung zweier Relationen, `MERGE Table` sowie `DROP Table` bleiben übrig. Sie enthalten zwar weder Duplikate noch Dangling-Tupel, dennoch beeinflussen zusätzlichen Provenance-Informationen den Chase-Inversentyp.

Lemma C.19. *Die Vereinigung zweier Relationen sowie die SMO `MERGE Table` haben eine ergebnisäquivalente* Chase*-Inverse. Zusätzliche Provenance-Informationen ermöglichen alternativ die Angabe einer exakten Inversen.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich ändernden Datenbank und `MERGE Table` ein Evolutionsschritt von S_t zu S_{t+1}. Seien alternativ S_t und S_{t+1} zwei Datenbankschemata einer sich ändernden Datenbank und $Q : S_t \to S_{t+1}$ die Anfrage `SELECT * FROM` R `UNION SELECT * FROM` V. Seien weiter $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\begin{aligned} \Sigma &= \{R(a,b,c) \to T(a,b,c),\ V(a,b,c) \to T(a,b,c)\}, \\ \Sigma^{-1} &= \{T(a,b,c) \to R(a,b,c),\ T(a,b,c) \to V(a,b,c)\}, \end{aligned}$$

welche die Vereinigung zweier Relationen bzw. `MERGE Table` und ihre inverse Abbildung formalisieren. O.b.d.A. seien R, V und T jeweils auf drei Attribute beschränkt. Ferner sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3})_{r_i}, V(x_{j_1}, x_{j_2}, x_{j_3})_{v_j} \mid i, i_1, i_2, i_3, j, j_1, j_2, j_3 \in \mathbb{N} \wedge r_i, v_j$ Tupelidentifikator des i-ten oder j-ten Tupels in R bzw. $V\}$ eine gegebene Instanz. Wird Provenance nicht verwendet, können r_i, v_i ignoriert werden. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2}, x_{i_3}), T(x_{j_1}, x_{j_2}, x_{j_3}) \mid i_1, i_2, i_3, j_1, j_2, j_3 \in \mathbb{N}\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3}), R(x_{j_1}, x_{j_2}, x_{j_3}), V(x_{i_1}, x_{i_2}, x_{i_3}), V(x_{j_1}, x_{j_2}, x_{j_3}) \mid i_1, i_2, i_3, j_1, j_2, j_3 \in \mathbb{N}\} \\ &\supseteq I \end{aligned}$$

Die Ergebnisinstanz I^* kann mehr Tupel enthalten als die Quell-Instanz I. Dies liefert eine ergebnisäquivalente Chase-Inverse.

Durch Hinzufügen von zusätzlichen Provenance-Informationen (blau hervorgehoben) können die Ergebnistupel auf ihre Quell-Relationen R und V beschränkt werden. Sei hierfür w die Zeugenbasis eines Tupels $t \in T$. Dann zeigt $w = \{\{r_i\}\}$ an, dass t ursprünglich aus R stammt. Alternativ besagt $w = \{\{r_i, v_j\}\}$, dass t sowohl in R als auch in T enthalten ist. Insgesamt erhalten wir eine exakte Chase-Inverse:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2}, x_{i_3})_{w_i}, T(x_{j_1}, x_{j_2}, x_{j_3})_{w_j} \mid i, i_1, i_2, i_3, j, j_1, j_2, j_3 \in \mathbb{N} \\ &\qquad \wedge\ w_i = \{\{r_i\}\} \text{ Zeugenbasis von } t_i \in T \wedge\ w_j = \{\{v_j\}\} \text{ Zeugenbasis von } t_j \in T\}) \\ &= \{R(x_{i_1}, x_{i_2}, x_{i_3})_{|w_i}, V(x_{j_1}, x_{j_2}, x_{j_3})_{|w_j} \mid i, i_1, i_2, i_3, j, j_1, j_2, j_3 \in \mathbb{N}\} \\ &= I. \end{aligned}$$

Die Chase-Inverse für die Vereinigung zweier Relationen und `MERGE Table` kann somit von einer ergebnisäquivalenten zu einer exakten Chase-Inversen unter Verwendung der Provenance verbessert werden. □

Lemma C.20. *`DROP Table` hat eine relaxte* Chase*-Inverse. Zusätzliche Provenance-Informationen verbessern den* Chase*-Inversentyp zu tp-relaxed.*

Beweis. Sei $S_1, ..., S_n$ eine Folge von Schemata einer sich verändernden Datenbank. Sei `DROP Table` ein Evolutionsschritt von S_t zu S_{t+1}. Sei $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{R(a,b,c) \to \emptyset\} \quad \text{und} \quad \Sigma^{-1} = \{\emptyset \to \exists A, B, C : R(A,B,C)\},$$

welche `DROP Table` und die damit verbundene inverse Abbildung formalisieren. O.B.d.A. sei R auf drei Attribute beschränkt. Weiter sei $I = \{R(x_{i_1}, x_{i_2}, x_{i_3})_{r_i} \mid i, i_1, i_2, i_3 \in \mathbb{N} \wedge r_i$ Tupelidentifikator des i-ten Tupels$\}$ eine gegebene Instanz. Wird die zusätzliche Provenance nicht benötigt, kann r_i unterdrückt werden. Die Anwendung des Chase&Backchase ergibt:

$$\begin{aligned} I^* &= \text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) \\ &= \text{Chase}_{\mathcal{M}^*}(\emptyset_p) \text{ mit } p = \{r_i | r_i \text{ Tupelidentifikator des } i\text{-ten Tupels}\} \\ &= \{(\eta_{i_1}, \eta_{i_2}, \eta_{i_3}) \mid i_1, i_2, i_3 \in \mathbb{N}\} \\ &\preccurlyeq_{(\text{tp})} I. \end{aligned}$$

Die (minimale) Teil-Datenbank $I^* = (\eta_{i_1}, \eta_{i_2}, \eta_{i_3})$ kann homomorph auf I abgebildet werden. Es existiert also eine relaxte Chase-Inverse. Zusätzliche Provenance-Informationen, hier ***where***-Provenance, verbessern den Typ der Chase-Inversen auf tp-relaxed. Wenn wir die Liste der Quell-IDs p analysieren, können wir die korrekte Anzahl des Null-Tupels rekonstruieren, d.h. $\mid I^* \mid = \mid I \mid$. □

Lemma C.21. *Die Differenz hat eine relaxte* Chase*-Inverse. Die Verwendung zusätzlicher Provenance-Informationen wie Zeugenbasen und Side Tables ermöglicht die Angabe einer exakten* Chase*-Inversen.*

Beweis. Seien S_t und S_{t+1} zwei Datenbankschemata und $Q : S_t \to S_{t+1}$ die Anfrage `SELECT` $*$ `FROM` R `EXCEPT` `SELECT` $*$ `FROM` V. Seien weiter $\mathcal{M} = (S_t, S_{t+1}, \Sigma)$ und $\mathcal{M}^* = (S_{t+1}, S_t, \Sigma^{-1})$ zwei Schemaabbildungen mit

$$\Sigma = \{R(a,b) \wedge \neg V(a,b) \to T(a,b) \wedge H(a,b)\} \quad \text{und} \quad \Sigma^{-1} = \{T(a,b) \to R(a,b), H(a,b) \to R(a,b) \wedge V(a,b)\},$$

welche die Differenz zweier Relationen und ihre inverse Abbildung formalisieren. O.B.d.A. seien R, V und T jeweils auf zwei Attribute beschränkt. Sei $I = \{R(x_{i_1}, x_{i_2}, R(x_{i_3}, x_{i_4}), V(x_{i_1}, x_{i_2}), V(x_{i_5}, x_{i_6}) \mid i_1, i_2, i_3, i_4, i_5, i_6 \in \mathbb{N} \wedge r_i, v_i$ Tupelidentifikatoren des i-ten Tupels in R bzw. $V\}$ eine gegebene Instanz. Dann ist w Zeugenbasis eines Tupels $t \in T$. Wird die zusätzliche Provenance nicht benötigt, können r_i, v_i unterdrückt werden. Dann liefert die Anwendung des Chase&Backchase:

$$\begin{aligned}
I^* &= \textsc{Chase}_{\mathcal{M}^*}(\textsc{Chase}_{\mathcal{M}}(I)) \\
&= \textsc{Chase}_{\mathcal{M}^*}(\{T(x_{i_1}, x_{i_2}) \mid i_1, i_2 \in \mathbb{N}\} \\
&\qquad \cup \{H(x_{i_3}, x_{i_4})_{r_i}, H(x_{i_5}, x_{i_6})_{v_i} \mid i_3, i_4, i_5, i_6 \in \mathbb{N} \wedge r_i, v_i \text{ Tupelidentifikatoren}\}) \\
&= \{R(x_{i_1}, x_{i_2}) \mid i_1, i_2 \in \mathbb{N}\} \\
&\preccurlyeq I.
\end{aligned}$$

Einige Tupel wie $R(x_{i_1}, x_{i_2})$ und $V(x_{i_5}, x_{i_6})$ gehen bei der Differenzbildung verloren. Dies hat zur Folge, dass die Anzahl der Tupel in I^* kleiner ist als die Anzahl der Tupel in I. Und wir erhalten eine relaxte Chase-Inverse. Mit Hilfe von Provenance und Side Tables (blau hervorgehoben) können diese „ungenutzten“ Tupel gespeichert und später in R bzw. V übertragen werden. So können wir für die Differenzbildung eine exakte Chase-Inverse garantieren. □

D. Ein- und Ausgabe-Dateien für ChaTEAU und ProSA

In diesem Abschnitt führen wir einige Ein- und Ausgabe-Dateien für ChaTEAU und ProSA auf. Alle angegebenen Dateien sind zudem in den zugehörigen Git-Repositories zu finden. Hierzu gehören:

ChaTEAU-Dateien:

- Ein- und Ausgabe-Dateien für ChaTEAU
- Benchmark-Dateien als Eingabe für ChaTEAU

ProSA-Dateien:

- Evolution als Eingabe für ProSA
- Domänengeneralisierungshierarchie als Eingabe für ProSA
- provenance-erweiterter Parser-Output
- provenance-erweiterter Chase-Output
- (minimale) Teil-Datenbank als Ergebnis von ProSA

Repositories:

- `https://git.informatik.uni-rostock.de/ta093/chateau-demo`
- `https://git.informatik.uni-rostock.de/ta093/prosa-demo`

D.1. ChaTEAU-Dateien

```
<input origin="prosa">
    <schema>
        <relations>
            <relation name="student" tag="S">
                <attribute name="studentid" type="int"/>
                <attribute name="lastname" type="string"/>
                <attribute name="firstname" type="string"/>
                <attribute name="studies" type="string"/>
            </relation>
            <relation name="participant" tag="S">
                <attribute name="courseid" type="int"/>
                <attribute name="studentid" type="int"/>
                <attribute name="semester" type="int"/>
            </relation>
            <relation name="grade" tag="T">
                <attribute name="courseid" type="int"/>
                <attribute name="studentid" type="int"/>
                <attribute name="semester" type="int"/>
                <attribute name="grade" type="int"/>
            </relation>
        </relations>

        <dependencies>
            <sttgd>
                <body>
                    <atom name="student">
                        <variable name="studentid" type="V" index="1" />
                        <variable name="lastname" type="V" index="1" />
                        <variable name="firstname" type="V" index="1" />
                        <variable name="studies" type="V" index="1" />
                    </atom>
                    <atom name="participant">
                        <variable name="courseid" type="V" index="1" />
                        <variable name="studentid" type="V" index="1" />
                        <variable name="semester" type="V" index="1" />
                    </atom>
                </body>
                <head>
                    <atom name="grade">
                        <variable name="courseid" type="V" index="1" />
                        <variable name="studentid" type="V" index="1" />
                        <variable name="semester" type="E" index="1" />
                        <variable name="grade" type="E" index="1" />
                    </atom>
                </head>
            </sttgd>
        </dependencies>
    </schema>

    <instance>
        <atom name="student" id="S1">
            <constant name="studentid" value="3" />
            <constant name="lastname" value="Miller" />
            <constant name="firstname" value="Max" />
            <constant name="studies" value="Electrical engineering" />
        </atom>
        <atom name="student" id="S2">
            <null name="studentid" index="1" />
            <constant name="lastname" value="Miller" />
            <constant name="firstname" value="Max" />
            <constant name="studies" value="Electrical engineering" />
        </atom>

        <atom name="participant" id="P1">
            <constant name="courseid" value="2" />
            <constant name="studentid" value="3" />
            <null name="semester" index="2" />
        </atom>
        <atom name="participant" id="P2">
            <constant name="courseid" value="7" />
            <constant name="studentid" value="3" />
            <null name="semester" index="1" />
        </atom>
    </instance>
</input>
```

XML-Datei D.1 XML-Datei für die Eingabe in ChaTEAU (am Studierenden-Beispiel)

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
  <schema>
    <relations>
      <relation name="grade">
        <attribute name="courseid" type="int" />
        <attribute name="studentid" type="int" />
        <attribute name="semester" type="int" />
        <attribute name="grade" type="int" />
      </relation>
    </relations>

    <dependencies />
  </schema>

  <instance>
    <atom name="grade">
      <constant name="courseid" value="2" />
      <constant name="studentid" value="3" />
      <null name="semester" index="3" />
      <null name="grade" index="1" />
    </atom>
    <atom name="grade">
      <constant name="courseid" value="7" />
      <constant name="studentid" value="3" />
      <null name="semester" index="4" />
      <null name="grade" index="2" />
    </atom>
  </instance>
</input>
```

XML-Datei D.2 XML-Datei als Ausgabe aus ChaTEAU (am Studierenden-Beispiel)

ChaTEAU-Eingabedateien Auch in ChaTEAU testen wir die sieben Benchmark-Anfragen QA_1 bis QB_4. Die Eingabe-Dateien für die ersten drei sind hier zu finden (siehe XML-Dateien D.3 bis D.5). Die Anfragen QB_1 bis QB_4 hingegen können wegen des fehlenden BACKCHASE nicht getestet werden.

```
<input>
  <schema>
    <relations>
      <relation name="R1">
        <attribute name="a1" type="string" />
        <attribute name="a2" type="string" />
      </relation>
      <relation name="R2">
        <attribute name="a2" type="string" />
        <attribute name="a3" type="string" />
      </relation>
    </relations>
    <dependencies>
      <tgd>
        <body>
          <atom name="R1">
            <variable name="a1" type="V" index="1" />
            <variable name="a2" type="V" index="1" />
          </atom>
        </body>
        <head>
          <atom name="R2">
            <variable name="a2" type="V" index="1" />
            <variable name="a3" type="E" index="1" />
          </atom>
        </head>
      </tgd>
    </dependencies>
  </schema>

  <query>
    <body>
      <atom name="R1">
        <variable name="a1" type="V" index="1" />
        <variable name="a2" type="E" index="1" />
      </atom>
      <atom name="R2">
        <variable name="a2" type="E" index="1" />
        <variable name="a3" type="E" index="1" />
      </atom>
    </body>
    <head>
      <atom>
        <variable name="a1" type="V" index="1" />
      </atom>
    </head>
  </query>
</input>
```

XML-Datei D.3 Eingabe-Datei für ChaTEAU (Anfrage QA1)

```
<input>
  <schema>
    <relations>
      <relation name="R">
        <attribute name="a1" type="string" />
        <attribute name="a2" type="string" />
        <attribute name="a3" type="string" />
      </relation>
    </relations>

    <dependencies>
      <egd>
        <body>
          <atom name="R">
            <variable name="a1" type="V" index="1" />
            <variable name="a2" type="V" index="1" />
            <variable name="a3" type="V" index="1" />
          </atom>
          <atom name="R">
            <variable name="a1" type="V" index="2" />
            <variable name="a2" type="V" index="1" />
            <variable name="a3" type="V" index="2" />
          </atom>
        </body>
        <head>
          <atom>
            <variable name="a3" type="V" index="1" />
            <variable name="a3" type="V" index="2" />
          </atom>
        </head>
      </egd>
    </dependencies>
  </schema>

  <query>
    <body>
      <atom name="R">
        <variable name="a1" type="V" index="1" />
        <variable name="a2" type="V" index="1" />
        <variable name="a3" type="E" index="1" />
      </atom>
      <atom name="R">
        <variable name="a1" type="E" index="1" />
        <variable name="a2" type="V" index="1" />
        <variable name="a3" type="V" index="1" />
      </atom>
    </body>
    <head>
      <atom>
        <variable name="a1" type="V" index="1" />
        <variable name="a2" type="V" index="1" />
        <variable name="a3" type="V" index="1" />
      </atom>
    </head>
  </query>
</input>
```

XML-Datei D.4 Eingabe-Datei für ChaTEAU (Anfrage QA2)

```
<input>
  <schema>
    <relations>
      <relation name="R">
        <attribute name="a1" type="string" />
        <attribute name="a2" type="string" />
        <attribute name="a3" type="string" />
      </relation>
    </relations>

    <dependencies>
      <tgd>
        <body>
          <atom name="R">
            <variable name="a1" type="V" index="1" />
            <variable name="a2" type="V" index="1" />
            <variable name="a3" type="V" index="1" />
          </atom>
          <atom name="R">
            <variable name="a1" type="V" index="2" />
            <variable name="a2" type="V" index="1" />
            <variable name="a3" type="V" index="2" />
          </atom>
        </body>
        <head>
          <atom name="R">
            <variable name="a1" type="V" index="1" />
            <variable name="a2" type="V" index="1" />
            <variable name="a3" type="V" index="2" />
          </atom>
        </head>
      </tgd>
    </dependencies>
  </schema>

  <query>
    <body>
      <atom name="R">
        <variable name="a1" type="V" index="1" />
        <variable name="a2" type="V" index="1" />
        <variable name="a3" type="E" index="1" />
      </atom>
      <atom name="R">
        <variable name="a1" type="E" index="1" />
        <variable name="a2" type="V" index="1" />
        <variable name="a3" type="V" index="1" />
      </atom>
    </body>
    <head>
      <atom>
        <variable name="a1" type="V" index="1" />
        <variable name="a2" type="V" index="1" />
        <variable name="a3" type="V" index="1" />
      </atom>
    </head>
  </query>
</input>
```

XML-Datei D.5 Eingabe-Datei für ChaTEAU (Anfrage QA3)

D.2. ProSA-Dateien

```
<input>
  <schema>
        <relations>
          <relation name="lecturer" tag="T">
                <attribute name="courseid" type="int" />
                <attribute name="lecturer" type="string" />
          </relation>
          <relation name="course" tag="T">
                <attribute name="courseid" type="int" />
                <attribute name="title" type="string" />
          </relation>
          <relation name="lecturerWithTitle" tag="S">
                <attribute name="courseid" type="int" />
                <attribute name="title" type="string" />
                <attribute name="lecturer" type="string" />
          </relation>
          <relation name="courseWithoutTitle" tag="S">
                <attribute name="courseid" type="int" />
          </relation>
        </relations>

      <dependencies>
                <sttgd>
                  <body>
                        <atom name="lecturerWithTitle">
                          <variable name="courseid" type="V" index="1" />
                          <variable name="title" type="V" index="1" />
                          <variable name="lecturer" type="V" index="1" />
                        </atom>
                  </body>
                  <head>
                  <atom name="course">
                        <variable name="courseid" type="V" index="1" />
                        <variable name="title" type="V" index="1" />
                  </atom>
                  <atom name="lecturer">
                        <variable name="courseid" type="V" index="1" />
                        <variable name="lecturer" type="V" index="1" />
                  </atom>
                </head>
          </sttgd>
          <sttgd>
                <body>
                  <atom name="courseWithoutTitle">
                        <variable name="courseid" type="V" index="1" />
                  </atom>
                </body>
                <head>
                  <atom name="course">
                        <variable name="courseid" type="V" index="1" />
                        <variable name="title" type="E" index="1" />
                  </atom>
                </head>
          </sttgd>
        </dependencies>
  </schema>

  <instance />
</input>
```

XML-Datei D.6 Evolution als Eingabe für ProSA (am Studierenden-Beispiel)

```
<input origin="prosa">
  <schema>
    <relations>
      <relation name="student" tag="S">
        <attribute name="studentid" type="int" />
        <attribute name="lastname" type="string" />
        <attribute name="firstname" type="string" />
        <attribute name="studies" type="string" />
        <attribute name="student_id" type="string" />
      </relation>
      <relation name="Result" tag="T">
        <attribute name="studentid" type="int" />
        <attribute name="lastname" type="string" />
        <attribute name="firstname" type="string" />
        <attribute name="studies" type="string" />
        <attribute name="Result_id" type="string" />
        <attribute name="student_id" type="string" />
      </relation>
    </relations>

    <dependencies>
      <sttgd>
        <body>
          <atom name="student">
            <variable name="studentid" type="V" index="1" />
            <variable name="lastname" type="V" index="1" />
            <variable name="firstname" type="V" index="1" />
            <variable name="studies" type="V" index="1" />
            <variable name="student_id" type="V" index="1" />
          </atom>
        </body>
        <head>
          <atom name="Result">
            <variable name="studentid" type="V" index="1" />
            <variable name="lastname" type="V" index="1" />
            <variable name="firstname" type="V" index="1" />
            <variable name="studies" type="V" index="1" />
            <variable name="student_id" type="V" index="1" />
          </atom>
        </head>
      </sttgd>
    </dependencies>
  </schema>

  <instance>
    <atom name="student" id="student_1">
      <constant name="studentid" value="1" />
      <constant name="lastname" value="Moore" />
      <constant name="firstname" value="Donald" />
      <constant name="studies" value="Teaching" />
      <constant name="student_id" value="student_1" />
    </atom>
      ...
    <atom name="student" id="student_8">
      <constant name="studentid" value="8" />
      <constant name="lastname" value="John" />
      <constant name="firstname" value="Jennifer" />
      <constant name="studies" value="Theory" />
      <constant name="student_id" value="student_8" />
    </atom>
  </instance>
</input>
```

XML-Datei D.7 provenance-erweiterter Parser-Output (am Studierenden-Beispiel)

```
semester: SUBSTRING {0 to 2};

grade: map {1.0 to A AND 1.3 to A AND 1.7 to B AND 2.0 to B AND 2.3 to B AND 2.7 to C
       AND 3.0 to C AND 3.3 to C AND 3.7 to D AND 4.0 to D AND 5.0 to E};

studies: map {Teaching to Education AND Mathematics to Science AND Computer Science to Science};
```

TXT-Datei D.8 Domänengeneralisierungshierarchie als Eingabe für ProSA (am Studierenden-Beispiel)

```
<?xml version="1.0" encoding="UTF-8"?>
<input origin="prosa">
  <schema>
    <relations>
      <relation name="student" tag="S">
        <attribute name="studentid" type="int" />
        <attribute name="lastname" type="string" />
        <attribute name="firstname" type="string" />
        <attribute name="studies" type="string" />
        <attribute name="student_id" type="string" />
      </relation>
      <relation name="Result" tag="T">
        <attribute name="studentid" type="int" />
        <attribute name="lastname" type="string" />
        <attribute name="firstname" type="string" />
        <attribute name="studies" type="string" />
        <attribute name="Result_id" type="string" />
        <attribute name="student_id" type="string" />
      </relation>
    </relations>
    <dependencies />
  </schema>

  <instance>
    <atom name="Result" id="Result_1">
      <constant name="studentid" value="1" />
      <constant name="lastname" value="Moore" />
      <constant name="firstname" value="Donald" />
      <constant name="studies" value="Teaching" />
      <constant name="student_id" value="student_1" />
    </atom>
    ...
    <atom name="Result" id="Result_8">
      <constant name="studentid" value="8" />
      <constant name="lastname" value="John" />
      <constant name="firstname" value="Jennifer" />
      <constant name="studies" value="Theory" />
      <constant name="student_id" value="student_8" />
    </atom>
  </instance>

  <provenance>
    <atom name="Result" id="Result_1">
      <where>
        <tuples>
          <tuple id="student_1" />
        </tuples>
        <relations>
          <relation name="student" />
        </relations>
      </where>
      <why>
        <witnesses>
          <witness id="student_1" />
        </witnesses>
      </why>
      <how polynom="(student_1)" />
    </atom>
    ...
    <atom name="Result" id="Result_8">
      <where>
        <tuples>
          <tuple id="student_8" />
        </tuples>
        <relations>
          <relation name="student" />
        </relations>
      </where>
      <why>
        <witnesses>
          <witness id="student_8" />
        </witnesses>
      </why>
      <how polynom="(student_8)" />
    </atom>
  </provenance>
</input>
```

XML-Datei D.9 provenance-erweiterter Chase-Output (am Studierenden-Beispiel)

```
<?xml version="1.0" encoding="UTF-8"?>
<input origin="prosa">
  <schema>
    <relations>
      <relation name="student" tag="T">
        <attribute name="studentid" type="int" />
        <attribute name="lastname" type="string" />
        <attribute name="firstname" type="string" />
        <attribute name="studies" type="string" />
        <attribute name="student_id" type="string" />
      </relation>
      <relation name="Result" tag="S">
        <attribute name="studentid" type="int" />
        <attribute name="lastname" type="string" />
        <attribute name="firstname" type="string" />
        <attribute name="studies" type="string" />
        <attribute name="Result_id" type="string" />
        <attribute name="student_id" type="string" />
      </relation>
    </relations>
    <dependencies />
  </schema>

  <instance>
    <atom name="student">
      <constant name="studentid" value="1" />
      <constant name="lastname" value="Moore" />
      <constant name="firstname" value="Donald" />
      <constant name="studies" value="Teaching" />
      <constant name="student_id" value="student_1" />
    </atom>
    <atom name="student">
      <constant name="studentid" value="2" />
      <constant name="lastname" value="Morgan" />
      <constant name="firstname" value="Sarah" />
      <constant name="studies" value="Mathematics" />
      <constant name="student_id" value="student_2" />
    </atom>

    ...

    <atom name="student">
      <constant name="studentid" value="8" />
      <constant name="lastname" value="John" />
      <constant name="firstname" value="Jennifer" />
      <constant name="studies" value="Theory" />
      <constant name="student_id" value="student_8" />
    </atom>
  </instance>
</input>
```

XML-Datei D.10 (minimale) Teil-Datenbank als Ergebnis von ProSA (am Studierenden-Beispiel)

E. Fragenkatalog der Interview-Studie

Die Fragen unserer Interview-Studie unterteilen sich in drei Bereiche: (1) persönliche Situation, (2) Provenance und Privacy, sowie (3) Forschungsdatenmanagement. Die vollständigen Fragen sowie eine Auswahl an Antworten können in [Sch20] nachgelesen werden.

Persönliche Situation

(A) *Woran forschen oder arbeiten Sie? Welchen Status haben Sie an der Universität, oder üben Sie einen Beruf außerhalb der Universität aus?*
Diese Fragen ermöglichen es uns, die Antworten einer bestimmten Forschungsdisziplin zuzuordnen. Zudem ist der Umgang mit Daten abhängig vom Umfeld sowie Erfahrung der unserer Teilnehmer.

(B) *Welche Arten und Mengen von Daten fallen in bei Ihnen/bei Ihrer Forschung an?*
Daten gibt es in vielen verschiedenen Formen und Größen, was sich ebenfalls auf den Umgang mit ihnen auswirkt.

(C) *Wie lange werden die Daten gespeichert? Und wer kümmert sich um die Speicherung?*
Forschungsdaten werden in der Regel über Jahre hinweg aufbewahrt. Dies gilt sowohl für Primär- als auch für Sekundärdaten, d.h. für die Rohdaten sowie ihre Verarbeitung. Die Verwaltung der Daten erfordert dabei einen gewissen Aufwand, welcher je nach Projekt von den Forschern selbst übernommen oder aber ausgelagert wird.

Provenance und Privacy

(D) *Was wissen Sie über Provenance? Und was verstehen Sie unter dem Begriff Privacy?*
In erster Linie interessiert uns das Wissen unserer Teilnehmer zu den beiden Themen Provenance und Privacy. Die Fragen sind daher absichtlich vage gehalten. Weder werden die „Datenherkunft“ noch der Datenschutz im Besonderen erwähnt.

(E) *Können Sie sich Situationen vorstellen, in denen Privacy Daten meint, welche nicht personengebunden sind? Falls ja, welche?*
In ProSA betrachten wir nicht nur personenbezogene Daten, sondern Forschungsdaten im Allgemeinen als schützenswerte Daten [ASH21a]. Damit fallen auch sie für uns unter den Begriff Privacy.

(F) *Beispiel: Kann der in Tabelle E.1 gegebene Datensatz so veröffentlichen werde, wie er ist? Falls nicht, wie muss der Datensatz angepasst werden?*
Um zu ergründen, wie gut das Verständnis unserer Teilnehmer von allgemeinen Datenschutz-Problemen ist, präsentieren wir ihnen eine Auszug eines fiktiven medizinischen Beispieldatensatzes (siehe Tabelle E.1). Wie in [Sch20] beschrieben ist „der Datensatz ist aus Sicht des Datenschutzes höchst problematisch: Zwar fehlen Vor- und Nachnamen, aber es besteht keine k-Anonymität und somit auch keine l-Diversität. Zudem war es um die Jahrtausendwende herum möglich, 87% aller in den USA lebenden Personen nur anhand der Kombination aus Postleitzahl (ZIP-Code), Geburtsdatum und Geschlecht eindeutig zu identifizieren, wie Latanya Sweeney 2000 herausfand [Swe00]. Noch dazu ist eine medizinische Diagnose ein höchst privates und daher sehr schützenswertes Attribut. Diese Frage ist bewusst als Fangfrage formuliert, um die Antwort nicht versehentlich in eine bestimmte Richtung zu lenken. Die Person sollte die Probleme des Datensatzes erkennen und an dieser Stelle widersprechen.“

Postleitzahl	Geburtsdatum	Geschlecht	Diagnose
18059	06.03.1998	männlich	Influenza
18055	21.09.1995	weiblich	Depression
18106	29.02.1994	männlich	Herzinfakt
...	...	...	...

Tabelle E.1. Auszug eines fiktiven medizinischen Datensatzes, nach [Sch20].

Forschungsdatenmanagement

(G) *Was sind Forschungsdaten und was versteht man unter Forschungsdatenmanagement?*
Der Begriff der Forschungsdaten kann sehr weit gefasst werden. Die Definitionen variieren je nach Forschungsgebiet. Das Forschungsdatenmanagement ist zudem von der jeweiligen Disziplin abhängig.

(H) *Würden Sie einer anderen Person erlauben, Ihre Forschungsdaten einzusehen, nachdem Sie darüber veröffentlicht haben, und warum? Welche Bedingungen würden Sie stellen? Wie viele Daten würden Sie freigeben?*
Forschungsdaten sind oft nicht das Eigentum eines einzelnen Wissenschaftlers. Die Entscheidung, ob und wenn ja, welche Daten veröffentlicht werden, liegt also nicht allein beim zugehörigen Wissenschaftler. Zudem kann das Veröffentlichen der Forschungsdaten kann im Widerspruch zu bestimmten Interessen stehen, welche an dieser Stelle ermittelt werden sollen.

(I) *Steht dies für Sie im Widerspruch zur Idee der offenen Wissenschaft? Wie ist Ihr Standpunkt dazu?*
Die Idee hinter Open Science ist, dass sowohl wissenschaftliche Publikationen als auch die zugrundeliegenden Forschungsdaten frei zugänglich gemacht werden, d.h. kostenlos und ohne Einschränkungen zur Verfügung gestellt werden.

F. Veröffentlichungen und betreute Arbeiten

F.1. Vorarbeiten und Veröffentlichungen

Konferenzbeiträge

1. **T. Auge**, A. Heuer: ProSA — Using the Chase for Provenance Management. *ADBIS 2019*
2. **T. Auge**, A. Heuer: Tracing the History of the Baltic Sea Oxygen Level. *BTW 2021*
3. **T. Auge**, M. Hanzig, A. Heuer: ProSA Pipeline — Provenance conquers the Chase. *ADBIS 2022*

Workshop-Beiträge

1. **T. Auge**, A. Heuer: Inverse im Forschungsdatenmanagement. *GvDB 2018*
2. **T. Auge**, A. Heuer: The Theory behind Minimizing Research Data — Result equivalent Chaseinverse Mappings. *LWDA 2018*
3. **T. Auge**: Extended Provenance Management for Data Science Applications. *PhD@VLDB 2020*
4. **T. Auge**, E. Manthey, S. Jürgensmann, S. Feistel, A. Heuer: Schema Evolution and Reproducibility of Long-term Hydrographic Data Sets at the IOW. *LWDA 2020*
5. **T. Auge**, N. Scharlau, A. Heuer: Provenance and Privacy in ProSA — A Guided Interview on Privacy-Aware Provenance. *DEXA Workshops 2021*

Poster und Kurzbeiträge

1. **T. Auge**, A. Heuer: Combining Provenance Management and Schema Evolution. *ProvenanceWeek 2018*
2. **T. Auge**, N. Scharlau, A. Heuer: Privacy Aspects of Provenance Queries. *ProvenanceWeek 2020*

Technische Berichte und Pre-Prints

1. **T. Auge**, N. Scharlau, A/,.Görres, J. Zimmer, A. Heuer: ChaTEAU — A Universal Toolkit for Applying the Chase. `https://arxiv.org/pdf/2206.01643.pdf`
2. **T. Auge**, A. Heuer: Testing provenance systems. Technical Report CS 01-22, `https://eprints.dbis.informatik.uni-rostock.de/1076/`
3. **T. Auge**, A. Heuer: Enhanced Inversion of Schema Evolution with Provenance. (Veröffentlichung auf `https://arxiv.org` am 28.11.2022)

Weitere Beiträge

1. **T. Auge**: Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen. *Masterarbeit*, 2017
2. **T. Auge**, A. Heuer: Provenance Management für Data-Science-Anwendungen unter Berücksichtigung von Daten- und Schema-Evolution. *Exposé zum Promotionsvorhaben*, 2018
3. **T. Auge**: Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen. *Abstrakt im Magazin der Fachgruppe Frauen und Informatik, GI*, 2019

F.2. Betreute studentische Abschlussarbeiten und Projekte (themenbezogener Auszug)

KSWS/Projekt/NEidI

1. F. Renn, F. Röger: Chase-Tools (Teil I), *KSWS*, WS 2017/18
2. W. Brekenfelder, R. Kronenberg, S. Rutofski, S. Schimanski, J. Zimmer: Chase-Tools (Teil II), *Projekt*, SS 2019
3. R. Flach, L. Herrmann, C. Röhrs, A. Strelnikov: Data Provenance-Tools (Teil I), *KSWS*, SS 2020
4. W. Brekenfelder, M. Lamster, R. Kronenberg, E. Manthey, J. Zimmer: Chase-Tools (Teil III), *NEidI*, SS 2020
5. R. Flach, M. Lamster, C. Röhrs, N. Scharlau: Data Provenance-Tools (Teil II), *Projekt/NEidI/Gebietsseminar*, WS 2020/21
6. L. Förster, M. Heuser, J.-H. Overath, A.-S. Waterstradt, Wolpers: Data Provenance, *KSWS*, SS 2021
7. I. Kavisanczki, T. Rudolph, T. Siegl, M. Zuska: sqltsttg, *KSWS*, SS 2021
8. M. Albus, E. Buch, L. Görtz, M. Hanzig, E. Maier: Qualitätssicherung für ChaTEAU, *KSWS*, SS 2021

Bachelorarbeiten

1. Fabian Renn: Erweiterung des Chase-Werkzeugs ChaTEAU um Anfragetransformationen. Bachelorarbeit, WS 2018/19
2. Nic Scharlau: Provenance und Privacy in ProSA. Bachelorarbeit, WS 2019/20
3. Jakob Zimmer: Vereinheitlichung des Chase auf Instanzen und Anfragen am Beispiel ChaTEAU. Bachelorarbeit, WS 2019/20
4. Erik Manthey: Beschreibung der Veränderungen von Schemata und Daten am IOW mit Schema-Evolutions-Operatoren. Bachelorarbeit WS 2019/20
5. Ivo Kavisanczki: Erweiterung des ProSA-Parsers. Bachelorarbeit, WS 2021/22
6. Moritz Hanzig: Ein Framework für ProSA. Bachelorarbeit, WS 2021/22
7. Eric Maier: Erweiterung der ProSA-Pipeline um die Verarbeitung einzelner Schemaevolutionschritte. Bachelorarbeit, SS 2022

Masterarbeiten

1. Martin Jurklies: Chase und Backchase: Entwicklung eines Universal-Werkzeugs für eine Basistechnik der Datenbankforschung. Masterarbeit, SS 2018
2. Andreas Görres: Erweiterung des ChaseWerkzeugs ChaTEAU um ein Terminierungskriterium. Masterarbeit, WS 2019/20
3. Jakob Zimmer: Datenbankintegration durch inverse Schemaabbildungen: Erweiterung der Rostocker GaLVE-Technik. Masterarbeit, SS 2021
4. Maximilian Lamster: Provenance-unterstütze Datenanalyse in Kombination mit intensionalen Antworten zur Steigerung der Privatsphäre. Masterarbeit SS 2021
5. Nic Scharlau: Anonymisierung von Data Provenance in ProSA. Masterarbeit, WS 2021/22
6. Dennis Spolwind: Inverse Anfragen in ProSA. Masterarbeit, WS 2021/22
7. Maximilian Tilman Kaseler: So-tgds und Skolemisierung für ChaTEAU. Masterarbeit, SS 2022

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Rostock, den 28.11.2022 _____

Curriculum Vitae

Persönliche Daten

Name	**Tanja Auge**
Geburtstadum	**25.11.1991**
Geburtsort	**Hamburg**

Beruflicher Werdegang

seit 2022	**Wiss. Mitarbeiterin**, *Lehrstuhl für Data Engineering*, Universität Regensburg
2017 – 2022	**Wiss. Mitarbeiterin**, *Lehrstuhl für Datenbank- und Informationssysteme*, Universität Rostock
2017 – 2018	**Wiss. Mitarbeiterin**, *Lehrstuhl für Praktische Informatik*, Universität Rostock

Ausbildung

Juli 2023	**Promotion**, Universität Rostock (*Provenance Management unter Verwendung von Schemaabbildungen mit Annotationen*)
Oktober 2017	**Master Informatik**, Universität Rostock — Master Informatik (Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen)
April 2016	**Master Mathematik**, Universität Rostock (Reinkomponentenzerlegung über Gauß- und Lorentzbasen für NMR-spektroskopische Daten)
Februar 2014	**Bachelor Mathematik**, Universität Hamburg (Die Weihrauchgrade klassischer Sätze der Mathematik — Ein Vergleich des Zwischenwertsatzes und des schwachen Lemmas von König)
Juli 2009	**Abitur**, Johann-Rist-Grymnasium Wedel