Masterarbeit

Provenance-unterstütze Datenanalyse in Kombination mit intensionalen Antworten zur Steigerung der Privatsphäre

eingereicht von: Maximilian Lamster
Matrikelnummer: 214207151
Geburtsdatum: 08.07.1995

eingereicht am: 31.08.2021

Gutachter: Prof. Dr. rer. nat. habil. Andreas Heuer
apl. Prof. Dr.-Ing. habil. Meike Klettke

Zusammenfassung

Die letzten Jahre hatten einen großen Einfluss auf die Erforschung der Data Provenance in verschiedenen Bereichen. Während sich zurückliegende Forschungsarbeiten dabei hauptsächlich auf die Steigerung der Nachvollziehbarkeit, Reproduzierbarkeit und Qualität von Produktionsprozessen durch entsprechende Provenance-Notationen konzentrieren, beschäftigen sich junge Arbeiten vor allem mit dem Problem der Privacy, das durch Data Provenance entstehen kann. Zum einen kann mittels einer Provenance-Anfrage der relevante Teildatensatz einer Datenbank ermittelt werden, um z.B. die Nachvollziehbarkeit einer Datenanalyse zu erhöhen, zum anderen kann das Ergebnis einer solchen Anfrage private Daten sichtbar machen. Eine Lösung dieses Problems wird z.B. in der intensionalen Beantwortung von Provenance-Anfragen vermutet, die nicht den Originaldatensatz, sondern nur dessen relevanten Teil in anonymisierter Form zurückliefern. Eine Möglichkeit zur Beantwortung einer intensionalen Provenance-Anfrage wurde bereits durch die extensionale Beantwortung der Provenance-Anfrage mit anschließender Generalisierung in [Sva16] konzipiert. Da ein generalisierter Teildatensatz alleine aber nicht unbedingt zur Nachvollziehbarkeit eines Anfrageergebnisses führt, wird in dieser Arbeit ein neuer Ablauf einer Provenance-unterstützten Datenanalyse zur Steigerung der Privatsphäre konzipiert. Eine solche Datenanalyse nutzt die Provenance aktiv, um den Datensatz auf einen relevanten Teil zu reduzieren und diesen anschließend zu anonymisieren. Durch eine Anpassung der eigentlichen Auswertungsanfrage und den Vergleich anonymer Ergebnisse ist dann eine Nachvollziehbarkeit unter Privacy-Aspekten möglich. Die Formulierung einer solchen Datenanalyse, die Umsetzung einzelner Komponenten unter Beachtung der Privacy sowie eine toolunabhängige Formulierung des Gesamtkonzeptes mit einer prototypischen Implementierung als Proof of Concept bilden den Gesamtbeitrag dieser Arbeit zur Thematik Provenance und Privacy.

Abstract

The past years had a major impact on research in various areas of data provenance. While previous research mainly focuses on increasing understandability, reproducibility and quality of a production process through appropriate provenance notations, recent work mainly deals with the problem of privacy, which arises from the specification of data provenance itself. On one hand the relevant data set of a result can be determined by a provenance query, e.g. to increase the traceability, on the other hand the result of such a query can release private information. One solution to this problem is assumed to be the intensional answering of such provenance queries that only return the relevant data set in an anonymized form. One possibility for answering a provenance query in an intensional way has been designed as the combination of extensional answering with subsequent generalization [Sva16]. Since a generalized but relevant data set alone doesn't necessarily lead to the desired traceability of a query result, a new way of a provenance-supported data analysis is proposed to increase privacy while maintaining traceability. Such an analysis actively uses provenance to calculate the relevant data set that is then anonymized. By adapting the original query and comparing the anonymous results, traceability under privacy aspects can be feasible. The formulation of such a data analysis, the implementation of individual components in compliance with privacy and a toolindependent formulation of the overall concept with a prototype implementation as proof of concept is the overall contribution of this work on the topic of provenance and privacy.

Inhaltsverzeichnis

1. Einleitung

Die vorliegende Masterarbeit "*Provenance-unterstützte Datenanalyse in Kombination mit intensionalen Antworten zur Steigerung der Privatsphäre*" befasst sich mit einer möglichen Vereinigung von zwei kontrovers erscheinenden Prinzipien bzw. Konzepten: **Provenance** und **Privacy**.

1.1. Einführung in die Thematik

Provenance, was auch Herkunft oder Ursprung im Deutschen bedeutet, ist ein Konzept mit bereits weit verbreiteter Anwendung, allerdings unter der jeweiligen Bezeichnung des entsprechenden Anwendungsfalls wie z.B. Verständlichkeit, Reproduzierbarkeit oder Qualität. Provenance-Informationen sind dabei nichts anderes als Metadaten, welche Objekte und Aktionen beschreiben, die an einem gewissen Ergebnis eines Produktionsprozesses beteiligt sind. Genau diese Provenance-Informationen bzw. deren Erhebung und Speicherung sorgt für eine größere Verständlichkeit des Produktionsprozesses oder Endproduktes, erlaubt die Nachvollziehbarkeit und somit Reproduzierbarkeit des erhaltenen Ergebnisses und sichert dessen Qualität [HDBL17]. Allerdings erlauben vor allem umfangreiche Provenance-Informationen das Ziehen von Rückschlüssen auf Originaldaten bzw. Objekte, was im Konflikt zu den Prinzipien der Privacy steht.

Privacy oder auch Datenschutz ist dazu da, die "*Grundrechte und Grundfreiheiten natürlicher Personen und insbesondere deren Recht auf Schutz personenbezogener Daten*" zu bewahren (*Europäische Datenschutz-Grundverordnung*[1] *(DSGVO)* Artikel 1). Artikel 4 der *DSGVO* definiert dabei personenbezogene Daten als "*alle Informationen, die sich auf eine identifizierte oder identifizierbare natürliche Person* [...] *beziehen*". Vor allem die ungewollte Verarbeitung personenbezogener Daten soll mit einer solchen grundlegenden Gesetzgebung unterbunden werden. Als Verarbeitung wird in Artikel 4 "*das Erheben, das Erfassen, die Organisation, das Ordnen, die Speicherung, die Anpassung oder Veränderung, das Auslesen, das Abfragen, die Verwendung, die Offenlegung durch Übermittlung, Verbreitung oder eine andere Form der Bereitstellung, den Abgleich oder die Verknüpfung, die Einschränkung, das Löschen oder die Vernichtung*" gezählt.

Nehmen wir an, es wurden personenbezogene Daten im Rahmen der Gesetzgebung für eine Analyse erhoben und gespeichert, so müssen die Ergebnisse aus diesen Analysen in einer Art veröffentlicht werden, welche weder die Privacy verletzt noch die Nachvollziehbarkeit der Analyse und dessen Rekonstruktion einschränkt. Auch bei Forschungsdaten ohne jeglichen Personenbezug ist eine solche Bewahrung geistigen Eigentums bei der Veröffentlichung wünschenswert, da die Datenerhebung meist mit erhöhtem Zeit- und Kostenaufwand verbunden ist. Auch mit Blick auf die im Jahr 2016 veröffentlichten *FAIR-Prinzipien*[2], die zur Verbesserung von **F**indability, **A**ccessability, **I**nteroperability und **R**euse für eine optimale Nachnutzung von Forschungsdaten auffordern [WDA+16], ist eine Vereinbarung von Provenance und Privacy zielführend.

[1] https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:02016R0679-20160504
[2] https://www.go-fair.org/fair-principles/

Nic Scharlau beschäftigte sich in seiner Bachelorarbeit [Sch20] bereits mit den auftretenden Privacy-Problemen, die bei der Anwendung von Data Provenance entstehen können. Außerdem erläutert er einige Möglichkeiten für die Vereinbarung beider Konzepte, die im weiteren Verlauf der Arbeit ebenfalls betrachtet werden. Ein Lösungsvorschlag für das *Provenance-Privacy-Problem (PPP)* könnte nach [Sch20] die Umwandlung extensionaler Provenance-Antworten, welche beteiligte Datenelemente explizit nennen, in intensionale Provenance-Antworten sein. Intensionale Antworten liefern im Gegensatz zu extensionalen Antworten nur eine Beschreibung der eigentlichen Daten [Mot94] und sind somit eher mit dem Konzept der Privacy vereinbar. Jan Svacina erforschte in seiner Bachelorarbeit [Sva16] bereits die Möglichkeit der intensionalen Beantwortung von Provenance-Anfragen mittels Generalisierung. Inwieweit sich Datensätze und Provenance-Anfragen nun genau intensionalisieren lassen und ob intensionale Provenance-Antworten für die Nachvollziehbarkeit und Reproduzierbarkeit der eigentlichen Ergebnisse noch genutzt werden können, sind zentrale Fragestellungen dieser Arbeit. Im nächsten Abschnitt wird dazu ein Beispielablauf einer Datenanalyse vorgestellt, der das PPP und resultierende Fragestellungen sehr anschaulich präsentiert und motiviert.

1.2. Beispielablauf einer Datenanalyse

Anhand eines Beispielablaufs einer Datenanalyse soll die Problematik von Provenance und Privacy in diesem Abschnitt noch einmal bildlich und im Kontext der Betrachtungen dieser Arbeit dargestellt werden. Eine grundlegende oder auch konventionelle Datenanalyse ist dabei in Abbildung 1.1 zu sehen.

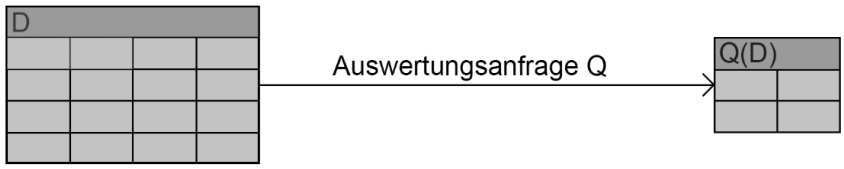

Abbildung 1.1.: Beispiel einer Datenanalyse

Wie in Abbildung 1.1 erkennbar its, beschränken wir uns bei der Formulierung einer Datenanalyse auf das relationale Datenbankmodell. Sei D eine Datenbankinstanz mit einer oder mehreren Relationen bzw. Tabellen (aus Platzgründen ist in Abbildung 1.1 nur eine Tabelle dargestellt). Eine Datenanalyse sei dann die Ausführung der Auswertungsanfrage Q auf dem Datensatz D, was zum Anfrageergebnis bzw. der Zielrelation Q(D) führt. Im Rahmen einer Veröffentlichung soll das Ergebnis Q(D) verständlich, nachvollziehbar und reproduzierbar sein. Dies erfordert neben der Veröffentlichung von Q(D) auch eine Veröffentlichung von D, Q und im besten Fall auch Provenance-Informationen. In den meisten Szenarien ist für die Rekonstruktion des Ergebnisses Q(D) nicht immer der gesamte Datensatz D, sondern nur eine relevante Teilmenge **D** erforderlich. Dieser relevante Teil des Datensatzes reicht für eine Sicherung der Reproduzierbarkeit somit aus. Um diesen relevanten Teil der Datenbank ermitteln zu können, wird als Hilfsmittel die *Data Provenance* genutzt. Data Provenance, z.B. in Form der Why-Provenance, speichert für jede Zeile (Tupel) des Ergebnisses Q(D) die eindeutigen Identifikatoren aller für dieses Ergebnistupel relevanten Zeilen des originalen Datensatzes D ab. Diese Identifikatoren zeigen im Falle der Why-Provenance auf die Zeugen des entsprechenden Ergebnistupels, also genau jene Zeilen des Originaldatensatzes, die an der Produktion der entsprechenden Zeile im Ergebnis Q(D) beteiligt waren. Durch Ausnutzung dieser Informationen lässt sich dann mittels einer Provenance-Anfrage der relevante Teil der Datenbankinstanz D bestimmen. Provenance-Informationen steigern durch die Formulierung der Herkunft entsprechender

Ergebnistupel aber auch die Nachvollziehbarkeit und Verständlichkeit, weswegen die zusätzliche Erhebung von Provenance-Informationen besonders wertvoll für die Nachnutzung ist [WDA+16]. Eine konventionelle Datenanalyse mit zusätzlicher Ausnutzung von Provenance-Informationen ist in Abbildung 1.2 dargestellt.

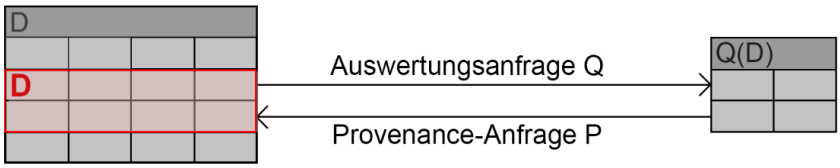

Abbildung 1.2.: Erweiterung einer Datenanalyse mittels Provenance-Nutzung

Mittels Data Provenance lässt sich der Datensatz D also auf einen relevanten Teildatensatz **D** reduzieren, ohne dabei die Reproduzierbarkeit des Ergebnisses Q(D) zu gefährden. Zusätzlich erfüllt die Datenanalyse durch die Angabe von Provenance-Informationen weitere Kriterien für eine gesteigerte Verständlichkeit und Nachvollziehbarkeit. Eine Veröffentlichung des relevanten Teildatensatzes sichert also wichtige Anforderungen und reduziert dabei die Anzahl veröffentlichter Originaldaten auf ein Minimum, was ein wichtiger Schritt in Richtung Privacy ist. Enthält der Datensatz D allerdings personenbezogene Daten, die es ermöglichen, einzelne Individuen indirekt oder sogar direkt zu identifizieren, so verstößt auch eine Veröffentlichung des relevanten Teildatensatzes **D** schon gegen grundlegende Prinzipien der Privacy. Zwingend sollte nicht nur die Veröffentlichung personenbezogener Daten, sondern auch die Veröffentlichung unter Kosten- oder Zeitaufwand erhobener Forschungsdaten ebenfalls vermieden werden. Es kann sich hierbe um das geistige Eigentum eines bestimmten Urhebers oder sensitive Informationen handeln, die ebenfalls Privacy-Ansprüchen unterliegen. In der Intention, dass unter solchen Umständen eine Veröffentlichung von Originaldaten vermieden werden soll, bleibt nur noch die Option einer Anonymisierung des Datensatzes D bzw. des relevanten Teils **D**, um wenigstens eingeschränkt die Möglichkeit der Nachvollziehbarkeit und Rekonstruierbarkeit zu bieten, gleichzeitig aber den Anforderungen der Privacy zu entsprechen. Ein Konzept für einen resultierenden Ablauf einer Datenanalyse inklusive Provenance-Ausnutzung und Anonymisierung ist in Abbildung 1.3 dargestellt.

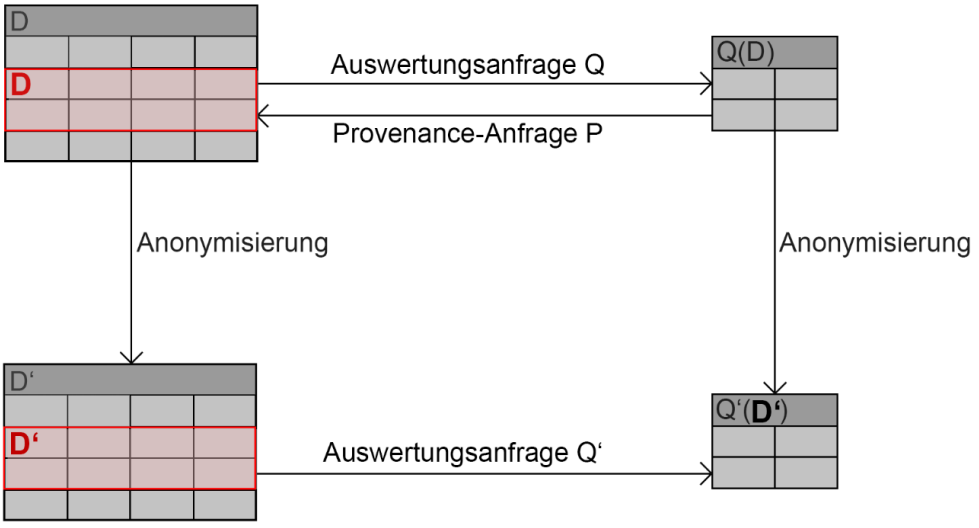

Abbildung 1.3.: Erweiterung einer Datenanalyse mittels Anonymisierung

Problematisch ist allerdings, dass die vorher formulierte Auswertungsanfrage Q durch spezielle An-

onymisierungstechniken unter Umständen nicht mehr mit dem anonymisierten Datensatz D' bzw. **D'** kompatibel ist. Ein anonymisierter Datensatz erfordert demnach eine an die Form der Anonymisierung angepasste Auswertungsanfrage Q', die dann aber auch zu einem anonymisierten Ergebnis Q'(**D'**) führen kann und deshalb nicht zwingend zur Reproduzierbarkeit von Q(D) beitragen muss. Allerdings kann das Ergebnis einer Anonymisierung des Ergebnisses Q(D) einen möglichen Zusammenhang zum Ergebnis der Auswertungsanfrage Q' liefern, um eine eingeschränkte Nachvollziehbarkeit und Validierung des Ergebnisses Q(D) zu ermöglichen.

Der Ausgangspunkt eines entsprechend angepassten Ablaufs einer Datenanalyse ist jedoch immer noch D bzw. **D**, was im Widerspruch zum Privacy-Konzept steht. Um den Weg von D bzw. **D** und die damit verbundene Veröffentlichung originaler Daten zu vermeiden, kann z.B. eine intensionale Provenance-Anfrage genutzt werden. Eine solche intensionale Provenance-Anfrage liefert als Ergebnis nämlich nur den bereits anonymisierten relevanten Teildatensatz **D'**. Legt man Q(D) nun als neuen Ausgangspunkt fest, so kann mittels einer intensionalen Provenance-Anfrage P_i, der modifizierten Auswertungsanfrage Q' und der Anonymisierungsmethode von Q(D) eine abgeschwächte Nachvollziehbarkeit und Reproduzierbarkeit unter Einhaltung von Privacy-Aspekten gewährleistet werden. Der Zusatz einer intensionalen Provenance-Anfrage zur Bildung einer Privacy-konformen Datenanalyse ist in Abbildung 1.4 dargestellt.

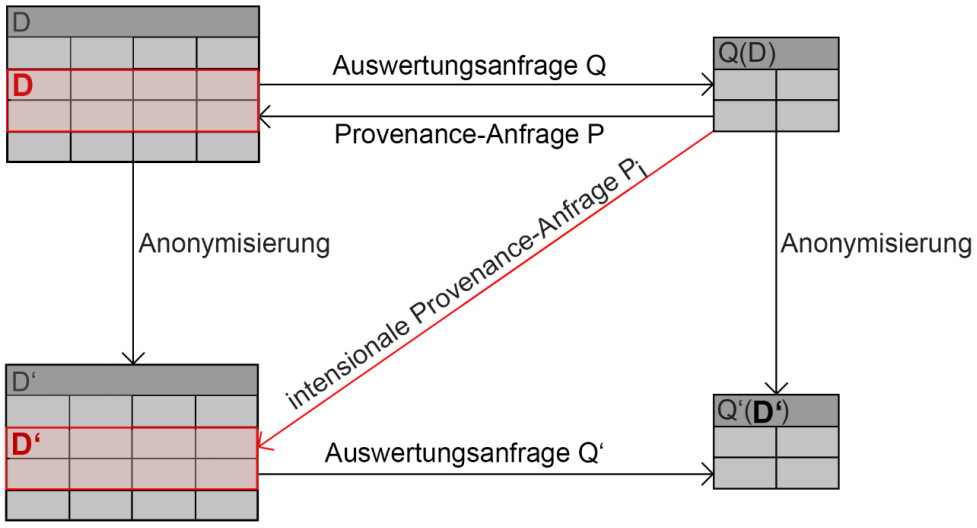

Abbildung 1.4.: Erweiterung einer Datenanalyse mittels intensionaler Provenance-Anfrage

Der gezeigte Beispielablauf in Abbildung 1.4 erscheint zunächst logisch, ist bisher aber nur eine Wunschvorstellung. Deren praktische Umsetzung muss für verschiedene Formen von Auswertungsanfragen Q unter Betrachtung theoretischer Grundlagen (Kapitel 2) und dem aktuellen Stand der Forschung (Kapitel 3) erst erst definiert werden. Es müssen auch entsprechende Fragen zur Privacy und vor allem den verschiedenen Umsetzungs- und Kombinationsmöglichkeiten dieses offen gestalteten Konzeptablaufs diskutiert werden. Die Auseinandersetzung mit den resultierenden Fragestellungen beginnt jedoch erst in Kapitel 4. Im nächsten Abschnitt 1.3 wird für eine anschauliche Gestaltung weiterer Betrachtungen und Erklärungen ein Beispieldatensatz inklusive Anfragen eingeführt.

1.3. Einführung Beispieldatensatz

In diesem Abschnitt wird ein Beispieldatensatz D inklusive möglicher Auswertungsanfragen Q_i verschiedener Schwierigkeitsstufen i eingeführt. Durch die Einführung von Beispielrelationen und Beispielanfragen lassen sich die Grundlagen zum Relationenmodell, aber auch die Grundlagen zur Notation der verschiedenen Data Provenance-Typen viel einfacher erklären. Für eine gesteigerte Übersichtlichkeit besteht die Beispieldatenbank nur aus zwei Tabellen. Die Tabelle 1.1 NOTEN beinhaltet dabei nur acht Einträge, die durch eine Kursnummer (kurs_nr), eine StudierendenID (student_id), das Semester (semester) und die entsprechende Note (note) charakterisiert sind. Die Tabelle 1.2 STUDIERENDE beinhaltet nur fünf Einträge verschiedener Personen, charakterisiert durch eine StudierendenID (student_id), Nachname (nachname), Vorname (vorname) und Studiengang (studiengang). Man beachte, dass die erste Spalte (ohne Überschrift) nicht zum eigentlichen Datensatz gehört, sondern nur einen eindeutigen Tupelidentifikator darstellt, welcher später verwendet wird, um die einzelnen Tupel (Zeilen) im Bereich der Data Provenance eindeutig zu referenzieren.

	kurs_nr	**student_id**	**semester**	**note**
N_1	001	1	SS 16	1.7
N_2	001	2	SS 16	1.3
N_3	001	3	SS 16	2.3
N_4	001	4	SS 16	3.3
N_5	002	1	WS 15/16	3.0
N_6	002	3	WS 15/16	1.0
N_7	002	4	WS 15/16	2.7
N_8	002	5	WS 15/16	1.7

Tabelle 1.1.: Tabelle `NOTEN`

	student_id	**nachname**	**vorname**	**studiengang**
S_1	1	Muster	Max	Informatik
S_2	2	Neuhaus	Nancy	Informatik
S_3	3	Altmann	Anne	Mathematik
S_4	4	Fern	Friedrich	Informatik
S_5	5	Unruh	Uwe	Mathematik

Tabelle 1.2.: Tabelle `STUDIERENDE`

Neben den zwei vorgestellten Tabellen enthält das Beispiel auch vier Anfragen verschiedenen Typs. Anfrage 1.1 stellt eine Selektion, Anfrage 1.2 eine Projektion, Anfrage 1.3 einen Verbund mit Selektion und Anfrage 1.4 eine Aggregatfunktion dar. Der Sinn hinter jeder Anfrage ist in dem Label der jeweiligen Anfrage wörtlich formuliert.

```
1 SELECT * FROM NOTEN WHERE note = 1.7;
```

Anfrage 1.1: Selektion von NOTEN auf Einträge mit der Note 1.7

```
1 SELECT DISTINCT studiengang FROM STUDIERENDE;
```

Anfrage 1.2: Projektion von STUDIERENDE auf belegte Studiengänge

```
1 SELECT
2     s.student_id, s.lastname, s.firstname, s.studiengang,
3     n.kurs_nr, n.semester, n.note
4 FROM
5     STUDIERENDE s INNER JOIN NOTEN n
```

```
    ON s.student_id = n.student_id
WHERE
    n.note = 1.0
;
```

Anfrage 1.3: Verbund von NOTEN und STUDIERENDE mit Selektion auf Einträge mit der Note 1.0

```
SELECT kurs_nr, AVG(note) AS durchschnitt FROM NOTEN GROUP BY kurs_nr;
```

Anfrage 1.4: Berechnung der Durchschnittsnote für verschiedene Kursnummern der NOTEN-Tabelle

Der anschließende Abschnitt 1.4 beschließt das Kapitel der Einleitung mit der Präsentation des weiteren Verlaufs dieser Masterarbeit.

1.4. Aufbau der Arbeit

An dieser Stelle wird der Aufbau der Arbeit für einen besseren Überblick dargestellt. Kapitel 2 behandelt nach dieser Einleitung die Grundlagen des relationalen Datenbankmodells in Abschnitt 2.1, die Grundlagen der Provenance in Abschnitt 2.2, die Grundlagen der Data Provenance in Abschnitt 2.3 und abschließend die Grundlagen zum Thema Privacy in Abschnitt 2.4. Kapitel 3 geht dann auf den aktuellen Stand der Forschung ein. Unter anderem werden die Kombination von Provenance und Privacy in Abschnitt 3.1, die Kombination von Data Provenance und Privacy in Abschnitt 3.2 sowie die Möglichkeit zur Formulierung intensionaler Antworten in Abschnitt 3.3 auf Basis vorangegangener Forschungen vorgestellt. Kapitel 4 behandelt dann das Konzept dieser Arbeit. Die vorgestellten Bestandteile des Kapitels sind die Auswertungsanfrage Q in Abschnitt 4.1, die Intensionalisierung von Datensätzen in Abschnitt 4.2, der Datensatz D in Abschnitt 4.3, die Generalisierung in Abschnitt 4.4, die angepasste Auswertungsanfrage Q' in Abschnitt 4.5, eine ID-basierte Notation der Provenance in Abschnitt 4.6, die ID-basierte Formulierung von Provenance-Anfragen in Abschnitt 4.7 und die intensionale Beantwortung solcher Provenance-Anfragen in Abschnitt 4.8 für die schlussendliche Zusammensetzung eines Gesamtablaufs in Abschnitt 4.9. Kapitel 5 enthält anschließend Informationen zur Implementierung, darunter die Wahl der Programmier- und Datenbanksprache in Abschnitt 5.1, die Beschreibung des Programmaufbaus und des Programmablaufs in Abschnitt 5.2 sowie getestete Anfragen und Argumente in Abschnitt 5.3. Kapitel 6 beendet diese Arbeit dann mit einem Fazit in Abschnitt 6.1 und gibt anschließend noch einen Ausblick auf weitere Forschungsmöglichkeiten in Abschnit 6.2 an.

2. Grundlagen

Die in diesem Kapitel vorgestellten Grundlagen sollen das Erfassen der im späteren Verlauf vorgestellten Konzepte, Ideen und Umsetzungen erleichtern. Angefangen wird dabei in Abschnitt 2.1 mit der Vorstellung des relationalen Datenbankmodells von [Cod70] und einigen notwendigen Operatoren der entsprechenden Relationenalgebra in Abschnitt 2.1.1. In Abschnitt 2.2 wird dann das Thema der Provenance genauer betrachtet, wozu eine genaue Definition in Abschnitt 2.2.1, die verschiedenen Provenance-Typen in Abschnitt 2.2.2 und die unterschiedlichen Anwendungsfälle der Provenance in Abschnitt 2.2.3 gehören. Danach wird in Abschnitt 2.3 die Thematik der Data Provenance noch einmal genauer behandelt. Dazu zählen die typischen Fragestellungen ***Why?***, ***How?*** und ***Where?*** in den Abschnitten 2.3.2, 2.3.3 und 2.3.4. Abschnitt 2.4 präsentiert anschließend den Privacy-Begriff, angefangen mit einem Beispiel zur Notwendigkeit des Konzepts und der Präsentation zweier bekannter Anonymitätsmaße in Abschnitt 2.4.1. In den Abschnitten 2.4.2 bis 2.4.6 werden abschließend noch einige Anonymisierungsmethoden zur Sicherung der Privacy vorgestellt.

2.1. Relationales Datenbankmodell

Wegen seiner Verbreitung, Einfachheit und Exaktheit wird auch in dieser Arbeit das relationale Datenbankmodell für die Darstellung von Datensätzen verwendet. Der Name des Modells leitet sich von dem Begriff Relation als eine mögliche mathematische Darstellungsform für eine Tabelle ab. Das Datenmodell geht dabei auf [Cod70] aus dem Jahre 1970 zurück. Das relationale Datenbankmodell wird im Folgenden aber auf Basis von [HSS18] vorgestellt.

Definition 2.1: (*Universum, Attribut und Wertebereich*) Wir bezeichnen im Folgenden eine endliche, nicht-leere Menge von Attributen als *Universum* U. Ein Element A dieser Menge U heißt *Attribut*. Sei nun $D = \{D_1, \ldots, D_m\}$ eine Menge endlicher, nicht-leerer Mengen mit $m \in \mathbb{N}$, so wird jedes $D_i \in D$ als *Wertebereich* oder Domäne bezeichnet und es existiert die total definierte Funktion $dom : U \to D$. Der Wertebereich bzw. die Domäne eines Attributes A wird durch den Ausdruck $dom(A)$ dargestellt. Ein Element $w \in dom(A)$ heißt dementsprechend *Attributwert* von Attribut A.

Definition 2.2: (*Relationenschema, Relation und Tupel*) Sei R eine Menge von Attributen mit $R \subseteq U$, so spricht man bei R von einem *Relationenschema*. Eine *Relation* r über ein Relationenschema $R = \{A_1, \ldots, A_n\}$ (kurz: r(R)) mit $n \in \mathbb{N}$ ist eine endliche Menge von Abbildungen $t : R \to \bigcup_{i=1}^{m} D_i$, wobei gilt, dass jedes $t(A) \in dom(A)$ ist. Jede Abbildung t trägt dabei die Bezeichnung *Tupel*.

Definition 2.3: (*Datenbankschema und Datenbank*) Eine Menge von Relationenschemata $S := \{R_1, \ldots, R_q\}$ mit $q \in \mathbb{N}$ wird als *Datenbankschema* bezeichnet. Eine *Datenbank* eines entsprechenden Datenbankschemas S ist dann eine Menge von Relationen $d := \{r_1, \ldots, r_p\}$, sodass gilt: $r_i(R_i)$ für alle $i \in \{1, \ldots, q\}$.

Begriffe am Beispiel: Um die Begriffe Relationenschema, Relation, Attribut und Tupel noch einmal bildlich am in Abschnitt 1.3 definierten Beispiel darzustellen, folgt eine beschriftete Darstellung der NOTEN-Tabelle in Abbildung 2.1.

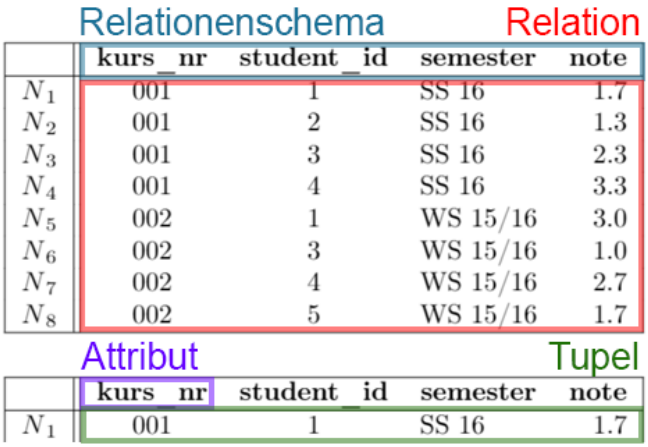

Abbildung 2.1.: Veranschaulichung der Begriffe des Relationenmodells

2.1.1. Relationenalgebra

Um Datensätze des relationalen Datenbankmodells zu verändern und innerhalb solcher Datensätze suchen zu können, sind relationale Operatoren notwendig. Die Relationenalgebra, das erste formale Anfragemodell und ebenfalls vorgestellt in [Cod70], ist eine beliebte Basis für verschiedene relationale Datenbanksprachen. Im Folgenden werden einige Standardoperationen der Relationenalgebra beschrieben, die im weiteren Verlauf dieser Arbeit betrachtet werden. Die einzelnen Definitionen basieren dabei wieder auf [HSS18].

Definition 2.4: (*Projektion*) Die Projektion hat die folgende Syntax: $\pi_{Attributmenge}(Relation)$. Sei r(R) eine Relation und $X \subseteq R$ eine Attributmenge in R, dann ist die Semantik erklärt durch: $\pi_X(r) := \{\, t(X) \mid t \in r\}$.

Definition 2.5: (*Selektion*) Die Syntax der Selektion lautet wie folgt: $\sigma_{Bedingung}(relation)$. Die Semantik der Selektion ist dann durch $\sigma_F(r) := \{t \mid t \in r \wedge F(t) = true\}$ erklärt, wobei die Formel F eine der beiden folgenden Bedingungen darstellt:

- F ist eine Konstantenselektion der Form: Attribut θ Konstante, bei der ein bestimmtes Attribut jedes Tupels mit einer vorgegebenen Konstante verglichen wird. Das Vergleichssymbol θ ist dabei entweder $=$ oder $\neq$, bei linear geordneten Wertebereichen kann es aber auch $<, \leq, \geq$ oder $>$ sein.
- F ist eine Attributselektion der Form: Attribut1 θ Attribut2, bei der für jedes Tupel zwei Attributwerte mit kompatiblen Wertebereichen verglichen werden.

Definition 2.6: (*Natürlicher Verbund*) Der natürliche Verbund hat die folgende Syntax: $Relation_1 \bowtie Relation_2$. Die Semantik ist dann wie folgt erklärbar: $r_1 \bowtie r_2 := \{t \mid t(R_1 \cup R_2) \wedge \exists t_1 \in r_1 : t_1 = t(R_1) \wedge \exists t_2 \in r_2 : t_2 = t(R_2)\}$. Der natürliche Verbund verknüpft somit zwei verschiedene Relationen über gleich benannte Spalten unter der Bedingung, dass zwei gleiche Attributwerte in $t1 \in r1$ und $t2 \in r2$ existieren.

Neben den bisher definierten Standardoperationen sind auch Aggregatfunktionen wichtige Bestandteile relationaler Anfragen. Nachfolgend sind die wichtigsten Aggregatfunktionen erklärt:

- **Count:** bestimmt die Anzahl der Werte einer Spalte, im Sonderfall Count(*) die Anzahl der Tupel einer Relation
- **Sum:** berechnet die Summe der Werte einer Spalte mit numerischem Wertebereiche
- **Avg:** berechnet den Mittelwert der Werte einer Spalte mit numerischem Wertebereich
- **Min:** berechnet den kleinsten Wert einer Spalte mit numerischem Wertebereich
- **Max:** berechnet den größten Wert einer Spalte mit numerischem Wertebereich

Neben den bisher vorgestellten Standardoperationen gehören auch grundlegende Mengenoperationen zu den Standardoperationen dazu. Da diese aber zwei Relationen mit demselben Schema und analogen Wertebereichen benötigen, was im späteren Verlauf der Arbeit keinen Anwendungsfall darstellt, ist eine Vorstellung dieser Operationen an dieser Stelle nicht vorgesehen.

Die bisher vorgestellten Operationen Selektion (S), Projektion (P), natürlicher Verbund (engl. Join (J)) und Aggregation (A) können in einer Anfrage nicht nur einzeln, sondern auch in Kombinationen vorkommen. Im weiteren Verlauf wird deshalb für Anfragen bestehend aus den Standardoperation die Bezeichnung *SPJA-Anfrage* verwendet, die alle möglichen Anfragekombinationen umfasst. Zur Überprüfung bestimmter Anfragekombinationen werden allerdings die entsprechenden Buchstaben verwendet, z.B. SP-Anfrage für eine Kombination von Selektion und Projektion.

Nach der Vorstellung des Datenmodells und wichtigen Operationen auf diesem Modell folgt eine präzise Vorstellung zum Thema Provenance in Abschnitt 2.2.

2.2. Provenance

Provenance ist, auch wenn der Begriff anfänglich nicht geläufig erscheint, ein Prinzip mit häufiger und weitreichender Verwendung. Übersetzbar mit z.B. Ursprung oder Herkunft beschreiben Provenance-Informationen den Werdegang eines gewünschten Produktes. Sei dieses Produkt ganz alltäglich gewählt, z.B. ein Eierkuchen, so sind die Mengenangaben der Zutaten und das Kochrezept bereits Provenance-Informationen zu dem gewählten Produkt. Im technischen Bereich beschränkt man sich bei Provenance-Informationen jedoch oft auf einzelne Workflows oder Datenelemente, welche in der Unterklasse der *Workflow Provenance* bzw. *Data Provenance* Anwendung finden. Eine Firma mit mehreren Standorten will beispielsweise einem anderen Standort einen wichtigen Datensatz bereitstellen. Provenance-Informationen können dann zum Beispiel die Herkunft der Datenelemente beschreiben. Also genau woher bestimmte Datenelemente ursprünglich kommen. Durch die Vielfalt der Interpretation, der Anwendungsmöglichkeiten und resultierenden Provenance-Typen wird in diesem Kapitel ein Überblick zu den verschiedenen Provenance-Klassen in Form einer Hierarchie auf Basis von [HDBL17] gegeben. Auch spezifische Anwendungsfälle der Provenance, die sich in den letzten Jahren herausgebildet haben, werden auf Basis von [HDBL17] in diesem Kapitel präsentiert.

2.2.1. Definition

"*Provenance* generally refers to any information that describes the production process of an end product, which can be anything from a piece of data to a physical object" [HDBL17].

Freie Übersetzung: Unter *Provenance* versteht man alle Informationen, die einen Produktionsprozess eines Endproduktes beschreiben. Dieses Endprodukt kann alles Mögliche sein - von einzelnen Datenelementen bis hin zu physischen Objekten.

Aus dieser sehr allgemein gehaltenen und formalen Definition ist leicht zu erkennen, dass eine weitere Einteilung der Provenance nötig ist, um eine zielgerichtete Anwendung des Konzepts zu ermöglichen. Im nächsten Abschnitt 2.2.2 werden deshalb die verschiedenen Provenance-Klassen erläutert und für ein besseres Verständnis hierarchisch präsentiert.

2.2.2. Provenance-Hierarchie

In diesem Abschnitt werden die verschiedenen Provenance-Klassen basierend auf der in [HDBL17] definierten Hierarchie wiedergegeben. Besonderer Wert wird dabei auf die Zusammenhänge der einzelnen Klassen gelegt, wie auch in Abbildung 2.2 dargestellt. Abbildung 2.2 weicht dabei von der eigentlichen Hierarchie-Darstellung in [HDBL17] in der Form ab, dass nur die Teilmengenbeziehungen der einzelnen Klassen in Bezug auf dazugehörige Produktionsprozesse dargestellt sind.

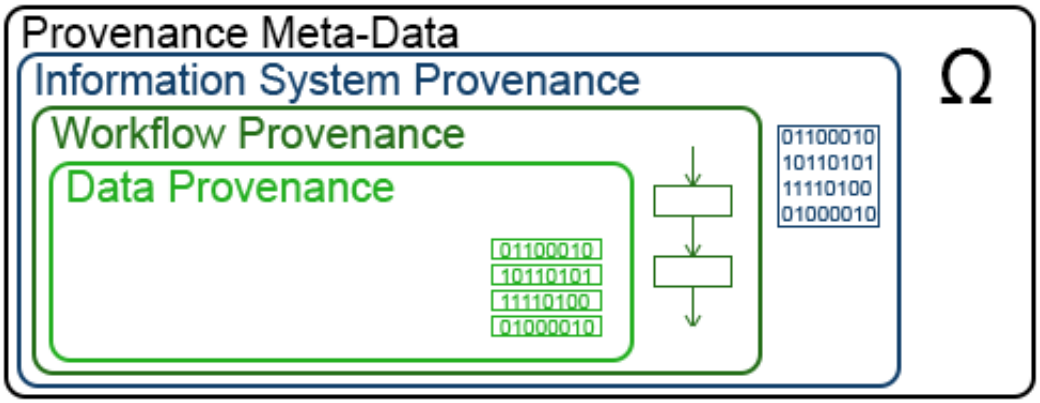

Abbildung 2.2.: Hierarchie der Provenance-Klassen, angelehnt an [HDBL17]

Provenance Meta-Data *Provenance Meta-Data* ist der allgemeinste Provenance-Typ. In diese Klasse gehören alle möglichen Provenance-Informationen zu irgendeinem Produktionsprozess. Provenance-Informationen dieser Klasse haben schlussfolgernd einen großen Interpretationsspielraum für die Verarbeitung, Speicherung und Modellierung, haben dafür aber auch keine vordefinierten Verarbeitungsmöglichkeiten. Auch das in diesem Kapitel erwähnte Eierkuchenrezept wäre Teil dieser Klasse, da das Endprodukt (in diesem Fall Eierkuchen), der Produktionsprozess und die Art der Modellierung der Provenance-Informationen in dieser Klasse frei wählbar sind.

Information System Provenance Zur Klasse der *Information System Provenance* zählen nur jene Produktionsprozesse, die digitale Daten innerhalb von Informationssystemen verarbeiten. Provenance-Informationen lassen sich dabei oft durch Eingabe, Ausgabe und gesetzte Parameter berechnen, während die genaue Struktur des Produktionsprozesses für Außenstehende dabei verborgen bleibt. Ein Informationssystem soll in diesem Fall nur grob ein System bezeichnen, das zur Speicherung, Abrufung, Kommunikation und Verteilung von Informationen genutzt werden kann. Man beachte, dass es sich hierbei also um eine Teilmenge der Produktionsprozesse handelt, wie sie in Provenance Meta-Data noch erlaubt waren. In diese Klasse fallen also alle digitalen Systeme, die sich explizit mit Provenance-Informationen auseinandersetzen. Ein einfaches Beispiel wäre ein Script, welches Provenance-Informationen für einen Server sammelt, in denen der jeweilige Nutzer, die Uhrzeit und ein in der Kommandozeile ausgeführter Befehl abgespeichert werden. Durch die vielfältigen Strukturen der Provenance-Informationen und

die entsprechend geringe Instrumentierung hat sich die Klasse der Workflow Provenance herausgebildet.

Workflow Provenance Bei Provenance-Informationen der *Workflow Provenance* handelt es sich auch wieder um eine Teilmenge der zur Information System Provenance gehörigen Produktionsprozesse. Die Klasse der Workflow Provenance schränkt dabei die Produktionsprozesse weiter auf jene ein, die als Workflow darstellbar sind. Ein Workflow kann dabei als gerichteter Graph modelliert werden, bestehend aus Knoten, welche für Module bzw. Funktionen stehen (charakterisiert durch Eingabe, Ausgabe und Parameter) und aus Kanten, die einen Datenfluss bzw. Kontrollfluss zwischen enstsprechenden Knoten darstellen. Dieses detailiertere Modell erlaubt einen geordneteren Erhalt von Provenance-Informationen mittels Ausnutzung der jeweils gegebenen Graphstruktur. Mittels dieser Struktur lassen sich z.B. Fragen zum Autor gewisser Daten, zu vorgenommenen Modifikationen, Verarbeitungszeitpunkten usw. beantworten. Ein einfaches Beispiel für einen Workflow wäre ein Datenflussdiagramm eines beliebigen informationsverarbeitenden Systems. Der Nachteil der Workflow Provenance ist allerdings, dass sich die Provenance-Informationen nicht auf einzelne Datenelemente, sondern meist auf ganze Datensätze beziehen, was genauere Analysen und das Tracken einzelner Datenelemente zum Problem werden lässt. Diese oftmals notwendigen Anforderungen werden durch die Klasse der Data Provenance gelöst.

Data Provenance Statt weiterhin die zu betrachtenden Produktionsprozesse einzuschränken, limitiert die Klasse der *Data Provenance* nun die Menge der möglichen Provenance-Informationen auf solche, die einzelne Datenelemente beschreiben. Unter anderem wird das Tracken einzelner Datenelemente ermöglicht, was genauere Analysen zu bestimmten Datenelementen erlaubt. Der Erhalt solcher Provenance-Informationen geschieht meist auf Basis strukturierter Datenmodelle und dazugehöriger deklarativer Anfragesprachen mit fester Semantik. Im Unterschied zur Worfklow-Provenance kann nämlich neben der Struktur auch die jeweilige Semantik ausgenutzt werden, auch wenn die Fragestellungen beider Klassen sich ansonsten sehr ähnlich sind. Mit dieser Granularität von Provenance-Informationen lassen sich sehr spezifische Fragestellungen auch für einzelne Datenelemente beantworten, z.B. ***Where*** (Wo kommt das Datenelement her?), ***Why*** (Warum haben wir das Datenelement erhalten?) und ***How*** (Wie ist dieses Datenelement zu Stande gekommen?). Ein Beispiel für Data Provenance ist z.B. die Notation der Originalrelation für jede Zeile einer definierten Sicht, um zu beantworten wo das Tupel oder dessen Anteile denn ursprünglich herkommen. Da diese Klasse mit seinen speziellen Fragestellungen für diese Arbeit von größerer Bedeutung ist, erfolgen in Abschnitt 2.3 noch genauere Ausführungen zur Data Provenance. Doch vorerst werden in 2.2.3 die möglichen Anwendungsfälle von Provenance allgemein vorgestellt.

2.2.3. Anwendungsfälle

Doch wofür können gesammelte Provenance-Informationen denn nun eigentlich genutzt werden? In [HDBL17] werden drei grundlegende Klassen von Anwendungsfällen unterschieden: *Understandability*, *Reconstruction* und *Quality*. Die Reihenfolge der Präsentation beruht dabei auf dem logischen Schluss, dass man einen Produktionsprozess zuerst verstehen muss, bevor man ihn rekonstruieren kann, um schlussendlich die Qualität des entsprechenden Produktionsprozesses oder Endproduktes zu verbessern. Jede der genannten Klassen kann dabei in weitere Unterklassen unterteilt werden, bestimmt durch den Provenance-Urheber und durch die Zielgruppe, an die die Provenance-Informationen gerichtet sind. Abbildung 2.3 stellt die entsprechenden Klassen und Unterklassen in Beziehung und zeigt die zur Unterklasse gehörenden Kombinationen von Provenance-Produzent und -Konsument. **Expert** bezeichnet hierbei eine oder mehrere Personen desselben Fachbereiches, **Self** bezeichnet den Provenance-Autor selbst und **All** bezeichnet eine oder mehrere Personen außerhalb des eigenen Fachbereiches.

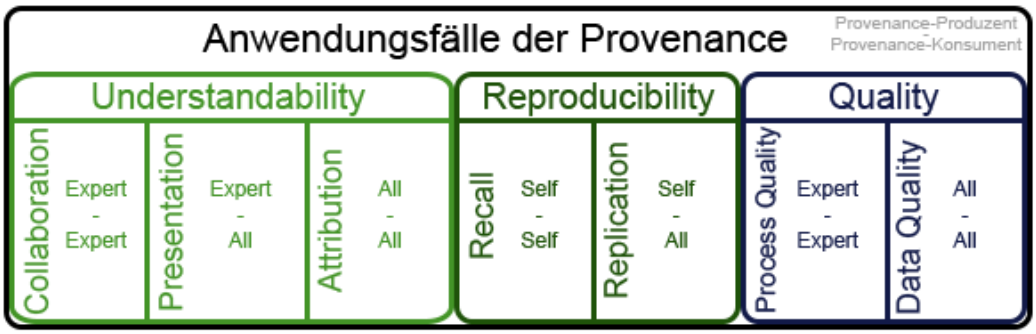

Abbildung 2.3.: Provenance Anwendungsfälle nach [HDBL17]

Understandability

Provenance-Informationen, die darauf abzielen, einen Produktionsprozess für eine bestimmte Zielgruppe transparenter und dadurch verständlicher zu gestalten, fallen in den Bereich der *Understandability*. Da die Vermittlung bzw. Erhöhung der Verständlichkeit durch gesammelte Provenance-Informationen an vielen Stellen und in unterschiedlicher Form Anwendung findet, unterscheidet man die folgenden Unterklassen:

- **Collaboration:** Bei einer Gruppe von Personen, die Provenance-Informationen innerhalb eines gemeinsamen Projektes zur Verbesserung der Verständlichkeit bereitstellen, spricht man von *Collaboration*, einer Unterklasse der Understandability. In Projektarbeiten müssen vorgenommene Änderungen natürlich verständlich und nachvollziehbar für alle anderen Teilnehmer gestaltet werden, um eine bessere Zusammenarbeit zu ermöglichen. Diese Gruppe von Personen sind somit sowohl Provenance-Urheber als auch -Konsumenten. Änderungen und entsprechende Provenance-Informationen können dabei je nach Bedarf *synchron* oder *asynchron* zur Verfügung gestellt werden. Bei der synchronen Collaboration werden alle Änderungen am System und die dazugehörigen Provenance-Informationen sofort für alle anderen Teilnehmer öffentlich gemacht. Im Bereich der Data Warehouses trifft man z.B. häufig auf die Form der asynchronen Collaboration. Die Bereitstellung von Provenance-Informationen erfolgt dabei meist zu festgelegten Zeitpunkten in bestimmten Zeitintervallen und wird entsprechend dokumentiert.

- **Presentation:** Dieser Bereich der Understandability umfasst solche Provenance-Informationen, die für Personen oder Gruppen außerhalb des eigenen Kompetenzbereiches für ein besseres Verständnis gesammelt werden. Der Name leitet sich dabei aus dem steigenden Bedarf einer geeigneten Präsentation bzw. Visualisierung ab, die sogar interaktiv erfolgen kann. Verschiedene Möglichkeiten der Visualisierung können nämlich auf unterschiedliche Weise zum Verständnis beitragen.

- **Attribution:** Von *Attribution* spricht man, sofern die Provenance-Informationen sich nur auf den Urheber in einem Produktionsprozess beschränken. Provenance-Informationen werden unter anderem dafür genutzt, das Urheberrecht oder den Besitz von Daten festzustellen, um z.B. deren Zitierung zu ermöglichen oder die für fehlerhafte Daten verantwortlichen Personen haftbar zu machen. Gerade im Bereich der Kommunikation sind solche Provenance-Informationen wertvoll. Urheberrechtsinformationen können z.B. durch User-Management explizit erhoben werden, sind oftmals aber auch implizit in den vorliegenden Daten erhalten (z.B. wenn ein Zitat bekannt ist, aber nicht als solches angegeben wurde).

Reproducibility

Die natürlichste Bestrebung zur Sammlung von Provenance-Informationen stellt jedoch der Anwendungsfall der *Reproducibility* dar. Reproduzierbarkeit beschreibt in diesem Fall das Vermögen, mit gleichem Ausgangsmaterial und identischen Methoden auch das gleiche gewünschte Ergebnis zu erhalten. Dies kann des Produktes wegen, oder aber zur Verifizierung bzw. Validierung präsentierter Ergebnisse erfolgen. Auch bei der Reproducibility unterscheidet man folgende zwei Unterklassen.

- **Recall:** Provenance-Informationen gehören zur Unterklasse des *Recall*, sofern Provenance-Produzent als auch -Konsument ein und dieselbe Entität darstellen. Die gesammelten Provenance-Informationen haben demnach den Sinn einer Erinnerungsstütze für einen Produktionsprozess. Diese Informationen beschreiben z.B. ausgeführte Aktionen, deren Auswirkungen und vorgenommene Anpassungen bis hin zu einem gewünschten Endprodukt. Provenance-Informationen können somit auch die nicht zum Ziel führenden Aktionen enthalten, welche korrigiert werden mussten. Ein Beispiel wäre die Auflistung verschieden gesetzter Parameter und die resultierenden Veränderungen im Testablauf eines Programms.
- **Replication:** Bei der *Replication* geht es nur darum, das gewünschte Ergebnis bestmöglich replizieren zu können. Im Gegensatz zum Recall beschreiben die Provenance-Informationen nur die wirklich notwendigen und sinnvollen Aktionen und deren Auswirkungen. Die Zielgruppe sind dabei meist Personen desselben Fachbereich, die solche Informationen wie eine beschreibende Anleitung interpretieren können. Zu beachten ist aber, dass die Provenance-Informationen auch in einem Umfang gesammelt werden müssen, der eine anschließende Interpretation überhaupt ermöglicht.

Quality

Der letzte Punkt, für den Provenance-Informationen genutzt werden können, ist die Sicherung oder Verbesserung der Qualität des Endproduktes (*Data Quality*) oder des entsprechenden Produktionsprozesses (*Process Quality*). Eine Verbesserung des Produktionsprozesses wird dabei meist von Experten vorgenommen. Eine Verbesserung des Endproduktes hingegen kann durch einzelne Individuen vorgenommen werden, da Provenance-Informationen spezifische Qualitätsprobleme sichtbar machen können. Die Sicherung und Verbesserung der Qualität ist zwar ein wichtiger Anwendungsfall von Provenance, dieser findet im weiteren Verlauf dieser Arbeit aber keine weitere Beachtung und wird deswegen nicht weiter ausgeführt.

2.3. Data Provenance

Im Bereich der Data Provenance beschreiben die gesammelten Provenance-Informationen, wie in Kapitel 2.2.2 bereits erwähnt, nur einzelne Datenelemente. Übliche Fragestellungen, die sich im Bereich der Data Provenance dabei herausgebildet haben, sind in Tabelle 2.1 zusammen dargestellt. Da die Darstellung der Datensätze in dieser Arbeit auf dem relationalen Datenbankmodell basiert, sind eine Anfrage Q, eine Datenbankinstanz D und das Anfrageergebnis Q(D) auch in dem entsprechenden Datenmodell angegeben. Die Data Provenance erfasst diesbezüglich den Zusammenhang zwischen der Datenbankinstanz D und dem Anfrageergebnis Q(D) auf der Ebene einzelner Datenelemente. Man unterscheidet dabei zwei grundlegende Typen. Der erste Typ ist die Provenance von Existing Results, also Data Provenance von Datenelementen, die auch im Ergebnis Q(D) enthalten sind. Der zweite Typ ist dagegen die Provenance von Missing Results, die sich mit Datenelementen befasst, welche aus irgendeinem Grund nicht im Ergebnis Q(D) vorkommen. Typische Fragestellungen der Provenance für Existing Results sind die

in Tabelle 2.1 erwähnten Fragestellungen ***Why?***, ***How?*** und ***Where?***. Die Provenance von Missing Results beantwortet dabei immer nur die Fragestellung ***Why-not?***, die an dieser Stelle nur der Vollständigkeit halber erwähnt wird. In Abschnitt 2.3.1 wird die Data Lineage als wichtiger Vorreiter der Data Provenance erläutert. Abschnitt 2.3.2 beschäftigt sich dann mit der Why-Provenance, gefolgt von Abschnitt 2.3.3, der die How-Provenance erläutert. Abschließend beleuchtet Abschnitt 2.3.4 die Where-Provenance.

Data Provenance-Typ	beantwortete Fragestellung
Where	Wo kommt das Datenelement her?
Why	Warum haben wir dieses Datenelement erhalten?
How	Wie ist dieses Datenelement zu Stande gekommen?
Why-not	Warum fehlt ein bestimmtes Datenelement im Ergebnis?

Tabelle 2.1.: Fragestellungen der Data-Provenance

2.3.1. Data Lineage

In diesem Abschnitt soll die *Data Lineage* (kurz: Lineage) als Vorgänger der Why-Provenance auf Basis von [CCT09] präsentiert werden. "*The lineage of an output record is based on identifying a subset of input records relevant to the output record. Intuitively, an input record is relevant to an output record if it contributed to the existence of that output record*" [CCT09]. Frei übersetzt und mit vorher eingeführter Symbolik erklärt ist die Lineage für jeden Eintrag des Ergebnisses Q(D) eine Teilmenge von D, die nur solche Einträge enthält, die auch an der Produktion der Einträge im Ergebnis Q(D) beteiligt waren. Formal ist die Lineage aber wie folgt definiert:

Definition 2.8: (*Lineage für einen beliebigen relationalen Operator*, nach [CCT09] Definition 2.2 auf Basis von [CWW00] Definition 4.1) Die *Lineage* eines Eintrags $t \in Op(R_1, \ldots, R_n)$ bei einem beliebigen relationalen Operator Op über die Relationen $R_1, \ldots, R_n$ ist eine Sequenz $R'_1, \ldots, R'_n$ von Teilmengen $R'_i \subseteq R_i$ und es gilt:

1. $Op(R'_1, \ldots, R'_n) = \{t\}$
2. $1 \leq i \leq n$ und $\forall t_i \in R'_i$ ist $Op(R'_1, \ldots, R'_{i-1}, \{t\}, \ldots, R'_n) = \emptyset$
3. $R'_1, \ldots, R'_n$ ist maximale Untermenge von $R_1, \ldots, R_n,$ die 1. und 2. erfüllen.

Betrachten wir dazu einmal das in Abschnitt 1.3 eingeführte Beispiel, spezieller die Anfrage 1.1, bei der nur solche Einträge der NOTEN-Tabelle selektiert werden, die eine Note von 1.7 enthalten. Das Ergebnis dieser Anfrage ist dann die folgende Relation:

	kurs_nr	student_id	semester	note
X_1	001	1	SS 16	1.7
X_2	002	5	WS 15/16	1.7

Tabelle 2.2.: Ergebnistabelle der Anfrage 1.1

Man beachte, dass die Tupel der Lineage über die im Beispiel eingeführten Tupelidentifikatoren eindeutig referenzierbar sind. Die Lineage für das erste Tupel der Ergebnistabelle 2.2 X_1 enthält somit die QuelltupelID N_1. Die Lineage für die zweite Zeile des Ergebnisses mit dem Tupelidentifikator X_2 enthält dagegen die QuelltupelID N_8. Komplizierter wird die Darstellung der Lineage bei der Anfrage 1.2, die das folgende Ergebnis, dargestellt in Tabelle 2.3, liefert:

	studiengang
Y_1	Informatik
Y_2	Mathematik

Tabelle 2.3.: Ergebnistabelle der Anfrage 1.2

Die Lineage für das erste Tupel der Ergebnistabelle 2.3 mit der ID Y_1 enhält somit die Tupelidentifikatoren S_1, S_2, S_4. Für die zweite Zeile mit der ID Y_2 ist Lineage durch S_3, S_5 charakterisiert. Die Lineage beider Ergebnisrelationen wird in der Tabelle 2.4 noch einmal dargestellt:

	Data Lineage
X_1	N_1
X_2	N_8
Y_1	S_1,S_2,S_4
Y_2	S_3,S_5

Tabelle 2.4.: Data Lineage vom Ergebnis der Anfragen 1.1 und 1.2

Aus der Notation der Lineage ist nun aber nicht erkennbar, dass S_1, S_2 oder S_4 jeweils ausreichend sind, um den Eintrag im Ergebnis mit der ID Y_1 zu erzeugen. Auch ist nicht ersichtlich, dass sowohl S_3 als auch S_5 für die Produktion von Y_2 jeweils ausreichend sind. Die Lineage enthält somit für jeden Eintrag im Ergebnis nur einen einzelnen Zeugen, wobei ein Zeuge wie folgt definiert ist:

Definition 2.9: (*Zeuge*, nach [CCT09] und [BKT01]) Sei I eine Datenbankinstanz, Q eine Anfrage auf der Datenbankinstanz I und $t \in Q(I)$ ein Tupel des Ergebnisses. Eine Instanz $I' \subseteq I$ ist ein Zeuge für t, falls $t \in Q(I')$.

I ist an dieser Stelle anstatt D als Symbol für die Datenbankinstanz gewählt worden, da D' in Abschnitt 1.2 bereits für den anonymisierten Datensatz steht.

Die Notation der Lineage erlaubt also keine Rückschlüsse auf einzelne oder minimale Zeugen und speichert nur die Menge aller relevanten Einträge als einen möglichen Zeugen ab. Eine mögliche Lösung für diese Problematik bietet die Notation der Why-Provenance, die in Abschnitt 2.3.2 vorgestellt und definiert wird.

2.3.2. Why-Provenance

In diesem Abschnitt wird die *Why-Provenance* auf Basis von [CCT09] präsentiert. Durch die historische Entwicklung lief auch die Erforschung der *Why-Provenance* in [WS97] und [CW00] anfänglich noch unter dem Begriff der Data Lineage ab. Die erste spezifische Definition der Why-Provenance als Zeugenbasis erfolgte dann in [BKT01]. Die bereits vorgestellte Notation der Lineage erfasst bereits alle Tupel der Eingabe, die maßgeblich an der Produktion eines Ergebnistupels beteiligt waren, ist aber nicht in der Lage, die einzelnen Zeugen zu unterscheiden, geschweige denn die minimalen Zeugen zu ermitteln. Für eine genaue Antwort auf die Fragestellung ***Why?*** wäre das ermitteln aller Zeugen, auch Zeugenmenge genannt, von Vorteil. Die Zeugenmenge wird dabei wie folgt definiert:

Definition 2.10: (*Zeugenmenge*, nach [CCT09] und [BKT01]) Die Menge aller Zeugen (*Zeugenmenge*) für ein Tupel t, entstanden aus der Datenbankinstanz I und der Anfrage Q, ist definiert als: $Wit(Q, I, t) = \{J \subseteq I \mid t \in Q(J)\}$.

Mit der Zeugenmenge haben wir zwar eine sinnvolle Antwort auf das Why, jedoch ensteht auch ein Problem durch den vorher definierten Begriff des Zeugen: die obere Grenze zur Formulierung eines

Zeugen liegt bei der Datenbankinstanz I selbst. Die Zeugenmenge umfasst somit auch alle Zeugen, die noch irrelevante Tupel enthalten. Ein Problem ist das deshalb, weil dies zu einer exponentiellen Größe der Zeugenmenge führen kann, selbst wenn I endlich ist. Buneman et al. [BKT01] definieren die Why-Provenance deshalb nur als eine bestimmte Teilmenge besagter Zeugenmenge: die Zeugenbasis.

Definition 2.11: (*Zeugenbasis* bzw. *Why-Provenance*, nach [CCT09] und [BKT01]) Sei Q eine SPJRU-Anfrage, I eine Datenbankinstanz und t ein Tupel aus $Q(I)$ dann ist die *Zeugenbasis* $Why(Q, I, t)$ wie folgt definiert:

$$Why(\{t\}, I, \{u\}) = \begin{cases} \{\emptyset\}, & \text{falls } (t = u) \\ \emptyset, & \text{sonst.} \end{cases}$$

$$Why(R, I, t) = \begin{cases} \{\{(R, t)\}\}, & \text{falls } (t \in R(I)) \\ \emptyset, & \text{sonst.} \end{cases}$$

$$Why(\sigma_\theta(Q), I, t) = \begin{cases} Why(Q, I, t), & \text{falls } \theta(t) \\ \emptyset, & \text{sonst.} \end{cases}$$

$$Why(\pi_U(Q), I, t) = \cup\{Why(Q, I, t) \mid u \in Q(I), t = u[U]\}$$

$$Why(\rho_{A \mapsto B}(Q), I, t) = Why(Q, I, t[B \mapsto A])$$

$$Why(Q_1 \bowtie Q_2, I, t) = Why(Q_1, I, t[U_1]) \uplus^* Why(Q_2, I, t[U_2])$$

$$Why(Q_1 \cup Q_2, I, t) = Why(Q_1, I, t) \cup Why(Q_2, I, t)$$

$^*\uplus$ ist dabei die paarweise Vereinigung zweier Mengen mit $S \uplus \perp = \perp \uplus S = \perp$ und $S \uplus T = \{s \cup t \mid s \in S, t \in T\}$ sonst.

Durch Beachtung der Syntax von Q werden also nur noch solche Zeugen ermittelt, deren Größe auch Q entspricht. In [CCT09] werden die Zeugen der Zeugenbasis auch Proof-Witness genannt. Jeder Zeuge innerhalb der Zeugenbasis beschreibt nun exakt die Ausgabe und schließt dabei alle irrelevanten Tupel aus. Die Problematik der exponentiellen Größe ist damit auch gelöst, denn die Zeugenbasis kann im Vergleich zur Zeugenmenge sehr viel kleiner ausfallen. In [BKT01] werden die Begriffe Why-Provenance und Zeugenbasis synonym verwendet. Man beachte außerdem, dass die Zeugenbasis als eine Menge von Mengen dargestellt wird.

Betrachten wir nun wieder die Anfrageergebnisse in Tabelle 2.2 und 2.3. Man beachte, dass wir die Notation der Zeugenbasis von $\{\{(R, t)\}\}$ auf $\{\{t\}\}$ reduzieren, da wieder die eindeutigen Tupelidentifikatoren des Beispiels genutzt werden. Die Zeugenbasis für das Ergebnistupel mit der ID X_1 ist $\{\{N_1\}\}$ und die Zeugenbasis für das zweite Ergebnistupel mit der ID X_2 ist $\{\{N_8\}\}$. Diese sehen der Lineage zunächst noch ähnlich. Die Zeugenbasis für das Ergebnistupel Y_1 ist hingegen $\{\{S_1\}\{S_2\}\{S_4\}\}$ und für das Ergebnistupel Y_2 ist die Zeugenbasis $\{\{S_3\}\{S_5\}\}$. Aus diesen wird im Gegensatz zur Lineage deutlich sichtbar, dass die einzelnen Quelltupel S_x jeweils eigene Zeugen und somit zur Produktion des jeweiligen Ergebnistupels ausreichend sind. Tabelle 2.5 zeigt noch einmal die jeweilige Zeugenbasis und stellt auch noch einmal die Lineage zum Vergleich dar.

	Data Lineage	**Why-Provenance**
X_1	N_1	$\{\{N_1\}\}$
X_2	N_8	$\{\{N_8\}\}$
Y_1	S_1, S_2, S_4	$\{\{S_1\}\{S_2\}\{S_4\}\}$
Y_2	S_3, S_5	$\{\{S_3\}\{S_5\}\}$

Tabelle 2.5.: Data Lineage und Why-Provenance von den Ergebnissen der Anfragen 1.1 und 1.2

Ein Problem besteht allerdings. Da die Zeugenbasis an die Struktur der Anfrage gebunden ist, kann diese trotz äquivalenter Anfragen unterschiedlich ausfallen. Invariant ist dagegen die Definition der Why-Provenance als minimale Zeugenbasis. Eine Menge $s \in S$ wird als *minimales Element* einer Menge S verstanden, wenn für jedes $s' \in S$ mit $s' \subseteq s$ gilt, dass $s' = s$ ist. Die minimale Zeugenbasis sei dann wie folgt definiert:

Definition 2.12: (*minimale Zeugenbasis*, nach [CCT09]) Sei Q eine SPJRU-Anfrage, I eine Datenbankinstanz und t ein Tupel aus $Q(I)$. Die minimale Zeugenbasis für $t \in Q(I)$ ist dabei $MWhy(Q, I, t)$, die Menge aller minimalen Elemente von $Why(Q, I, t)$ mit: $MWhy(Q, I, t) = \{J \in Why(Q, I, t) \mid J \text{ ist minimal in } Why(Q, I, t)\}$.

Im Gegensatz zur Zeugenbasis, die an die Struktur einer Anfrage gebunden ist, ist die minimale Zeugenbasis nur an die Semantik der Anfrage gebunden. Die minimale Zeugenbasis ist bei äquivalenten SPJRU-Anfragen Q und Q' auf einer Datenbankinstanz I für jedes Ergebnistupel t somit gleich: $MWhy(Q, I, t) = MWhy(Q', I, t)$. Für SPJRU-Anfragen gilt außerdem, dass sich die Lineage und die minimale Zeugenbasis aus der Why-Provenance (Zeugenbasis) berechnen lassen. Lineage und Why-Provenance (Zeugenbasis) lassen sich hingegen nicht aus der minimalen Zeugenbasis berechnen. Die Definition der Why-Provenance als Zeugenbasis ist für die Beantwortung des ***Why?*** zwar durchaus sinnvoll, will man aber genau wissen wie ein Ergebnistupel aus den Zeugen abgeleitet wurde, so benötigt man die noch genauere Beschreibung der How-Provenance, die in Abschnitt 2.3.3 präsentiert wird.

2.3.3. How-Provenance

Die bereits vorgestellte Why-Provenance erfasst also die Zeugenbasis für ein bestimmtes Ergebnistupel, nicht aber zusätzliche Informationen darüber, wie ein Tupel genau zu Stande kam. Eine Lösung dafür stellt die Notation der How-Provenance dar, deren Grundstein in [GKT07] gelegt wurde. Die How-Provenance soll an dieser Stelle aber auf Basis von [CCT09] vorgestellt werden. Die Definition der How-Provenance basiert auf zwei Standardtransformationen von Quelltupeln: entweder werden sie verbunden, oder aber mittels einer Projektion zusammengeführt. Entsteht ein Ergebnistupel durch die Projektion auf einen Attributwert, der sowohl im Quelltupel t_1 als auch im Quelltupel t_2 vorkommt, so wird dies durch einen Ausdruck der Form $t_1 + t_2$ kenntlich gemacht. Ein Ergebnistupel, dass durch den Verbund des Quelltupels t_3 mit dem Quelltupel t_4 einer anderen Relation entsteht, wird durch den Ausdruck $t_1 \cdot t_2$ beschrieben. Green et al. [GKT07] stellten dabei fest, dass diese Art von Ausdrücken, die beschreiben, wie Ergebnistupel mittels + und $\cdot$ entstehen, Polynome eines kommutativen Semirings $(K, 0, 1, +, \cdot)$ sind.

Betrachten wir für diese Art der Provenance die Ergebnisse der Anfragen 1.1 und 1.2 in den Tabellen 2.2 und 2.3, so ergibt sich die in Tabelle 2.6 dargestellte How-Provenance. Die Data Lineage und Why-Provenance der Ergebnistupel sind für einen Vergleich mit aufgezeigt:

	Data Lineage	**Why-Provenance**	**How-Provenance**
X_1	N_1	$\{\{N_1\}\}$	N_1
X_2	N_8	$\{\{N_8\}\}$	N_8
Y_1	S_1, S_2, S_4	$\{\{S_1\}\{S_2\}\{S_4\}\}$	$S_1 + S_2 + S_4$
Y_2	S_3, S_5	$\{\{S_3\}\{S_5\}\}$	$S_3 + S_5$

Tabelle 2.6.: Data Lineage, Why- und How-Provenance von den Ergebnissen der Anfragen 1.1 und 1.2

Auch bei der How-Provenance lassen sich die einzelnen Zeugen unterscheiden, allerdings mittels +. Ein wirklicher Unterschied zu den anderen Formulierungen ist bis jetzt jedoch nicht sichtbar. Der eigentliche Vorteil der How-Provenance gegenüber der Why-Provenance zeigt sich am nachfolgend präsentierten Ergebnis der Anfrage 1.3, das in Tabelle 2.7 gezeigt ist.

	student_id	lastname	firstname	studiengang	kurs_nr	semester	note
Z_1	3	Altmann	Anne	Mathematik	002	WS 15/16	1.0

Tabelle 2.7.: Ergebnistabelle der Anfrage 1.3

Die How-Provenance für das Ergebnistupel Z_1 der Tabelle 2.7 ist dann das Polynom $S_3 \cdot N_6$, welches verdeutlicht, dass das Ergebnis durch den Verbund der beiden Tupel entstanden ist. Die Why-Provenance hingegen enthält nur die Information, dass S_3 und N_6 zusammen für die Produktion des Ergebnistupels notwendig sind, also einen Zeugen der Zeugenbasis darstellt. Tabelle 2.8 zeigt für dasselbe Ergebnistupel Z_1 auch noch einmal die Lineage und die Why-Provenance:

	Data Lineage	Why-Provenance	How-Provenance
Z_1	S_3, N_6	$\{\{S_3, N_6\}\}$	$S_3 \cdot N_6$

Tabelle 2.8.: Data Lineage, Why- und How-Provenance vom Ergebnis der Anfrage 1.3

An diesem Beispiel erkennt man gut die Probleme und Vorteile der jeweiligen Provenance-Typen. Die Data Lineage zeigt die relevanten Tupel, enthält aber keine Information darüber, ob wirklich S_3 und N_6 gebraucht werden oder ob eines der beiden Quelltupel ausreichend ist. Die Why-Provenance zeigt dann, dass sowohl S_3 als auch N_6 notwendig sind, da sie zusammen in einem Zeugen der Zeugenbasis stehen. Die How-Provenance zeigt dann als einziges, dass das Ergebnistupel durch den Verbund von S_3 und N_6 entstanden ist, worüber die Why-Provenance und Data Lineage jeweils keine Auskunft geben. Es gibt jedoch noch eine dritte Form der Provenance: die Where-Provenance, die im anschließenden Abschnitt 2.3.4 erläutert wird.

2.3.4. Where-Provenance

Ist im Gegensatz zur genauen Entwicklung (How) oder der Zeugenbasis (Why) nur der Ursprungsort der Ergebnistupel verlangt, so findet die Where-Provenance Anwendung, die in diesem Abschnitt auf Grundlage von [CCT09] vorgestellt wird. Die Definition der Where-Provenance ist vielseitig. So gibt es drei verschiedene Arten, die den verschiedenen Granularitäten der Beschreibung von Datensätzen entspringen. Ist die genaue Zelle eines Tupels gefragt, so ist die ursprüngliche Definition in [BKT01] und [CCT09] gemeint. Die Where-Provenance mit der Granularität von Zellen wurde dabei in [BKT01] eingeführt. Mit dieser Where-Provenance wird im Gegensatz zur How- und Why-Provenance die exakte Herkunft eines Attributwertes im Ergebnis beschrieben, also von wo genau dieser kopiert wurde. Diese Art der Where-Provenance beschreibt somit die genaue Beziehung zwischen den Orten der Ein- und Ausgabe. Buneman et al. [BKT01] nutzen dafür Anmerkungen zum Ursprungsort in der Form (R, t_i, A_j), wobei R für die entsprechende Relation, t_i für das i-te Tupel der Relation R und A_j für das j-te Attribut steht. Diese Anmerkung beschreibt für einen Attributwert im Ergebnis genau die Zellen der Ausgangsrelation, von der dieser Wert kopiert wurde. In [BKT01] und [CCT09] wird die genaue Propagierung solcher Anmerkungen beschrieben. Dabei wird jede Zelle der Ausgangsrelation mit einer entsprechenden Annotation versehen, die entsprechend in das Ergebnis übernommen wird und somit auf den originalen Standort verweist. Da diese Form der Where-Provenance jedoch durch die Menge an notwendigen Annotationen für eine Anwendung auf großen Datensätzen zu umfangreich ist, wird dieser Typ an dieser Stelle nicht weiter vertieft.

Die Darstellung der Where-Provenance kann aber auch ressourcensparender durch alternative Definitionen mit gröberer Granularität erfolgen, wie sie auch in den Vorarbeiten und verwandten Themen ([Sva16], [Aug17], [Sch20], [Aug20]) am Lehrstuhl für Datenbanken & Informationssysteme der Universität Rostock verwendet wurden. Eine davon ist die *tupelorientierte Where-Provenance*. Die zellorientierte Where-Provenance wird entsprechend von der Form (R, t_i, A_j) auf die Form (R, t_i) reduziert, die nicht mehr für

alle Attribute, sonder für jedes Tupel definiert wird. Unter der Annahme, dass alle Tupelidentifikatoren t_i in der Datenbankinstanz I nur einmal vorkommen, also unabhängig vom Relationennamen sind, reicht t_i zur Beschreibung der Where-Provenance aus und beschreibt die genauen Quelltupel, aus dem das Ergebnis kopiert wurde. Im Folgenden soll diese Form der Where-Provenance als Where_t bezeichnet werden. Die zweite Alternative ist die von der tupelorientierten Where-Provenance (R, t_i) noch gröber formulierte *relationenorientierte Where-Provenance* der Form (R). An dieser Stelle ist nur noch die Speicherung der Relationennamen als Ursprungsort für jedes Ergebnistupel notwendig. Diese Form der Where-Provenance wird im Folgenden als Where_r bezeichnet. Ergänzen wir nun die Provenance-Tabellen 2.6 und 2.8 um Where_t und Where_r und entfernen die Lineage, so erhalten wir die Zusammenfassung aller betrachteter Data Provenance-Darstellungen in Tabelle 2.9 (ohne Data Lineage).

	Why-Provenance	**How-Provenance**	**Where$_t$**	**Where$_r$**
X_1	$\{\{N_1\}\}$	N_1	$\{N_1\}$	NOTEN
X_2	$\{\{N_8\}\}$	N_8	$\{N_8\}$	NOTEN
Y_1	$\{\{S_1\}\{S_2\}\{S_4\}\}$	$S_1 + S_2 + S_4$	$\{S_1, S_2, S_4\}$	STUDIERENDE
Y_2	$\{\{S_3\}\{S_5\}\}$	$S_3 + S_5$	$\{S_3, S_5\}$	STUDIERENDE
Z_1	$\{\{S_3, N_6\}\}$	$S_3 \cdot N_6$	$\{S_3, N_6\}$	STUDIERENDE, NOTEN

Tabelle 2.9.: Why-, How- und Where-Provenance vom Ergebnis der Anfragen 1.1, 1.2 und 1.3

Where_t stellt also eine Zeugenmenge ähnlich der Why-Provenance oder auch Data Lineage dar. Mit dem Zusatz, dass wirklich nur solche Zeugen in der Zeugenliste aufgenommen werden, aus denen das Ergebnis auch kopiert werden kann. Ein Unterschied macht sich bemerkbar, sofern die Tupel einer mittleren Tabelle für den Verbund zweier weiterer Tabellen benötigt werden, die aber nicht im Ergebnis erscheinen - in diesem Fall würden die Tupel der mittleren Tabelle in der Why-Provenance vorkommen, in Where_t jedoch nicht. Where_r hingegen zeigt nur noch die Namen der Relationen, aus denen das Ergebnis kopiert werden kann. Where_r lässt sich somit aus den anderen Provenance-Typen herleiten. Auch Where_t hat im Endeffekt weniger Informationsgehalt als die Why- oder How-Provenance. Mit diesen Informationen lässt sich eine Hierarchie der Data Provenance-Typen basierend auf deren Informationsgehalt erstellen [Aug20].

2.4. Privacy

Privacy oder auch Datenschutz sind geläufige Begriffe, deren Auslegungen und Interpretationen oftmals auseinander gehen (siehe [Sch20] Seite 55). Nach der europäischen Datenschutz-Grundverordnung (*DSGVO*[1] umfasst der Begriff Datenschutz die "Grundrechte und Grundfreiheiten natürlicher Personen und insbesondere deren Recht auf Schutz personenbezogener Daten", sowie "Vorschriften zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten und [Vorschriften] zum freien Verkehr solcher Daten". Diese Vorschriften beziehen sich dabei auf sämtliche Vorgänge, angefangen bei der Erhebung bis hin zur Vernichtung bzw. Löschung eines Datenelements. Alleine aus dieser einen Beschreibung ist klar zu erkennen, dass die Umsetzung und Definition des Konzeptes Datenschutz weitaus schwieriger ist, als man es oftmals annimmt. Im Bereich der personenbezogenen Daten ist eine erste Maßnahme deshalb oftmals das Entfernen eindeutiger personenbezogener Attribute wie z.B. Name und Vorname, die im Endeffekt nichts anderes als eine identifizierende Attributmenge darstellen:

Definition 2.14: (*Identifizierende Attributmenge*, nach [HSS18]) Sei R ein Relationenschema, dann ist eine Menge $K := \{B_1, \ldots, B_n\} \subseteq R$ eine *identifizierende Attributmenge*, wenn für jede Relation $r(R)$ gilt:

[1] https://eur-lex.europa.eu/legal-content/DE/TXT/HTML/?uri=CELEX:32016R0679

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

Ein *Schlüssel* ist dann eine bezüglich $\subseteq$ minimale identifizierende Attributmenge.

Soll ein Tupel in einem relationalen Datensatz also nicht eindeutig identifizierbar sein, so ist das Entfernen solcher Attribute von Bedeutung, die in den minimalen identifizierenden Attributmengen vorkommen. Dass das jedoch für eine grundlegende Privacy nicht immer ausreicht, beweist das Beispiel von [Swe00]:

Die *National Association of Health Data Organizations* hatte darin mitgeteilt, dass Daten von Krankenhauspatienten in bis zu 17 Staaten der USA, mit Attributen wie Postleitzahl (PLZ), Geburtsdatum, Geschlecht, Ethnizität, Diagnose, Medikation und Art der Behandlung, erhoben wurden, die durch das Fehlen eindeutig personenbezogerner Attribute, wie z.B. Name und Vorname, als anonym gewertet wurden. *Latanya Sweeney* machte jedoch deutlich, dass auch einfache demographische Merkmale oftmals wieder zur Identifizierung von Personen führen können, sofern sie mit anderen Datensätzen kombiniert werden. Eine gekaufte Liste mit expliziten Daten über die Wähler von Massachusetts bildete mit dem medizinischen Datensatz die in Abbildung 2.4 gezeigte, kritische Schnittmenge.

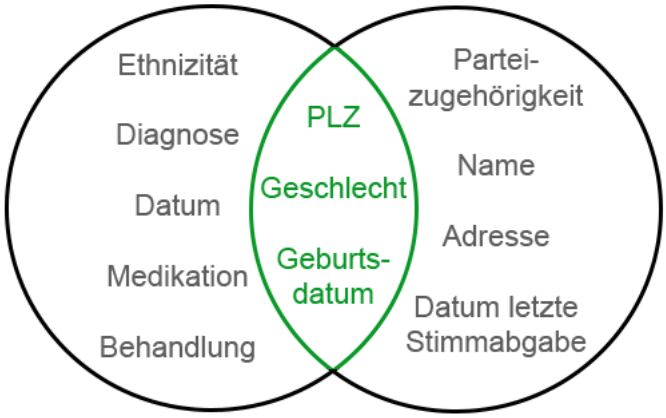

Abbildung 2.4.: Schnittmenge der Datensätze nach [Swe00]

Mittels dieser sich überschneidenden Attribute (PLZ, Geschlecht, Geburtsdatum) ist nämlich eine Zusammenführung der Datensätze möglich, die in mindestens einem Fall eine Person eindeutig identifizieren und dieser die entsprechenden sensiblen Attribute, z.B. die Diagnose und Medikation, zuordnen kann. Die Kombination aus Postleitzahl, Geschlecht und Geburtsdatum ist zwar keine identifizierende Attributmenge, jedoch ein Quasi-Identifikator. Nach [Dal86] und [Swe00] ist eine Menge von Attributen, die in Kombination mit externen Datensätzen eine Person eindeutig identifizieren kann, als ein solcher Quasi-Identifikator definiert. Somit ist die Betrachung möglicher Quasi-Identifikatoren ebenfalls eine Notwendigkeit zur Erfüllung des Datenschutzes. Um die Anonymität eines Datensatzes auch hinsichtlich möglicher Quasi-Identifikatoren evaluieren zu können, werden im nächsten Abschnitt 2.4.1 zwei geläufige Anonymitätsmaße vorgestellt.

2.4.1. Anonymitätsmaße

Zur Evaluierung der Anonymität eines Datensatzes werden in diesem Abschnitt zwei geläufige Anonymitätsmaße vorgestellt: die *k-Anonymität* und die *l-Diversität*. Begonnen wird hierbei mit der Erläuterung der k-Anonymität.

k-Anonymität

Die *k-Anonymität* nach [Swe02] basiert auf dem grundlegenden Modell des k-mappings. Demnach sollen sich Einträge veröffentlichter Datensätze mittels externer Quellen unter Betrachtung von Quasi-Identifiern immer nur auf k Tupel gleichzeitig abbilden lassen. Die k-Anonymität wurde dazu entwickelt, den Aufwand zur Überprüfung jedes Eintrags mittels externer Daten durch ein groß gewähltes k so groß wie möglich zu gestalten, um Originaldaten entsprechend zu schützen.

Definition 2.15: (*k-Anonymität*, nach [Swe02]) Sei $R = (A_1, \ldots, A_n)$ eine Relation und $QI_R \subset R$ ein Quasi-Identifikator in R. R genügt dann der k-Anonymität für QI_R genau dann, wenn auftretende Wertekombinationen für die Attribute $R[QI_R]$ auch mindestens k mal auftauchen.

Tabelle 2.10 zeigt eine beispielhafte k-Anonymität der Wertigkeit 4 für den Quasi-Identifikator QI_R = {**kurs_nr, semester**}. Externe Datensätze mit den Attributen des Quasi-Identifikators können dann zwar immer noch auf den anonymisierten Datensatz abgebildet werden, allerdings wird jede mögliche Attributwert-Kombination von **kurs_nr** und **semester** auf 4 Einträge gleichzeitig abgebildet. Die Erfülltheit einer 4-Anonymität bezüglich QI_R wird dabei durch |`NOTEN`(001, SS 16)| = 4 und |`NOTEN`(002, WS 15/16)| = 4 bestätigt.

	kurs_nr	**student_id**	**semester**	**note**
N_1	001	x	SS 16	1.7
N_2	001	x	SS 16	1.3
N_3	001	x	SS 16	2.3
N_4	001	x	SS 16	3.3
N_5	002	x	WS 15/16	3.0
N_6	002	x	WS 15/16	1.0
N_7	002	x	WS 15/16	2.7
N_8	002	x	WS 15/16	1.7

Tabelle 2.10.: `NOTEN`-Tabelle mit k-Anonymität = 4 bezüglich QI_R = {kurs_nr, semester}

Erfüllt eine Tabelle die k-Anonymität, so kann ein Außenstehender, der nur die Werte des entsprechenden Quasi-Identifikators kennt, nur mit einer Wahrscheinlichkeit von 1/k ein Tupel korrekt identifizieren [LLV07]. Trotz positiver Vorstellung hat die k-Anonymität auch einige Schwächen. Würde derselbe Datensatz ohne eine Note aber mit der expliziten **student_id** an anderer Stelle veröffentlicht werden, um z.B. zu zeigen, welche Studenten an welchen Prüfungen teilgenommen haben, so ließe sich über die Reihenfolge der Datensätze trotzdem ein Rückschluss auf die Noten einiger Studenten ziehen. Ein weiteres Problem ist, dass nicht immer alle Attribute einer Relation als Quasi-Identifikator betrachtet werden, da dies mit steigender Attributmenge auch mit erhöhtem Aufwand verbunden ist. So wäre der Quasi-Identifikator QI'_R = {**kurs_nr, semester, note**}, oder dessen zweielementige Teilmengen mit dem Attribut note bei einer Kombination mit einem anderen Datensatz dafür verantwortlich, Studenten wieder eindeutig identifizieren zu können. Auch eine mögliche Homogenität von Attributen stellt bei der k-Anonymität ein weiteres Problem dar. Wären alle Noten, z.B. für QI_R = {**kurs_nr, semester**}, also eine bestimmte Kursnummer und ein Semester gleich, so wäre eine eindeutige Zuordnung und somit Identifizierung möglich. Um Attacken, die auf der Ausnutzung von Hintergrundwissen oder mangelnder Diversität basieren, zu unterbinden, wurde die l-Diversität eingeführt.

l-Diversität

Die *l-Diversität* (nach [MKGV07]), eine Weiterentwicklung der k-Anonymität, unterbindet die vorher genannten Attacken, die bei besagter k-Anonymität noch möglich sind. Da diese einfacher an einem

Beispiel erklärt ist, wird an dieser Stelle auf eine formale Definition verzichtet. Siehe dazu eine modifizierte NOTEN-Tabelle, dargestellt in Tabelle 2.11.

	kurs_nr	**student_id**	**semester**	**note**
N_1	001	*	SS 16	1.3
N_2	001	*	SS 16	1.3
N_3	001	*	SS 16	1.3
N_4	001	*	SS 16	1.3
N_5	002	*	WS 15/16	3.0
N_6	002	*	WS 15/16	1.0
N_7	002	*	WS 15/16	2.7
N_8	002	*	WS 15/16	1.7

Tabelle 2.11.: abgewandelte NOTEN-Tabelle mit k-Anonymität = 4 bezüglich QI_R = {kurs_nr, semester}

An dieser abgewandelten Tabelle ist erkennbar, dass das sensitive Attribut **note** nicht ausreichend geschützt ist, da alle Einträge mit den Werten **kurs_nr** = 001 und **semester** = SS 16 denselben Wert für **note** = 1.3 enthalten. Ein genauer Rückschluss auf die Originaldaten ist durch die k-Anonymität zwar nicht möglich, jedoch ist dieser durch die gleichen Werte für das Attribut **note** auch überhaupt nicht notwendig. Da für die Wertekombination NOTEN(001, SS 16) des Quasi-Identifikators die Anzahl verschiedener Attributwerte bei eins liegt, entspricht der in Tabelle 2.11 dargestellte Datensatz nur einer 1-Diversität und ist somit nicht anonym, bzw. nicht gegen Angriffe geschützt. Hingegen ist der eigentliche Beispieldatensatz, siehe Tabelle 2.10, mit einer 4-Diversität entsprechend vor Angriffen geschützt, die auf die Ausnutzung einer mangelhaften Diversität abzielen.

Doch welche groben Ansätze zur Anonymisierung eines Datensatzes mit Blick auf die k-Anonymität oder l-Diversität stehen denn nun zur Verfügung bzw. sind bereits erforscht?

Da die k-Anonymität in den wenigsten Fällen in einem Datensatz direkt erfüllt ist, gibt es verschiedene Methoden zur Modifikation des Datensatzes, um eine bestimmte k-Anonymität zu erreichen. Die folgenden Abschnitte befassen sich demnach mit den Methoden der Generalisierung und des Unterdrückens in Abschnitt 2.4.2, der Differential Privacy in Abschnitt 2.4.3, der Permutation in Abschnitt 2.4.4, mit intensionalen Antworten in Abschnitt 2.4.5 und schlussendlich mit der Methode des Slicens in Abschnitt 2.4.6. Die überblickhafte Vorstellung dieser Methoden ist dabei an vorherige wissenschafftliche Arbeiten der Universität Rostock angelehnt, siehe [Sch20] und [Kle20].

2.4.2. Generalisieren und Unterdrücken

Generalisieren und Unterdrücken sind zwei unterschiedliche Methoden zum Erreichen einer k-Anonymität. Diese sollen aufgrund ihrer Einfachheit in diesem Abschnitt zusammen vorgestellt werden.

Unterdrücken bezeichnet dabei nichts anderes als das Weglassen von Tupeln t in einer Relation R, die die besagte k-Anonymität verletzen. Zwar kann mittels Unterdrückung eine bestimmte k-Anonymität erreicht werden, jedoch können Ergebnisse des Originaldatensatzes und des k-anonymen Datensatzes voneinander abweichen. Anstatt aber ganze Tupel zu verbergen, bietet es sich an, nur einzelne sensitive Attribute zu verbergen. Damit wird die Anzahl der Tupel aussagekräftig gehalten. Da diese Methode aber nicht immer zu einer gewünschten k-Anonymität führt und gleichzeitig die l-Diversität verschlechtern kann, bietet sich eher eine Generalisierung der sensitiven Attributwerte an [LLV07].

Beim *Generalisieren* werden einzelne Attributwerte entlang einer selbst definierten Hierarchie durch weniger präzise Werte ersetzt, mit der Bedingung, dass die Semantik dabei erhalten bleibt. Ein Datum kann sich so z.B. auf einen Monat, ein Jahr oder ein anderes Intervall wie z.B. Quartale generalisieren lassen.

Für numerische Daten bieten sich ebenfalls Intervalle an. Eine andere Möglichkeit der Generalisierung ist das Ersetzen einzelner Zeichen eines Wortes durch ein anonymes Token. Ein Beispiel hierfür wäre der Austausch der letzten Ziffern einer Postleitzahl durch das $*$-Symbol. In Tabelle 2.10 wurde eine entsprechende Generalisierung mittels $*$-Symbol für die student_id z.B. schon vorgenommen. Nach [Swe02] ist die Generalisierung eine Funktion der Form $f : A_i \Rightarrow A_i + 1$, die es erlaubt, die Ausgangsmenge A_0 endlich oft bis hin zu einem maximal generalisierten Element A_n zu generalisieren. Die Hierarchie, der dabei zu folgen ist, wird in sogenannten Generalisierungshierarchien bzw. Konzepthierarchien festgelegt, die sich auch als Bäume darstellen lassen. Eine mögliche Generalisierungshierarchie für das Attribut **note** ist in Form eines Baums in Abbildung 2.5 dargestellt.

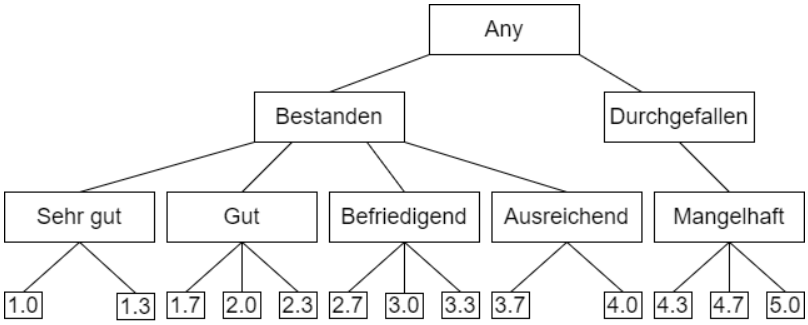

Abbildung 2.5.: Konzepthierarchie für das Attribut **note** der `NOTEN`-Tabelle

2.4.3. Differential Privacy

Bei der Differential Privacy handelt es sich um eine Methode, die darauf abzielt, die Aussagekraft eines Datensatzes für statistische Analysen zu erhalten, jedoch einzelne Individuen innerhalb der Datensätze zu schützen. Differential Privacy sorgt kurz erklärt also dafür, dass das Ergebnis einer beliebigen Analyse durch das Hinzufügen oder Entfernen eines einzelnen Tupels in einem Datensatz nicht beeinflusst wird [Dwo08]. Grundlegend für die Eingabe der Differential Privacy ist also ein Datensatz, der mittels einer randomisierenden Funktion für die Veröffentlichung zu einer verrauschten Ausgabe führt.

Definition 2.16, (nach [Dwo06]): Seien zwei Datenbanken D_1 und D_2 gegeben, wobei D_1 sich von D_2 nur um ± 1 Tupel unterscheidet. Eine randomisierende Funktion K erfüllt die ϵ-*differential Privacy* für D_1, D_2 und alle $S \subseteq \mathit{Range}(K)$ genau dann wenn:

$$\Pr[K(\mathrm{D}_1) \in S] \leq e^{\epsilon} \times \Pr[K(\mathrm{D}_2) \in S]$$

[Sch20] formuliert diese Definition für das relationale Datenbankmodell wie folgt um:

Definition 2.17, (nach [Dwo06] und [Sch20]): Eine randomisierende Anfragefunktion K bietet ϵ-*differential Privacy*, wenn für eine Relation R und ein neues Tupel t, sowie für alle $S \subseteq \mathit{Range}(K)$ gilt:

$$\Pr[K(\mathrm{R}) \in S] \leq e^{\epsilon} \times \Pr[K(\mathrm{R} \cup \{\mathrm{t}\}) \in S]$$

In dieser Definition steckt, dass K mit einer hohen Wahrscheinlichkeit sowohl für R als auch $\mathrm{R} \cup \{\mathrm{t}\}$ dasselbe Ergebnis bzw. nur ein minimal verändertes Ergebnis liefert, sollte ein Tupel aus der Relation R entfernt oder hinzugefügt werden. ϵ ist dabei ein Maß für den Datenschutz. Je kleiner ϵ gewählt

wird, desto eher ist eine Gleichheit der Wahrscheinlichkeiten gefordert und einzelne Individuen sind geschützt. Soll hingegen eine Gruppe von Individuen geschützt werden, so muss ϵ mit der Gruppengröße c multipliziert werden, was bei steigenden Gruppengrößen zu höheren Datenschutzrisiken führt [Dwo06]. ϵ-differential Privacy wird in [Dwo06] durch das Hinzufügen von zufälligem Rauschen erreicht. Es können dafür verschiedene Arten von zufälligem Rauschen genutzt werden. [DKM+06] nennt unter anderem Gaußsches-, Exponentielles- oder Poisson-Rauschen.

2.4.4. Permutieren

Um einer möglichen *Unsorted-Matching-Attacke* entgegenzuwirken, einem Angriff, der die Reihenfolge der Speicherung zweier Datensätze zur Rekombination originaler Daten ausnutzt, kann die *Permutation* genutzt werden. Hierbei soll möglichst fixpunktfrei die Reihenfolge der Tupel vertauscht werden, sodass an den Stellen sensitiver Attribute nun entsprechend andere Werte stehen als zuvor. Man beachte, dass bei einer Häufung bestimmter Werte wie z.B in Tabelle 2.11 für das Attribut **note** eine Permutation nicht immer zielführend ist, da nur wenige fixpunktfreie Abbildungen existieren. Da die Erforschung des Prinzips der Permutation weit in der Vergangenheit liegt und keine genauen Definitionen genannt werden, sei an dieser Stelle keine spezielle Quelle dazu benannt. Es sei aber erwähnt, dass moderne Datenbanksysteme meist schon eine Möglichkeit der Permutation mittels einer implementierten random()-Funktion anbieten.

2.4.5. Intensionale Antworten

Die Thematik intensionaler Antworten fand ihren Ursprung in verschiedenen Studien zur künstlichen Intelligenz, da Menschen, sofern sie mit einer bestimmten Frage konfrontiert werden, nicht nur die extensionale Antwort, also die genaue Antwort auf eine Frage, sondern oftmals auch intensionale Antworten geben, die die genaue Antwort unterstützen oder abstrahieren [Mot94]. Diese Art der Forschung geht dabei bis in die 1980er zurück [Web85]. [Mot94] war einer der ersten Überblicksartikel zur Thematik intensionaler Antworten und möglicher Ansätze, der neben einer klaren Definition für intensionale Antworten auch eine Klassifizierung solcher Antworten vorstellte. Motro [Mot94] definiert eine intensionale Antwort im Gegensatz zu konventionellen (extensionalen) Antworten als eine knappe Beschreibung oder als Menge nützlicher Aussagen, die eine Antwort betreffen. Als einfaches Beispiel nennt Motro dabei die Ausführung einer Anfrage an eine Datenbank. Eine solche Anfrage ist eine rein intensionale Information zu einem bestimmten Ergebnis. Allein aus diesem Beispiel ist allerdings ersichtlich, dass intensionale Informationen nicht immer intensionale Ergebnisse produzieren müssen, sondern z.B. auch einen extensionalen Datensatz (zumindest in Kombination mit den originalen Daten). Intensionale Antworten sind nach [Mot94] aber erstrebenswerte Antworten, die zusätzlich zur extensionalen Antwort (auf deren Ausgabe Datenbankmanagementsysteme üblicherweise ausgelegt sind) erzeugt und bereitgestellt werden sollten, um z.B. ein Ergebnis zu verdeutlichen und den Informationsgehalt des Gesamtergebnisses zu erhöhen. Zu solchen intensionalen Informationen zählt Motro im relationalen Datenbankmodell unter anderem Integritätsbedingungen, die Definitionen der Basisrelationen und die Sichtdefinitionen. Während also die Daten einer Relation extensional vorliegen, ist das Relationenschema beispielsweise eine intensionale Information. In logischen und objektorientierten Datenbankmodellen haben diese intensionalen Informationen entsprechend andere Formate, siehe [Mot94]. Das Ziel der Forschung an intensionalen Antworten ist nach [Mot94] die abstraktere Antwortmöglichkeit auf Anfragen, wobei intensionale Antworten bis dahin in verschiedenen Definitionen, Frameworks und Berechnungsmöglichkeiten umgesetzt wurden. Motro fasst diese Ansätze zusammen und erstellte auf deren Basis [Mot94] eine einheitliche Definition und Klassifizierung für solche intensionalen Antworten. Diese Klassifizierung soll auch in dieser Arbeit

Verwendung finden und wird deshalb entsprechend vorgestellt, allerdings nur für das relationale Datenbankmodell.

Klassifizierung intensionaler Antworten nach Motro [Mot94]:

- Datenmodell und intensionale Informationen werden verwendet
- Inklusion extensionaler Informationen innerhalb der intensionalen Antwort
- Vollständigkeit der intensionalen Charakterisierung
- Unabhängigkeit von extensionalen Informationen

Der erste Punkt klassifiziert dabei die intensionalen Antworten nach dem verwendeten Datenmodell und den zusätzlich bereitgestellten intensionalen Informationen, wie z.B. Integritätsbedingungen oder die Definition von Basisrelationen. Da wir uns im relationalen Datenbankmodell befinden ist diese Form der Klassifizierung im weiteren Verlauf jedoch überflüssig. Der zweite Punkt unterscheidet die Typen intensionaler Antworten nun in solche,aw die reine (*pure*) intensionale Informationen darstellen oder die mit extensionalen Informationen gemischt sind (*mixed*). Punkt drei unterscheidet nun derartige intensionalen Antworten, die nur eine partielle (*partial*) Darstellung der eigentlichen Antwort darstellen bzw. nur zusätzliche Informationen liefern, von denen, die eine extensionale Antwort vollständig (*complete*) charakterisieren. Der vierte und letzte Punkt klassifiziert die intensionalen Antworten auf Basis der Grundlage der Berechnung solcher Antworten. Basiert die intensionale Antwort auf reinen intensionalen Informationen, so ist diese von der Extension unabhängig (*independent*). Werden hingegen extensionale Informationen zur Generierung der intensionalen Antwort verwendet, so spricht man von einer abhängigen (*dependent*) intensionalen Antwort. Unter Betrachtung der Punkte zwei bis vier ergeben sich die sechs folgenden Klassen (unter der Annahme, dass mixed und independent unvereinbar sind):

1. pure-complete-independent
2. pure-complete-dependent
3. pure-partial-independent
4. pure-partial-dependent
5. mixed-complete-dependent
6. mixed-partial-dependent

Die in [Mot94] betrachteten Ansätze und Frameworks, die das relationale Datenmodell verwenden, fallen dabei in die Klassen 3 und 4, was in sofern ein Problem darstellt, da die intensionale Antwort, die in Abschnitt 1.2 gefordert ist, einer vollständigen (complete) Charakterisierung der extensionalen Daten entsprechen soll und nicht einer partiellen. Für diese Arbeit werden entsprechend relevantere Ansätze in Kapitel 3 in Abschnitt 3.3 und 3.3.1 präsentiert.

2.4.6. Slicing

Slicing wird von [Sch20] zwar nicht explizit als Lösung für das PPP genannt, ist jedoch eine Methode, die den Informationsgehalt eines Datensatzes besser erhält als die Methode der Generalisierung, weswegen das Slicen an dieser Stelle auf Basis von [GH15] trotzdem vorgestellt wird.

Die Anonymisierung mittels Slicing wurde aus dem Problem heraus entwickelt, dass die Generalisierung den Informationsgehalt eines Datensatzes zu sehr mindern kann [LLZM12]. Slicing bedient sich dabei (ähnlich zur Permutation) dem Prinzip der Vertauschung, allerdings einer spaltenweisen Vertauschung innerhalb festgelegter Zeilen- und Spaltengrupppen.

Im ersten Schritt erfolgt eine horizontale Partitionierung (tuple partition [LLZM12]) des Datensatzes in n disjunktive Teilrelationen, sogenannte Buckets. Selektionsbedingungen, die für diese Partitionierung

verantwortlich sind, müssen dabei geschickt gewählt werden, um ein Ungleichgewicht der Buckets zu vermeiden [GH15]. Im zweiten Schritt (attribute partition [LLZM12]) werden mittels Projektion die Teilrelationen weiter in einzelne Spalten bzw. Spaltenkombinationen unterteilt, mit der Bedingung, dass diese keine identifizierende Attributmenge enthalten und in Betrachtung der gewählten Analysefunktionen entsprechende Korrelationen bewahren [GH15]. Der dritte Schritt stellt dann die eigentliche Anonymisierung mittels Permutation innerhalb der gewählten Bereiche dar. Entsprechend müssen diese anonymisierten Attributpartitionen innerhalb einer Teilrelation wieder verbunden werden. Anschließend werden auch die Teilrelationen wieder miteinander verbunden. Dies kann z.B. mittels einer Nummerierung der einzelnen Tupel nach der horizontalen Partitionierung geschehen [GH15]. Slicing bietet sich durch den Erhalt exakter Werte deshalb für aggregierende Verfahren an [Kle20].

3. State of the Art

Nach der Präsentation der Grundlagen erfolgt nun die Präsentation des aktuellen Stands der Forschung. Abschnitt 3.1 geht dabei nochmal auf Artikel der letzten Jahre ein, die sich mit den möglichen Konflikten der Verbindung von Provenance und Privacy beschäftigen. Abschnitt 3.2 begrenzt die Betrachtungen weiter auf jene Arbeiten, die als Provenance-Klasse speziell die Data Provenance untersucht haben. Hierbei wird vor allem auf mögliche Privacy-Risiken aufmerksam gemacht, die in Verbindung mit einer Speicherung von Data Provenance auftreten können. Dabei wird vor allem die Invertierung einzelner Data Provenance-Typen untersucht. Abschnitt 3.3 beschreibt anschließend ein frühes Konzept zur Formulierung von intensionalen Antworten aus extensionalen Datensätzen von [YP99]. Der Ansatz von Yoon et al. nutzt dafür die in den Grundlagen bereits vorgestellte Methode der Generalisierung aus. Den Abschluss bildet Abschnitt 3.3.1, in dem eine theoretische Erweiterung von [Sva16] vorgestellt wird, die den in Abschnitt 3.3 beschriebenen Ansatz zur Generierung intensionaler Antworten für Provenance-Anfragen konzipiert.

3.1. Provenance und Privacy

Provenance und Privacy führen gezwungenermaßen zu einem Konflikt, was allein der Kontroversität beider Konztepte geschuldet ist. Provenance ziehlt darauf ab, so viel zusätzliche Informationen wie möglich zu einem Objekt, Workflow, oder Datenelement bereitzustellen, mit deren Hilfe sich bestimmte Fragen beantworten lassen, die für ein erhöhtes Verständnis, verbesserte Qualität aber auch für die Reproduzierbarkeit sorgen können (siehe Abschnitt 2.2). Privacy hingegen zielt darauf ab, die Veröffentlichung nicht direkt notwendiger Datensätze zu vermeiden. Das Ziel ist hierbei der Schutz von einzelnen Individuen und die Bewahrung geistigen Eigentums (siehe Abschnitt 2.4).

Der Konflikt dieser beiden Konzepte ist dabei alles andere als neu. [BGH+06] beschäftigte sich bereits 2006 mit den Problemen, die bei einer automatisierten Sammlung von Provenance-Informationen entstehen. Neben Herausforderungen wie die Verarbeitung verschiedener Granularitäten von Provenance-Informationen und dem Prunen, also der Ermittelung und Löschung nicht notwendiger Informationen, wird dort auch ein ganzes Kapitel entstehenden Problemen im Bereich Privacy und Security gewidmet. Ein Problem ist unter anderem die Sammlung von Provenance-Informationen über die Benutzer von Services. Ein bekanntes Beispiel hierfür ist das Speichern der Besuchszeiten von Benutzern einer Website, die Speicherung des Absenders gesendeter Mails, dessen Inhalt, Lesezeitpunkt etc. Nach der DSGVO ist die Erhebung solcher personenbezogenen Daten ohne Einwilligung zwar nicht erlaubt, aber auch bei Erlaubnis zur Speicherung besteht die Gefahr eines Datenlecks. Gerade bei der Auswertung solcher Informationen kann es passieren, dass einzelne Individuen, seien es Mitarbeiter, Kunden oder andere eindeutig identifizierbar sind. Als weitere Probleme werden der Speicherort der Provenance-Informationen und die mangelnde Restriktion gesammelter Provenance-Informationen genannt.

Ein Motivator für die Erforschung von Provenance unter Privacy-Aspekten war auch James Cheney. In [Che11] wies er bereits 2011 darauf hin, dass die Erhebung verschiedener Provenance-Informationen zwar

gut untersucht ist, aber eine Betrachtung der Security bzw. Privacy dabei grundlegend vernachlässigt wird. Cheney verweist in diesem Bereich auch wieder auf die in [BGH+06] aufgeworfenen Problematiken. Besonders wird dabei die Veröffentlichung von Provenance-Informationen hervorgehoben. In [Che11] werden anschließend Policies zur Veröffentlichung von Provenance unter Privacy-Aspekten (Disclosure und Obfuscation) und ein resultierendes Framework vorgestellt. Die Untersuchungen von [Che11] und [BGH+06] beziehen sich dabei auf digitale Systeme und fallen somit in den Bereich der Information System Provenance (siehe Abb. 2.2).

Etwas spezifischer sind dagegen die Betrachtungen von Susan B. Davidson et al. in "On Provenance and Privacy" [DKR+11]. So wird die erkannte Problematik aus [BGH+06] und [Che11] nun im Bereich der Workflow-Provenance untersucht. Mit dem Blick auf entsprechende Workflows werden drei Privacy-Aspekte propagiert: *Data, Module und Structural Privacy*. Data Privacy bezeichnet den Schutz von Daten, die zwischen den einzelnen Knoten eines Workflows übertragen werden. Diese Knoten oder auch Module eines Workflows entsprechen dabei immer einer bestimmten Funktion, die für Außenstehende nicht einsehbar ist. Die Module Privacy beschreibt neben fehlender Transparenz der Module ebenfalls, dass eine Ausgabe eines Moduls generell nicht von der Eingabe ableitbar sein darf. Structural Privacy erfordert zudem, dass Personen außerhalb ihrer Befugnis keine Teile der Provenance-Informationen einsehen dürfen, die eventuelle Rückschlüsse auf den Aufbau des Workflows erlauben. In [DKT+11] wird dieselbe Thematik auch noch einmal aufgegriffen.

Die Möglichkeiten zur Kombination von Provenance und Privacy sind so vielfältig wie das Thema Provenance selbst. Provenance-Informationen können unter anderem Aktionen bestimmter Nutzer aufzeichnen, die die Anonymität einzelner Personen verletzen können. [AL12] setzt sich unter anderem mit diesem Problem auseinander und stellt entsprechend ein Schema der Privacy-Preserving-Provenance vor, welches vor unautorisierten Zugriffen schützen und die Anonymität einzelner Nutzer wahren soll. [SBS18] und [CY20] sind als weitere Beispiele genannt, die gesammelte Provenance-Informationen in einem eigenen Provenance-Modell zur Erhöhung bestimmter Privacy-Anforderungen nutzen. Mit der Festlegung auf das relationale Datenbankmodell und die Data Provenance, wie schon in Abschnitt 1.2 angedeutet, werden vorangegangene Forschungsarbeiten mit den Stichworten Provenance und Privacy allerdings übersichtlicher, auch zu sehen an folgendem Abschnitt 3.2.

3.2. Data Provenance und Privacy

In diesem Abschnitt werden nun die für diese Arbeit relevanten Betrachtungen zu Data Provenance und Privacy vorgestellt, inklusive der auftretenden Probleme und vorgeschlagene Lösungsmöglichkeiten. Die Anzahl der Veröffentlichungen, die dieses Thema mit den genannten Eingrenzungen auf Why-, How- und Where-Provenance im relationalen Datenbankmodell behandeln, ist allerdings sehr überschaubar.

Eine der Arbeiten, dessen Fokus auf Privacy-Betrachtungen zur Data Provenance in den Formen Why, How und Where liegt, ist die Bachelorarbeit von Nic Scharlau aus dem Jahr 2020 ([Sch20]). Kapitel 4 behandelt dabei ausführlich die Fragestellungen "*inwieweit Data Provenance und Privacy miteinander einhergehen, ob gewisse Provenance-Techniken und -Anfragen möglicherweise den Datenschutz verletzen und falls ja, welche Lösungsansätze es dafür geben kann*" [Sch20]. Als größtes Problem der Data Provenance wird die mögliche Invertierbarkeit der einzelnen Provenance-Typen untersucht. Für den Test der Provenance-Typen auf Invertierbarkeit wird als Beispielanfrage eine Aggregation verwendet.

An dieser Stelle werden die Ergebnisse am Beispieldatensatz aus Abschnitt 1.3 mittels Anfrage 1.4, die ebenfalls eine Aggregation darstellt, analog zu dem Beispiel in [Sch20] nachvollzogen. Das Ergebnis der Anfrage 1.4 ist dafür in Tabelle 3.1 dargestellt, die entsprechende Provenance findet sich in Tabelle 3.2.

	kurs_nr	**durchschnitt**
A_1	001	2.15
A_2	002	2.1

Tabelle 3.1.: Ergebnis von Anfrage 1.4

	Why-Provenance	**How-Provenance**	**Where**$_t$	**Where**$_r$
A_1	$\{\{N_1, N_2, N_3, N_4\}\}$	$\frac{(N_1 \odot 1.7) \oplus (N_2 \odot 1.3) \oplus (N_3 \odot 2.3) \oplus (N_4 \odot 3.3)}{N_1 \oplus N_2 \oplus N_3 \oplus N_4}$	N_1, N_2, N_3, N_4	NOTEN
A_2	$\{\{N_5, N_6, N_7, N_8\}\}$	$\frac{(N_5 \odot 3.0) \oplus (N_6 \odot 1.0) \oplus (N_7 \odot 2.7) \oplus (N_8 \odot 1.7)}{N_5 \oplus N_6 \oplus N_7 \oplus N_8}$	N_5, N_6, N_7, N_8	NOTEN

Tabelle 3.2.: Provenance des Ergebnisses von Anfrage 1.4

Invertierbarkeit der Where-Provenance Die Where-Provenance stellt nach [Sch20] und [ASH21] nur bedingt ein Privacy-Risiko dar. Und zwar dann, wenn die Provenance-Informationen aus den Originaltupeln gestaltet sind, also nicht nur eine Referenz auf die Originaltupel darstellen. Die in Abschnitt 2.3.4 vorgestellten Formen der Where-Provenance entsprechen aber lediglich solchen Referenzen, deren Notation keine Originaltupelteile beinhaltet. Das eigentliche Problem der Where-Provenance ist jedoch, dass die Provenance-Informationen für die in dieser Arbeit betrachteten Anwendungen in den präsentierten Granularitäten Where_r und Where_t nicht unbedingt ausreichen, um typische Anwendungsfälle der Provenance, wie z.B. Nachvollziehbarkeit und Reproduzierbarkeit, zu gewährleisten. Die betrachteten Darstellungen der Where-Provenance eignen sich also nicht für die Erstellung einer nachvollziehbaren Datenanalyse, entsprechen dafür aber dem Privacy-Konzept.

Invertierbarkeit der Why-Provenance Die Why-Provenance stellt die Zeugenbasis und somit alle Zeugen für ein bestimmtes Ergebnistupel dar. Durch die Angabe der einzelnen Zeugen ohne irrelevante Tupel kann eine Datenanalyse nachvollziehbar und reproduzierbar gestaltet werden. Es bleibt also die Betrachtung der Why-Provenance in Bezug auf die Privacy. Wie auch bei der Where-Provenance stellt die Why-Provenance nur dann ein Privacy-Problem dar, wenn zur Annotation dieses Provenance-Typs Originaltupel anstatt Referenzen verwendet werden. Die in dieser Arbeit betrachtete Form der Why-Provenance beinhaltet aber nicht die Speicherung von Originaltupeln. Die Why-Provenance wird in der Form eindeutiger Tupelidentifikatoren dargestellt, die die jeweiligen Zeugen der Zeugenbasis referenzieren. Ein Privacy-Problem stellt eine Referenz nur im Falle der Aggregatfunktion dar, sofern die Varianz der aggregierten Werte bei 0 liegt und der entsprechende Ausgangswert dadurch sichtbar wird. Einen realen Anwendungsfall stellt dies aber in den meisten Fällen nicht dar, so [Sch20].

Invertierbarkeit der How-Provenance Die How-Provenance liefert eine genaue Berechnungsvorschrift, aus der das Ergebnistupel entstanden ist. Für Selektionen, Projektionen und den natürlichen Verbund reichen dabei "$\cdot$" und "$+$" aus. Im Falle der Aggregation brauchen wir allerdings zusätzlich das Tensorprodukt $\odot$, mit dem Polynome mit einem konkreten Wert verknüpft werden können. Auch wird die Summe $\oplus$ benötigt, die Polynome miteinander verbinden kann [Sch20]. Die How-Provenance für eine Aggregatfunktion ist dann als $\text{AVG(X)} = \frac{\text{SUM(X)}}{\text{COUNT(X)}}$ darstellbar [Sch20]. Die How-Provenance erlaubt von allen betrachteten Typen der Data Provenance die größte Nachvollziehbarkeit und Reproduzierbarkeit, stellt aber auch das größte Privacy-Risiko dar. Am Beispiel der in Tabelle 3.2 gezeigten How-Provenance ist es z.B. möglich, jede einzelne Note der Ausgangstabelle NOTEN durch den Ausdruck der How-Provenance wiederherzustellen. Auch [ASH21] sieht die Notation der How-Provenance kritisch,

da zu viele Informationen verarbeitet werden, die eine Rekonstruktion bzw. Rekonstruktion sensitiver Informationen ermöglichen. Sogar ohne die Bereitstellung von Tupeln des Originaldatensatzes ist die Notation der How-Provenance in Kombination mit einem Anfrageergebnis als Privacy-Risiko einzustufen.

Um diesen entsprechenden Invertierbarkeiten entgegenzuwirken, nennen [Sch20] und [ASH21] die in Abschnitt 2.4 genannten Methoden zur Anonymisierung. Als erster Lösungsansatz wird die Differential Privacy erwähnt, bei der originale Datensätze in dem Maße verrauscht werden, dass das Hinzufügen oder Entfernen eines Tupels in oder aus dem Datensatz keine Auswirkungen auf das Ergebnis einer Analyse hat, um einzelne Individuen zu schützen. Unter erhöhter Schwierigkeit soll dieses Phänomen auch auf Gruppen von Individuen ausweitbar sein. [Sch20] stellt jedoch heraus, dass sich Ergebnisse durch die Anwendung von Differential Privacy nicht mehr validieren lassen. Weitere Lösungsansätze sehen [Sch20] und [ASH21] in den Methoden der Generalisierens oder in der Methode des Unterdrückens. Durch eine entsprechende Verallgemeinerung (Generalisierung) kann den Erhalt der Privacy durch eine ausreichende Minderung des Informationsgehalts hergestellt werden, die nicht unbedingt die Nachvollziehbarkeit verletzen muss. Auch das Verbergen (Unterdrücken) von Tupeln, die ein entsprechendes Anonymitätsmaß verletzen, kann für den Erhalt der Privacy durch eine Minderung des Informationsgehalts sorgen. Allerdings durch eine viel intensivere Minderung, als es die Generalisierung ermöglicht. Um einer Unsorted-Matching-Attacke entgegenzuwirken, reicht nach [Sch20] und [ASH21] eine Permutation bereits anonymisierter Daten aus. Fixpunktfreie Permutationen von sensitiven Attributen erhöhen entsprechend die Privacy, sofern eine gewisse Diversität der Werte vorliegt und die Permutation auf bereits anonymisierten Daten erfolgt. Als letzten Lösungsansatz nennen [Sch20] und [ASH21] den Ansatz der intensionalen Provenance-Antworten. Im Gegensatz zu extensionalen Antworten liefern intensionale Antworten nicht die Daten selbst, sondern nur eine Beschreibung der jeweiligen Daten [Mot94]. Dabei muss unbedingt der Detailgrad solcher intensionalen Antworten beachtet werden, denn eine geforderte Nachvollziehbarkeit und Validierung eines Ergebnisses sollte mittels intensionaler Informationen trotzdem noch möglich sein [Sch20]. Abschnitt 3.3 beschäftigt sich speziell mit dem Thema der intensionalen Antworten und dem aktuellen Stand der Forschung dazu.

Neben bisher besprochenen Privacy-Problemen durch genutzte Provenance-Notationen können Teile der Datenanalyse wie z.B die Auswertungsanfrage oder der Datensatz selbst ein Privacy-Risiko darstellen, sofern diese veröffentlicht werden. Im Konzept-Kapitel 4 werden deshalb zusätzliche Privacy-Betrachtungen zu allen Bestandteilen einer Datenanalyse durchgeführt.

3.3. Intensionale Antworten

An dieser Stelle wird ein Ansatz zur Generierung intensionaler Antworten von [YP99] beschrieben. Entgegengesetzt zu [Mot94] wird dabei ein Ansatz beschrieben, der je nach Ausführung der Klasse 2 (pure-complete-dependent) oder 5 (mixed-complete-dependent) der Klassifizierung nach [Mot94] und somit den gestellten Anforderungen an eine Intensionalisierung entsprechen kann. Yoon et. al. bedienen sich dazu der Methode der Generalisierung auf Basis von Konzepthierarchien. Eine Konzepthierarchie kann wie in Abbildung 3.1 in einem entsprechenden Baum dargestellt werden. Die Wurzel des Baumes stellt dabei die Stufe der größten Verallgemeinerung dar, also auch die Stufe mit dem geringsten Informationsgehalt. Jede weitere Stufe in Richtung der Blätter des Baumes hat entsprechend einen höheren Informationsgehalt. Das Beispiel in Abbildung 3.1 stellt eine mehrstufige Konzepthierarchie dar. In den Blättern sind alle möglichen Werte für eine Note im akademischen System dargestellt. Eine Stufe weiter in Richtung Wurzel werden die Attributwerte der Blätter zu sinnvollen, nicht-überlappenden, monotonen Intervallen generalisiert, die bereits mehrere Möglichkeiten im Attributwert erlauben und deshalb privater sind. Ist

die Zuordnung zu solchen Intervallen immer noch zu spezifisch, lassen sich die Intervallgrenzen entsprechend auf zwei weitere Intervalle generalisieren, die nur noch den Unterschied zwischen bestanden und nicht bestanden darstellen. Ist auch diese Stufe für einen bestimmten Anwendungsfall nicht privat genug, so kann dann auf das allgemeinste Intervall in der Wurzel generalisiert werden, das jeden möglichen Attributwert für das Attribut **note** enthält. Die Wurzel hat somit den geringsten Informationsgehalt aller Stufen, entspricht aber auch dem Konzept der Privacy am meisten.

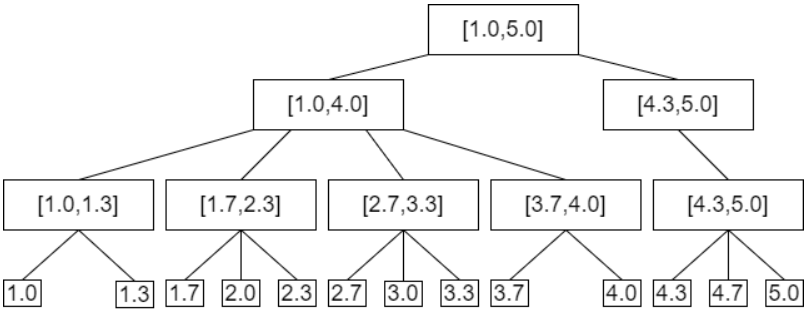

Abbildung 3.1.: Konzepthierarchie Typ 1 für das Attribut **note**

Der Ansatz zur Generierung intensionaler Antworten von [YP99] wird in drei Phasen unterteilt: *Preprocessing, Query Execution und Answer Generation.*

- In der Preprocessing Phase wird für jedes Attribut einer Datenbank eine Konzepthierarchie erstellt. Zu beachten ist dabei, dass sich nicht für jedes Attribut immer eine sinnvolle Hierarchie erstellen lässt. Eine Konzepthierarchie ist demnach nur durch Experten der entsprechenden Domänen erstellbar. Im zweiten Teil der Preprocessing Phase wird eine globale Sicht in Form einer virtuellen Hierarchie erstellt, die die Konzepte verschiedener Hierarchien in Beziehung setzt.

- In der Query Execution Phase werden drei Schritte ausgeführt. Schritt eins ist die extensionale Verarbeitung der gegebenen Anfrage. In Schritt zwei werden nicht relevante Attribute für die Antwort der Anfrage aus dem Ergebnis der Anfrage entfernt - auch solche Attribute, für die keine Konzepthierarchie erstellt werden konnte. In Schritt drei werden unter den verbleibenden Attributen dann jene ausgewählt, die generalisiert werden sollen. Dafür können verschiedene Strategien gewählt werden, um z.B. Korrelationen zu erhalten.

- In der Answer Generation Phase werden die relevanten Attribute nun immer mindestens einmal generalisiert. Die Generalisierung wird anschließend so lange wiederholt, bis eine weitere Generalisierung zur Vereinheitlichung aller Werte führen würde. Für jede dieser Generalisierungsphasen können verschiedene Grenzwerte für die Anzahl der Einteilungen und die Abbruchbedingung gesetzt werden. Im zweiten Schritt dieser Phase wird die intensionale Antwort generiert. Umfasst die Antwort nur ein einzelnes Ergebnis, so kann eine intensionale Antwort als Konjunktion der jeweiligen generalisierten Tupel formuliert werden.

Um den gesamten Prozess noch einmal zu verdeutlichen wird für die folgende Anfrage "`SELECT note FROM NOTEN`" aus dem Beispieldatensatz die Generierung einer intensionalen Antwort vorgenommen. Tabelle 3.3 zeigt das in der zweiten Phase erstellte Anfrageergebnis.

- In der Preprocessing Phase wird für das relevante Attribut **note** nun eine Konzepthierarchie erstellt, z.B. wie in Abbildung 3.1. Für die Attribute **kurs_nr**, **student_id** und **semester** ist keine sinnvolle Konzepthierarchie definiert und sie müssten entsprechend im Anfrageergebnis entfernt werden. Anschließend wird eine globale Sicht definiert, die den Zusammenhang der Attributwerte mit den jeweiligen Stufen darstellt.

	note
N_1	1.7
N_2	1.3
N_3	2.3
N_4	3.3
N_5	3.0
N_6	1.0
N_7	2.7
N_8	1.7

Tabelle 3.3.: Ergebnis von Anfrage "`SELECT note FROM NOTEN`"

- In der Query Execution Phase wird nun das in Tabelle 3.3 gezeigte Ergebnis generiert. Unnötige Attribute müssen durch die Natur der Projektion nicht mehr entfernt werden. Zum Schluss wird das Attribut **note** für die Generalisierung ausgewählt.

- In der Answer Generation Phase werden die in Phase zwei gekennzeichneten Attribute nun bis zur vorletzten Stufe, also eine Stufe vor der Wurzel, generalisiert. Entsprechend ist dann das Ergebnis in Tabelle 3.4 dargestellt, das schon eine intensionale Antwort darstellt. Durch eine Konjunktion der Ergebnistupel ergibt sich dann die pure intensionale Antwort: "Die Noten der `NOTEN`-Tabelle liegen alle im Intervall [1.0,4.0]".

	note
N_1	[1.0,4.0]
N_2	[1.0,4.0]
N_3	[1.0,4.0]
N_4	[1.0,4.0]
N_5	[1.0,4.0]
N_6	[1.0,4.0]
N_7	[1.0,4.0]
N_8	[1.0,4.0]

Tabelle 3.4.: Generalisiertes Ergebnis von Anfrage "`SELECT note FROM NOTEN`"

3.3.1. Intensionale Provenance-Antworten

In Abschnitt 1.2 wurde für den Weg zu einer privaten Validierung eines Ergebnisses Q(D) als Ausgangspunkt vorgeschlagen. Um keine Rückschlüsse auf den eigentlichen Datensatz D zu erhalten, wurde deswegen eine intensionale Beantwortung der Provenance-Anfrage in Betracht gezogen. Eine solche Generierung intensionaler Provenance-Antworten beschrieb Jan Svacina [Sva16] in seiner Bachelorarbeit im Jahr 2016, dessen Methode an die von [YP99] angelehnt ist. Die Generierung der intensionalen Provenance-Antwort erfolgt entsprechend auch mit der Generalisierung des extensionalen Ergebnisses entlang eigens definierter Konzepthierarchien. Das Konzept zur Generierung einer intensionalen Provenance-Antwort nach [Sva16] erweitert die Anzahl der Schritte dabei auf fünf, wobei der Ablauf im Gesamten aber sehr stark an die Schritte in [YP99] erinnert. Die Schritte zwei und drei werden außerdem parallel ausgeführt. Hier eine kurze Beschreibung der Schritte und darin ausgeführter Aktionen:

- Schritt 1: Erstellen von Konzepthierarchien für jedes Attribut

- Schritt 2: Ausführung der Auswertungsanfrage Q auf dem Datensatz D

- Schritt 3: Provenance-Berechnung für das in Schritt 2 berechnete Ergebnis Q(D)

- Schritt 4: Generierung einer Provenance-Anfrage, dadurch Vorfilterung des Datensatzes D auf den relevanten Teildatensatz **D**, anschließende Generalisierung von **D** zur Formulierung der intensionalen Antwort **D'**
- Schritt 5: Ausgabe der Antwort

Wie auch im Ansatz von [YP99] beginnt der Ansatz von [Sva16] mit der manuellen Erstellung von Konzepthierarchien zur späteren Generalisierung der Attributwerte. [Sva16] erweitert dabei die Methode der Erstellung der Konzepthierarchien von [YP99] so, dass alle Attribute gleich behandelt werden können, egal ob eine sinnvolle Konzepthierarchie erstellt werden kann oder nicht. [Sva16] ordnet die Attribute dabei den folgenden drei Typen zu:

- Typ 1: Attribute mit kategorisierbaren numerischen Werten
- Typ 2: Attribute mit nicht numerischen Werten ohne sinnvolle Hierarchie
- Typ 3: Kategorisierbare Attribute mit nicht numerischen Werten

Typ 1 lässt sich dabei, wie z.B. in Abbildung 3.1, in eine Konzepthierarchie mit sinnvollen Intervallen einteilen. Typ 3 lässt sich ebenfalls leicht in eine Konzepthierarchie einteilen, nur sind die Attributwerte der Stufen nicht mehr zwingend numerisch, siehe z.B. Abbildung 3.2.

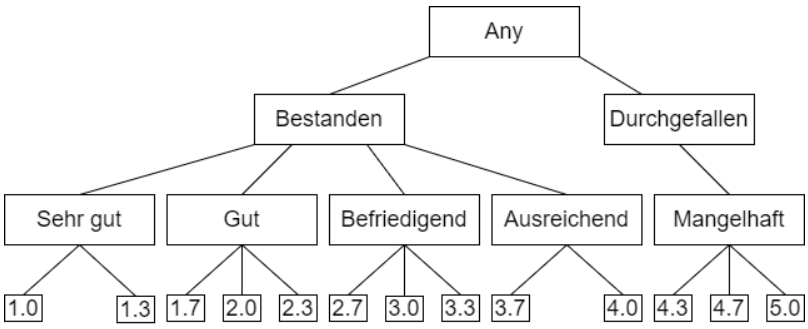

Abbildung 3.2.: Konzepthierarchie Typ 3 für das Attribut **note**

Im Gegensatz zum Ansatz von [YP99] werden für Attribute von Typ 2, die in [YP99] entfernt werden, an dieser Stelle generische Hierarchien erstellt, die die aussagekräftigen Attributwerte erst auf die gleiche Anzahl an zufälligen Platzhaltern zur Wahrung von Korrelationen generalisiert und anschließend in einer gemeinsamen Wurzel zusammenfasst, die ein vom Nutzer definiertes Anonymitätstoken enthält, wie z.B. `any`, $*$ oder x.

Für die Umsetzung der Auswertungsanfrage mit zusätzlicher Notation der Provenance bedient sich der Ansatz nach Svacina einem Tool, welches in Schritt 3 auch dazu verwendet wird eine entsprechende Provenance-Anfrage zu stellen. Da eine mögliche Notation der vorgestellten Provenance-Typen aber auch toolunabhängig formuliert werden kann, wird an dieser Stelle auf die weitere Präsentation des Tools verzichtet. Zu den Schritte der Ausführungen von [Sva16] sei noch gesagt, dass es sich hauptsächlich um theoretische Betrachtungen handelt, die in erwähnter Arbeit keine Umsetzung gefunden haben. Trotzdem wird der theoretische Ansatz im weiteren Verlauf dieser Arbeit dennoch zur Generierung intensionaler Provenance-Antworten verwendet und ein entsprechendes Konzept für die toolunabhängige Umsetzung dieses Ansatzes entwickelt.

4. Konzept

An dieser Stelle wird noch einmal der Verlauf einer durch Vereinigung des Privacy-Konzepts mit der Nachvollziehbarkeit der Provenance entstandenen Provenance-unterstützten Datenanalyse mit allen zu untersuchenden Bausteinen wiederholt. Die Abbildungen in diesem Kapitel sind dabei im Gegensatz zu den einstiegsfreundlichen Darstellungen aus Kapitel 1 in Abschnitt 1.2, die noch einzelne Tabellen bzw. Relationen als Pendant für mögliche Datensätze zeigten, abstrakter dargestellt. Diese abstrakten Darstellungen sollen ein Bewusstsein dafür schaffen, dass die mit D annotierten Datensätze mehr darstellen als nur eine einzelne Tabelle bzw. Relation. Abbildung 4.1 zeigt einleitend noch einmal eine konventionelle Datenanalyse inklusive der Nutzung von Data Provenance.

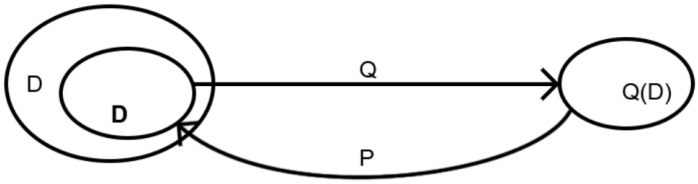

Abbildung 4.1.: Konventionelle Datenanalyse

In "*Provenance und Privacy in ProSA*" ([Sch20]) wird dabei vor allem die Darstellungsform der Data Provenance auf die Privacy untersucht, nicht jedoch die einzelnen anderen Bestandteile einer Datenanalyse. Da zur Privacy-Einhaltung der gesamten Datenanalyse, so wie es das Ziel dieser Arbeit ist, auch die Privacy jedes zu veröffentlichenden Teils garantiert werden muss, werden Privacy-Betrachtungen für die Auswertungsanfrage Q in Abschnitt 4.1, in Abschnitt 4.3 für den Datensatz D bzw. den durch Provenance ermittelten Teildatensatz **D** und für das entstehende Anfrageergebnis Q(D) ebenfalls in Abschnitt 4.1 für die jeweilige Auswertungsanfrage vorgenommen. In genau diesem Aspekt unterscheidet sich diese Arbeit auch von allen anderen Vorarbeiten, in denen entweder ausschließlich die Data Provenance auf Privacy-Risiken untersucht wurde, oder aber Lösungen zur intensionalen Beantwortung von Anfragen bzw. Provenance-Anfragen vorgestellt wurden, um eine Privacy-gerechte Darstellung des relevanten Teildatensatzes **D** und darin enthaltener Daten zu erlauben. Es nützt jedoch nichts, wenn im Kontext einer gesamten Datenanalyse nur die genannten Teile D und Data Provenance der Privacy entsprechen, der Rest einer konventionellen Datenanalyse aber nicht. Stellt die Auswertungsanfrage Q z.B. eine reine Selektion dar - im schlimmsten Fall auf genaue Individuen in einem Datensatz, der eigentlich den Privatheitsansprüchen unterliegt - so kann der relevante Teildatensatz noch so intensional gestaltet und die Provenance-Notation sogar durch leere Referenzen umgesetzt sein, so verletzt das Ergebnis der Anfrage mit allen im Ergebnis selektierten Einträgen und Attributen zu einem bestimmten Individuum die Privacy im höchsten Maß! Der Ansatz aus [Sch20] verfolgte dabei die Motivation, dass zur Nachvollziehbarkeit eines Analyseergebnisses der relevante Teildatensatz **D** und die Auswertungsanfrage Q immer veröffentlicht wird und nur noch die Provenance als mögliches Privacy-Risiko, als eine Möglichkeit zur Regenerierung nicht in **D** bereitgestellter Daten, übrig bleibt.

Um eine gesamte Datenanalyse Privacy-konform zu gestalten, sind entsprechende Betrachtungen zu den einzelnen Bestandteilen im Gesamtablauf einer Datenanalyse genau so wichtig und demzufolge vorzu-

nehmen. Datensatz **D** und Auswertungsanfrage Q stellen bei der Umsetzung einer konventionellen Datenanalyse im Sinne der Informatik immer die Eingabe, das Ergebnis Q(D) mit der Provenance und einer Möglichkeit zur Provenance-Anfrage die Ausgabe dar. Um eine Datenanalyse für die Nachvollziehbarkeit, Reproduzierbarkeit und Verständlichkeit des Ergebnisses Q(D) zu gestalten wird wie schon erwähnt die Notation der Data Provenance genutzt. Eine Umsetzung der Data Provenance auf relationalen Datensätzen ist dabei oftmals in für spezielle Datenbankmanagementsysteme entwickelten Tools realisiert, die den Raum der praktischen Anwendbarkeit um einiges einschränken können. In Abschnitt 4.6 wird deshalb das Konzept einer für das relationale Modell zugeschnittenen, toolunabhängigen, eigens entwickelten Lösung zur Umsetzung einer Data Provenance präsentiert, die zusätzlich noch dazu genutzt wird, die zu veröffentlichenden Datensätze vor Unsorted-Matching-Angriffen zu schützen. Das Privacy-Risiko einer Provenance-Darstellung kann, wie [Sch20] auch schon bemerkte, für verschiedene Provenance-Darstellungen und verschiedene Auswertungsanfragen Q unterschiedlich ausfallen. Umfangreich wird das das Ganze demnach auch noch, weil eine Umsetzung besagter Data Provenance in Abschnitt 4.6 sich nach der Auswertungsanfrage Q richtet. Die resultierende Möglichkeit einer Provenance-Anfrage zur Bestimmung des relevanten Datensatzteils in Abschnitt 4.7 hängt dann von der Umsetzung der Provenance-Notation ab, die wiederum abhängig von der Eingabe der Datenanalyse ist, was die einzelnen Betrachtungen zur Umsetzung einzelner Bausteine immer wieder auf die Form der Auswertungsanfrage zurückführt. Das alleine weist schon auf den Umfang der notwendigen Betrachtungen hin, die bisher nur für eine konventionelle Datenanalyse genannt wurden.

Um eine Veröffentlichung des Datensatzes **D** im Rahmen einer Datenanalyse vermeiden zu können, muss ein anonymisierter Datensatz D' bzw. **D'** bereitgestellt werden. Dessen Präsenz und die resultierenden Anpassungen der einzelnen Schritte einer konventionellen Datenanalyse resultieren dann in einem Konzeptablauf, wie er in Abbildung 4.2 dargestellt ist.

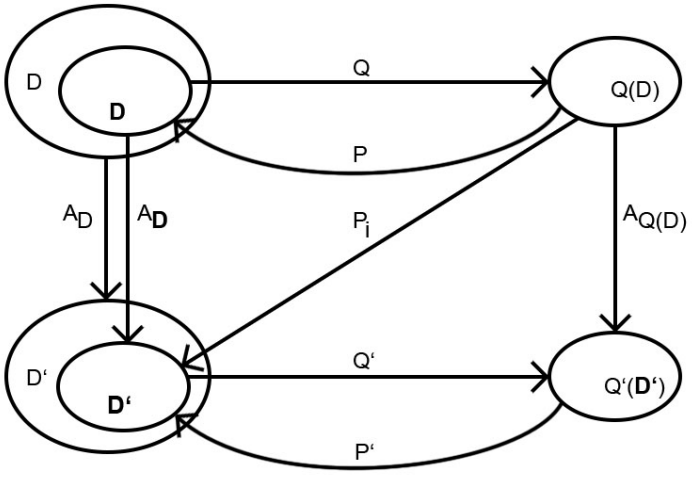

Abbildung 4.2.: Provenance-unterstützte Datenanalyse

Doch welcher Aufwand zur Einhaltung der Nachvollziehbarkeit und Privacy und vor allem zur umsetzbaren Gestaltung einer solchen Analyse entsteht dadurch? Dazu sei gesagt, dass das Konzept eines solchen Ablaufs neu ist und entsprechende Überlegungen zur Umsetzbarkeit und Privacy höchstens für einzelne Bestandteile, dann aber auch in anderen Kontexten vorgenommen wurden, siehe zum Beispiel [Sch20] und [Sva16].

Neben den Bausteinen der konventionellen Datenanalyse ist nun eine zusätzliche Methode zur Anonymisierung des Datensatzes D bzw. dessen relevanten Teils **D** notwendig. Die Auswahl eines solchen Kandidaten für diese Form der Datenanalyse wird dafür in Abschnitt 4.2 vorgenommen, der die in Abschnitt

2.4 vorgestellten möglichen Lösungen im Ablauf einer Provenance-unterstützten Datenanalyse diskutiert. Ein Konzept für eine toolunabhängige Umsetzung, der in diesem Abschnitt als sinnvoll bewerteten Methode für das relationale Datenmodell, wird dann in Abschnitt 4.4 vorgestellt. Da die Anzahl der im Datensatz enthaltenen relevanten Relationen einer Datenanalyse und somit auch die Anzahl vorzunehmender Anonymisierungen aus einer Auswertungsanfrage Q ableitbar sind, wird auch Intensionalisierung für die verschiedenen Möglichkeiten einer Auswertungsanfrage Q untersucht. Auch notwendige resultierende Anpassungen des Datensatzes D werden darin erläutert. Abschnitt 4.5 zeigt dann für alle möglichen Auswertungsanfragen Q, welche Anpassungen die Intensionalisierung auf die Anfrage Q hat, damit eine Ausführung auf generalisierten Attributwerten möglich bleibt.

Da die Veröffentlichung des originalen Datensatzes D oder dessen relevanten Teils **D** in Anbetracht der Privacy durch eine solche Form der Datenanalyse vermieden werden soll, wird der Ansatz von Svacina aus [Sva16] zur intensionalen Beantwortung der Provenance-Anfrage noch einmal aufgegriffen, da diese den Ausgangspunkt der Nachvollziehbarkeit und Validierung auf das Ergebnis Q(D) verschieben kann. Da eine intensionale Provenance-Anfrage immer eine Kombination einer Provenance-Anfrage und einer Methode zur Intensionalisierung ist, wird die Wichtigkeit der Abschnitte 4.2 bzw. 4.4 zur Intensionalisierung, des Abschnitts 4.6 zur Provenance-Notation und des Abschnitts 4.7 zur Formulierung einer Provenance-Anfrage noch einmal deutlich hervorgehoben. Abschnitt 4.8 beschäftigt sich dann mit der möglichen Formulierung einer intensionalen Provenance-Anfrage bzw. der Formulierung dessen Antwort. Durch die möglichen Abfolgen von Intensionalisierung und Provenance-Anfrage entsteht neben dem Ansatz von Svacina, der die Generalisierung einer Provenance-Antwort vornimmt, noch ein weiterer Ansatz. Diese verschiedenen Ansätze resultieren in unterschiedlichen Ausprägungen der Privacy aber auch der Nachvollziehbarkeit und werden demzufolge für die Umsetzung diskutiert. Da eine intensionale Provenance-Anfrage von der Provenance-Anfrage abhängt, deren Formulierung wieder von der Auswertungsanfrage Q abhängt, werden auch an dieser Stelle für die verschiedenen Möglichkeiten einer Auswertungsanfrage Q entsprechende Betrachtungen zur Realisierung einer solchen intensionalen Provenance-Anfrage vorgenommen. Abschnitt 4.9 fasst dann die Erkenntnisse vorheriger Abschnitte zusammen und beschreibt den kompletten Ablauf einer Provenance-unterstützten Datenanalyse für alle umsetzbaren und Privacy-konformen Auswertungsanfragen Q. Begonnen wird in Abschnitt 4.1 mit dem Baustein der Auswertungsfrage, auf den sich alle anderen Teile einer Datenanalyse und resultierende Konzepte und Umsetzungen stützen.

4.1. Auswertungsanfrage Q

Eine Provenance-unterstützte Datenanalyse ist wie eine konventionelle Datenanalyse abhängig vom Umfang und der Form der Eingabe. Die Auswertungsanfrage Q, mit der sich dieser Abschnitt intensiv auseinandersetzt, kann durch die Anzahl verschiedener relationaler Operatoren und Statements jedoch viel zu umfangreich werden. Für eine erste Betrachtung der Formulierung einer Provenance-unterstützten Datenanalyse werden in dieser Arbeit deswegen vorerst nur Standardoperationen betrachtet, um wenigstens grundlegende Formen einer Datenanalyse für eine Provenance-unterstützte Datenanalyse zu untersuchen, gleichzeitig aber den Rahmen durch die Kombinationsmöglichkeiten der einzelnen Operationen nicht zu sprengen. Die Wahl fällt dabei auf die Selektion, Projektion, den natürlichen Verbund und die Aggregation. Die betrachteten Formen der Aggregation umfassen dabei die Funktionen MIN, MAX, SUM, AVG, und COUNT, die einen großen Teil praktischer Anwendungsfälle abdecken. Auch die möglichen Kombinationen der genannten Operationen werden in entsprechenden Anfragen untersucht. Auf die Mengenoperationen Vereinigung, Durchschnitt und Differenz wird an dieser Stelle verzichtet, da diese nur in solchen Anwendungsfällen relevant sind, in denen zwei Relationen mit identischen Schemata

und Wertebereichen zusammen betrachtet werden. Auch lässt sich die Data Provenance für Mengenoperationen nicht zufriedenstellend formulieren, da entweder zu wenig oder zu viele Tupel in der Data Provenance notiert werden. Bei der Differenz ist eine Formulierung der Data Provenance zusätzlich nicht wirklich umsetzbar. Aus demselben Grund entfällt auch die Betrachtung einer Selektion von Ungleichheiten.

An dieser Stelle werden nun alle Möglichkeiten einer Auswertungsanfrage Q, bestehend aus den genannten Operationen aufgezählt, auf Privacy-Risiken untersucht und für eine eindeutige Handhabung in späteren Teilen dieser Arbeit in SQL-Schreibweise formuliert, um einen anfänglichen Rahmen möglicher Eingaben für die neue Form der Datenanalyse anzugeben. Abbildung 4.3 zeigt noch einmal den Teil des Ablaufs einer Provenance-unterstützten Datenanalyse, der hier betrachtet wird.

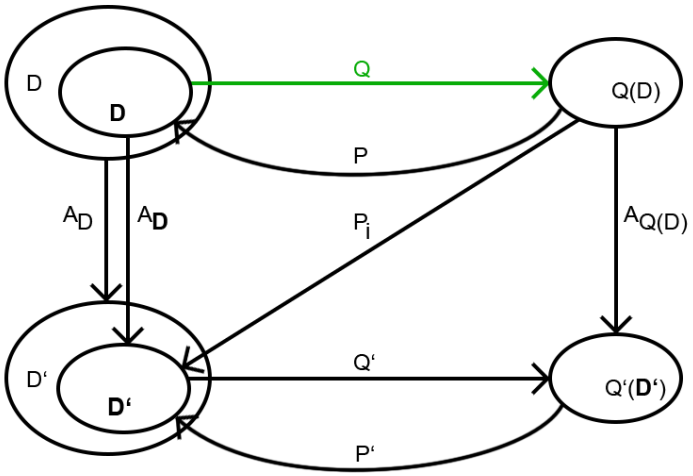

Abbildung 4.3.: Auswertungsanfrage Q

- **Selektion:** Eine Selektion entspricht einem SQL-Ausdruck der Form “`SELECT * FROM tabelle WHERE selektionsbedingung`”. Die Selektionsbedingung entspricht einem Ausdruck der Form $spalte\,\theta\,value$. Die Spalte kann dabei ein beliebiges Attribut der ausgewählten Tabelle sein, θ kann durch einen der Vergleichsoperatoren $<, \leq, =, \geq, >$ und $value$ durch einen bestimmten Attributwert der gewählten Spalte ersetzt werden. Eine solche Selektion liefert als Ergebnis Q(D) immer Tupel des originalen Datensatzes D, was dem Konzept der Privacy entgegensteht, da Datensätze ungefiltert im zu veröffentlichenden Ergebnis vorkommen. Die Umsetzung einer isolierten Selektion entspricht somit nicht dem Privacy-Konzept und kann demnach nicht in einer Provenance-unterstützten Datenanalyse als Eingabe genutzt werden.

- **Projektion:** Eine Projektion ist durch einen SQL-Ausdruck der Form “`SELECT spaltennamen FROM tabelle`” umsetzbar. Auch eine Projektion stellt ein Privacy-Risiko dar, sofern das Ergebnis auf solche Spalten projiziert wird, die Identifikatoren, Quasi-Identifikatoren oder sensitive Attribute enthalten. Eine isolierte Projektion ist demnach nur dann als Eingabe einer Provenance-unterstützten Datenanalyse erlaubt, wenn die Auswahl der Attribute mit dem Konzept der Privacy vereinbar ist. Entsprechende Privacy-Untersuchungen sind unbedingt vom Urheber der Eingabe vorzunehmen! In weiteren Auswertungsanfragen, in denen die Projektion als Bestandteil vorkommt, wird diese Bedingung als erfüllt vorrausgesetzt.

- **Natürlicher Verbund:** Ein natürlicher Verbund kann mit einem SQL-Ausdruck der Form “`SELECT * FROM tabelle1 NATURAL JOIN tabelle2`” ausgeführt werden. Das Ergebnis Q(D) ist dann die Verbindung beider Tabellen, sofern es ein gleichnamiges Attribut mit gleichen Attributwerten (Verbundpartner) gibt. Das Ergebnis eines natürlichen Verbundes stellt die Verbindung von Original-

tupeln zweier Ausgangstabellen dar und verletzt somit ebenfalls die Privacy, in der Annahme, dass der Verbund auch wirklich ein Ergebnis liefert. Ein isolierter natürlicher Verbund ist demnach nicht als Eingabe für eine Provenance-unterstützte Datenanalyse erlaubt.

- **Aggregation:** Eine Aggregation lässt sich durch einen SQL-Ausdruck der Form "`SELECT AGG(spalte) FROM tabelle`" darstellen. Das AGG lässt sich entsprechend durch eines der Schlüsselworte MIN, MAX, SUM, AVG oder COUNT ersetzen, wobei immer nur eine Spalte eines kompatiblen Datentyps ausgewählt werden darf. Die Ausnahme bildet das Schlüsselwort COUNT, welches immer nur in der Form COUNT(*) vorkommen darf. MIN, MAX, SUM, und AVG liefern als Ergebnis den minimalen oder maximalen Attributwert der gewählten Spalte, die Summe aller Attributwerte dieser Spalte oder den durchschnittlichen Attributwert der Spalte als Ergebnis. COUNT liefert im Gegensatz dazu die Anzahl aller Tupel in der gewählten Tabelle zurück. Da die Ergebnisse einer Aggregation einzelne Zahlenwerte darstellen, ist eine mögliche Verletzung der Privacy einer isolierten Aggregation sehr unwahrscheinlich, bei einheitlichen Attributwerten jedoch möglich [Sch20]. Eine isolierte Aggregation kann demnach als Auswertungsanfrage Q in einer Provenance-unterstützten Datenanalyse genutzt werden.

Aus diesen Standardoperationen der relationalen Algebra lassen sich nun weitere Kombinationen als mögliche Eingabe für eine Auswertungsanfrage Q formulieren. Die resultierenden Ausswertungsanfragen werden hierfür entsprechend mit den Anfangsbuchstaben der beinhalteten Operationen abgekürzt. Die einzige Ausnahme bildet dabei der natürliche Verbund, der durch den Buchstaben J (engl. für Join) dargestellt wird.

- **SP:** Eine Selektion in Kombination mit einer Projektion entspricht einem SQL-Ausdruck der Form "`SELECT spaltennamen FROM tabelle WHERE selektionsbedingung`". Das Ergebnis liefert entsprechende projizierte Spalten, die dem Privacy-Konzept entsprechen, sofern die Projektion wie zuvor besprochen auch der Privacy entspricht. Eine Projektion kann demnach in Kombination mit einer Selektion für die Einhaltung der Privacy sorgen. Als Eingabe für eine Provenance-unterstützte Datenanalyse kann eine SP-Anfrage somit genutzt werden.

- **SJ:** Eine Kombination aus Selektion und natürlichem Verbund entspricht einem SQL-Ausdruck der Form "`SELECT * FROM tabelle1 NATURAL JOIN tabelle2 WHERE selektionsbedingung`". Da sowohl Selektion als auch der natürliche Verbund Originaltupel bzw. deren Verbindung im Ergebnis enthalten, steht das Ergebnis dem Prinzip der Privacy entgegen. Eine SJ-Anfrage ist somit nicht als Eingabe für eine Provenance-unterstütze Datenanalyse erlaubt.

- **SA:** Eine Selektion in Kombination mit einer Aggregation entspricht einem SQL-Ausdruck der Form "`SELECT AGG(spalte) FROM tabelle WHERE selektionsbedingung`". Auch hier kann AGG wieder durch ein Schlüsselwort wie bereits bei der einfachen Aggregation ausgetauscht werden. Da die Aggregation in Kombination mit einer Selektion als Ergebnis nur einzelne oder aggregierte Zahlenwerte der selektierten Tupel als Ergebnis liefert, kann demnach eine SA-Anfrage als Eingabe für eine Provenance-unterstützte Datenanalyse genutzt werden. Es ist zu beachten, dass mittels des Attributwertes in der Selektionsbedingung einzelne Teilinformationen reproduziert werden können. Diese sind jedoch bei einer Veröffentlichung von Aggregationsergebnissen in den meisten Fällen sowieso für ein zusätzliches Verständnis angegeben, z.B. die Durchschnittsnote aller Noten des Kurses 001 (`WHERE kurs_nr = 001`).

- **PJ:** Eine Kombination von Projektion und natürlichem Verbund entspricht einem SQL-Ausdruck der Form "`SELECT spaltennamen FROM tabelle1 NATURAL JOIN tabelle2`". Entspricht die Projektion dabei dem Privacy-Konzept, so verletzt auch das Ergebnis, welches aus einzelnen projizie-

renten Spalten des Verbundergebnisses besteht, die Privacy nicht. Eine PJ-Anfrage kann demnach als Eingabe für eine Provenance-unterstützte Datenanalyse gewählt werden.

- **PA:** Bei der Formulierung eines SQL-Ausdrucks einer Projektion und einer Aggregation kommt es ohne Verwendung einer Gruppierung zu einem Problem, denn die Aggregation, wie sie in diesem Abschnitt isoliert dargestellt ist, beinhaltet schon die Projektion auf die aggregierte Spalte. Eine zusätzliche Projektion kann demnach für den betrachteten Anwendungsfall der Aggregation gar nicht formuliert werden. Dieses Problem wird im weiteren Verlauf unter den Begriffen PA-Problem oder ProjAgg-Problem geführt, welches immer Auftritt, sofern eine Projektion in Kombination mit einer Aggregation erfolgen soll. Eine PA-Anfrage ist demnach nicht als Eingabe für eine Provenance-unterstützte Datenanalyse geeignet, weil sie nicht umsetzbar ist.

- **JA:** Ein natürlicher Verbund in Verbindung mit einer Aggregation entspricht einem SQL-Ausdruck der Form "`SELECT AGG(spalte) FROM tabelle1 NATURAL JOIN tabelle2`". Das Ergebnis stellt hierbei einen einzelnen oder aggregierten Zahlenwert der durch vorhandene Verbundpartner entstandenen Tupel dar und verletzt somit die Privacy nicht. Eine JA-Anfrage ist demnach als Eingabe für eine Provenance-unterstützte Datenanalyse geeignet.

- **SPJ:** Eine Kombination aus Selektion, Projektion und natürlichem Verbund kann durch einen SQL-Ausdruck der Form "`SELECT spaltennamen FROM tabelle1 NATURAL JOIN tabelle2 WHERE selektionsbedingung`" dargestellt werden. Entspricht die Projektion an dieser Stelle wieder den Privacy-Bestimmungen, so entsprechen auch die projizierten Spalten des selektierten Verbundergebnisses der Privacy. Eine SPJ-Anfrage ist demnach als Eingabe einer Provenance-unterstützten Datenanalyse erlaubt.

- **SPA:** Wie bereits bei der PA-Anfrage erwähnt, besteht auch an dieser Stelle das ProjAgg-Problem. Eine SPA Anfrage kann demnach unter Eliminierung der Projektion höchstens auf eine SA-Anfrage reduziert werden. Eine SPA-Anfrage ist durch das PA-Problem somit nicht als Eingabe einer Provenance-unterstützten Datenanalyse erlaubt, da diese gar nicht umsetzbar ist.

- **SJA:** Eine Selektion in Kombination mit einem natürlichen Verbund und einer Aggregation entspricht einem SQL-Ausdruck der Form "`SELECT AGG(spalte) FROM tabelle1 NATURAL JOIN tabelle2 WHERE selektionsbedingung`". Wie bei anderen Kombinationen mit der Aggregation besteht auch bei dieser Anfrage das Ergebnis nur aus einem einzelnen bzw. aggregierten Zahlenwert. Somit entspricht eine SJA-Anfrageformulierung dem Privacy-Konzept und kann entsprechend als Eingabe einer Provenance-unterstützten Datenanalyse genutzt werden.

- **PJA:** Auch diese Kombination von Anfragen ist durch das PA-Problem nicht umsetztbar und demnach nicht als Eingabe einer Provenance-unterstützten Datenanalyse erlaubt. Eine alternative Anfrageformulierung wird durch eine JA-Anfrage dargestellt.

- **SPJA:** Eine Kombination aller betrachteten Operationen ist ebenfalls von dem ProjAgg-Problem betroffen. Eine solche Anfrageformulierung ist demnach nicht als Eingabe einer Provenance-unterstützten Datenanalyse erlaubt. Eine Alternative stellt eine SJA-Anfrage dar.

Die Erkenntnisse zur Umsetzbarkeit und Privacy der in dieser Arbeit betrachteten Anfragen ist noch einmal in Tabelle 4.1 zusammengefasst.

Anfrage	S	P	J	A	SP	SJ	SA	PJ	PA	JA	SPJ	SPA	SJA	PJA	SPJA
Umsetzbarkeit	✓	✓	✓	✓	✓	✓	✓	✓	X	✓	✓	X	✓	X	X
Privacy	X	✓	X	✓	✓	X	✓	✓		✓	✓		✓		

Tabelle 4.1.: Übersicht komplexerer Auswertungsanfragen

4.1.1. Privacy-Diskussion zur Auswertungsanfrage

Bei den Betrachtungen zu den einzelnen Anfragemöglichkeiten hat sich ergeben, dass sowohl die Selektion als auch der natürliche Verbund, sowie eine Kombination dieser beiden Operationen nicht den Privacy-Anforderungen bei der Veröffentlichung des Ergebnisses Q(D) genügen. Diese Operationen konnten aber in Kombination mit einer Privacy-gerechten Projektion oder aber einer Aggregation entsprechend Privacy-gerecht gestaltet werden. Es besteht jedoch ein weiteres Problem bei Anfragekombinationen, in denen eine Selektion vorhanden ist. Wird eine Selektionsbedingung zu spezifisch formuliert, sodass nur ein einzelnes Tupel selektiert wird, so liefern Aggregatfunktionen mit der Ausnahme von COUNT den exakten Wert für dieses Tupel, was ein Privacy-Risiko darstellen kann. Auch bei der Projektion besteht dadurch die Gefahr der exakten Zuordnung von Attributwerten zur gewählten Selektionsbedingung. Allein aus diesen Betrachtungen wird ersichtlich, dass eine Veröffentlichung der Anfrage Q selbst ein großes Privacy-Risiko darstellen kann, da Zusammenhänge in den Ergebnisdatensätzen mit Hilfe der Anfrage sichtbar werden können. Auch besteht unter Angabe der Anfrage Q eine Möglichkeit der Anfrage-Invertierung, die eine mögliche Schlussfolgerung auf originale Daten vom Ergebnis Q(D) aus ermöglichen kann. Besonders gravierend sind hierbei auch wieder die Selektionsbedingungen, da diese auf bestimmte Attributwerte schließen lassen.

Wird für eine Provenance-unterstützte Datenanalyse eine einzelne Projektion als Auswertungsanfrage Q verwendet, so besteht für Außenstehende die Möglichkeit eines Unsorted-Matching-Angriffs, der nur durch die Form der Auswertungsanfrage als Projektion motiviert sein kann. Durch eine Selektion oder einen natürlichen Verbund kann das Risiko einer solchen Attacke jedoch bereits durch eine Veränderung der Anzahl an Tupeln abgeschwächt werden. Zur Vermeidung eines solchen Angriffs kann auch einfach eine Veränderung der Tupelreihenfolge im Ergebnis mittels Permutation erfolgen, einer Methode, die im nächsten Abschnitt auch im Rahmen der Betrachtungen zur Intensionlisierung eines Datensatzes untersucht wird. Eine Auswertungsanfrage Q sollte also immer auf etwaige Privacy-Risiken untersucht werden!

4.2. Intensionalisierung von Datensätzen

Neben der Auswertungsanfrage Q gehört zur Eingabe einer Datenanalyse auch der zur Auswertungsanfrage passende Datensatz D. Doch bevor die Konventionen zur Übergabe eines zur Auswertunganfrage passenden Datensatzes D formuliert werden können, muss als zusätzliche Bedingung auch eine Kompatibilität des Datensatzes mit der gewählten Anonymisierungsmethode gesichert sein. Zu diesem Zweck beschäftigt sich dieser Abschnitt mit den Möglichkeiten zur Intensionalisierung bzw. Anonymisierung eines relationalen Datensatzes D bzw. des relevanten Teils **D** und des durch Q(D) dargestellten Ergebnisdatensatzes. Abbildung 4.4 hebt die entsprechenden, zur Diskussion stehenden Schritte noch einmal im Ablauf einer Provenance-unterstützten Datenanalyse hervor.

Um eine abgeschwächte aber ausreichende Nachvollziehbarkeit des Ergebnisses Q(D) zu ermöglichen, muss die Anonymisierung des Ergebnisses Q(D) abbildbar auf das Ergebnis der Auswertungsanfrage Q' sein. Demnach spielt die Wahl der Anonymisierungsmethode zur Erstellung von D' bzw. zur Formulierung der intensionalen Provenance-Antwort eine sehr große Rolle. Auch hängt die Formulierung der angepassten Auswertungsanfrage Q' von der Methode zur Intensionalisierung ab. Zwei neue Fragestellungen, die bei der Ausführung einer Anonymisierungsmethode in Kombination mit einer SPJA-Anfrage aufkommen, sind dann:

1. Welche Anonymisierungsmethode ist in Kombination mit einer SPJA-Anfrage kommutativ?
2. Ist eine Kommutativität der Kombination für eine Nachvollziehbarkeit notwendig?

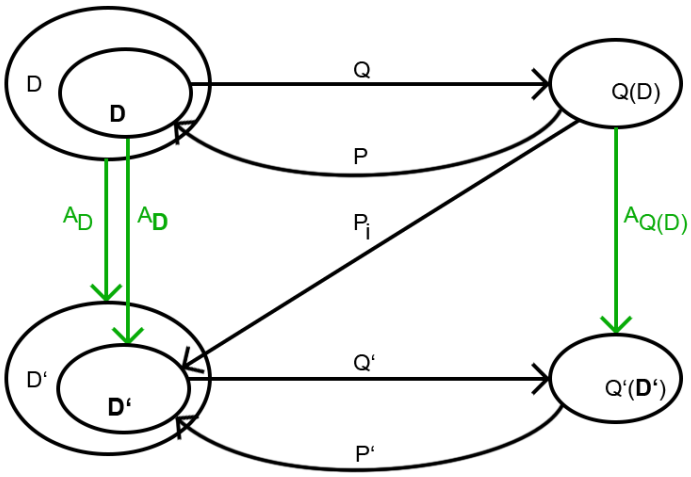

Abbildung 4.4.: Intensionalisierung von Datensätzen

Abbildung 4.5 stellt den Sachverhalt der Fragestellungen noch einmal dar.

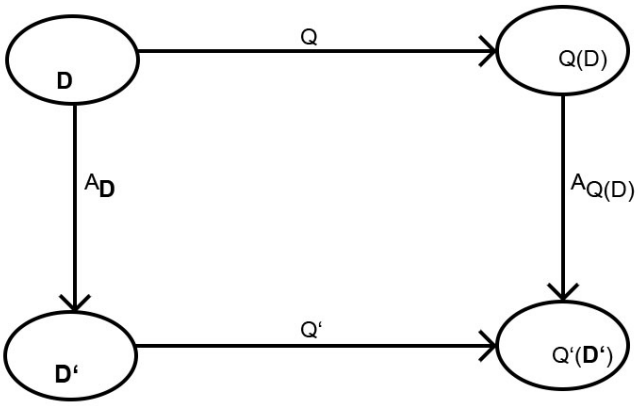

Abbildung 4.5.: Kommutativität SPJA-Anfrage und Anonymisierungsmethode

Zur Beantwortung dieser Fragestellungen werden die in Abschnitt 2.4 dargestellten Methoden zur Anonymisierung mit einzelnen ausgewählten Anfragemöglichkeiten kombiniert. Begonnen wird mit der Differential Privacy und anschließend wird die Methode des Splicens untersucht. Daran schließt sich die Methode des Unterdrückens an, bevor schlussendlich auf die Permutation und Generalisierung eingegangen wird.

Differential Privacy und SPJA-Anfragen: Bei der Differential Privacy ist der größte Vorteil zudem auch der größte Nachteil. Wird ein Datensatz mehrfach mittels Differential Privacy anonymisiert, wie es im gezeigten Verlauf der Fall ist, so werden die Datensätze auf den unterschiedlichen Wegen auch unterschiedlich verrauscht. Eine Ausführung der Auswertungsanfrage vor der Anonymisierung mittels Differential Privacy enthält demnach mit einer sehr großen Wahrscheinlichkeit eine ganz andere Menge von Tupeln (mit Ausnahme der Projektion), wie die Anonymisierung mit anschließender Ausführung der Auswertungsanfrage Q. Eine Abbildung der Ergebnisse dieser beiden Wege ist somit praktisch nicht umsetzbar.

Nehmen wir dazu die die Anfrage 1.1 zum Beispiel. Das Ergebnis dieser Anfrage enthält nur die zwei Einträge der `NOTEN`-Tabelle, die für das Attribut **note** den Wert 1.7 enthalten. Ein zufällig hinzugefügtes Rauschen des Attributs würde für dieselbe Anfrage in den meisten Fällen ein leeres Ergebnis zurückgeben, da die eigentlichen Attributwerte auf einen zufälligen anderen Zahlenwert verrauscht werden. Einzig bei der Aggregation des gesamten Datensatzes kann bei richtiger Umsetzung der Methode ein ähnlicher Wert erzielt werden.

Splicen und SPJA-Anfragen: Auch die Methode des Splicens entfällt aus demselben Grund wie zuvor die Differential Privacy: da innerhalb festgelegter Gruppen eine spaltenweise Permutation stattfindet, die ebenfalls zufällig verläuft, wird das Ergebnis der beiden Wege mit einer sehr hohen Wahrscheinlichkeit sehr unterschiedlich ausfallen, was weder eine Validierung noch eine Nachvollziehbarkeit erlaubt. Ein Zusammenhang beider Ergebnisse ist im besten Fall nur in den Mengen der Attributwerte erkennbar, was an dieser Stelle aber nicht den Anforderungen an eine nachvollziehbare Datenanalyse gerecht wird. Die fehlende notwendige Beachtung korrelierender Attribute für den Erhalt einer Abbildung der Ergebnisse verhindert somit die praktische Umsetzung in der Provenance-unterstützten Datenanalyse.

Nehmen wir als Beispiel wieder Anfrage 1.1, die aus der `NOTEN`-Tabelle die Einträge mit der **note** 1.7 selektiert. Eine zufällige Permutation der Noten innerhalb z.B. durch die **kurs_nr** aufgespannter Buckets würde dafür sorgen, dass die **note** 1.7 einer anderen **student_id** zugeordnet wird, sodass das Ergebnis grundlegend falsche Informationen enthält, die nur für eine Aggregatfunktion auf dem gesamten Datensatz nutzbringend sind.

Unterdrücken und SPJA-Anfragen: Das Unterdrücken einzelner Tupel ist durch die Minderung des Informationsgehalts und die resultierende Verfälschung von Auswertungsergebnissen ebenfalls nicht die sinnvollste Methode, auch wenn eine entsprechende Abbildung erhaltener Ergebnisse möglich wäre. Probleme bereitet vor allem die für eine Datenanalyse wichtige mögliche Umsetzung einer Aggregation. Das Entfernen von Tupel vor der Ausführung einer Aggregation verfälscht das Ergebnis, während nach der Ausführung einer Aggregation überhaupt keine Möglichkeit zur Unterdrückung mehr besteht, da das Ergebnis durch einzelne Zahlenwerte dargestellt wird.

Permutation und SPJA-Anfragen: Mit der Festlegung, dass die Permutation die Reihenfolge von Tupeln innerhalb einer Tabelle vertauscht, also nur die Speicherordnung und somit nicht die Tupel selbst verändert, erhalten wir einen zum originalen Datensatz D äquivalenten permutierten Datensatz D'. Da eine Relation als Menge von Tupeln definiert ist, spielt die Reihenfolge allerdings keine Rolle - D ist gleich D'. Die Auswertungsanfrage Q liefert auch entsprechend ein in der Mengenschreibweise gleiches bzw. in Tabellenform äquivalentes Ergebnis. Die Ausführung einer SPJA-Anfrage und die Permutation sind demnach kommutativ, auch wenn eine Permutation nach Ausführung einer Anfrage mit Aggregation unnötig ist. Kritisch ist jedoch, dass originale Tupel und sensitive Attribute in Gänze erhalten bleiben und die Veröffentlichung des permutierten Datensatzes der Veröffentlichung des originalen Datensatzes D gleich kommt und somit die Privacy absolut verletzt. Ohne weitere Anonymisierung reicht die Permutation alleine nicht einmal aus, um einer Unsorted-Matching-Attacke genügend entgegenzuwirken, da verschiedene Sortierungen für eine Unsorted-Matching-Attacke bei Veröffentlichung des permutierten Datensatzes D' einfach getestet werden könnten. Schlussfolgernd kann die Permutation nur zur Vorbehandlung des eigentlichen Datensatzes D genutzt werden, um vor der Ausführung der Auswertungsanfrage Q oder der Ausführung einer weiteren Anonymisierungsmethode einer Unsorted-Matching-Attacke entgegenzuwirken.

Generalisieren und SPJA-Anfragen: Betrachten wir den originalen Datensatz D, so führt eine Auswertungsanfrage Q zu dem Ergebnis Q(D), welches anschließend mittels Generalsierung einfach anonymisiert werden kann, was auch dem Konzept der Privacy entspricht und eine Veröffentlichung des an-

onymisierten Teildatensatzes ermöglicht. Ein Problem entsteht allerdings bei primärer Generalisierung, da sich Attributwerte und mit ihnen die Wertebereiche verändern können. Eine direkte Anonymisierung mittels Generalisierung kann somit dazu führen, dass die Auswertungsanfrage mit dem anonymisierten relevanten Datensatzteil **D'** nicht mehr kompatibel ist. Dementsprechend muss die Auswertungsanfrage Q so angepasst werden, dass diese wieder mit dem generalisierten Datensatz kompatibel ist. Entsprechend stellt sich die Frage inwiefern Q zu Q' modifiziert werden muss, um eine Kommutativität oder eine Abbildung der Ergebnisse beider Wege zu erhalten, sofern bei einer Anonymisierungsmethode die Kompatibilität von **D'** mit Q verloren geht. Diese Fragestellung wird für jede umsetzbare Auswertungsanfrage in den Abschnitten 4.5 und 4.9 ausführlich beantwortet. In dem genannten Kapitel wird auch erklärt, für welche Anfragen die gewünschte Kommutativität vorliegt und wenn nicht, inwiefern sich die Ergebnisse beider Wege für eine abgeschwächte Nachvollziehbarkeit trotzdem in Beziehung setzen lassen. Es sei vorweggenommen, dass, sofern eine Anpassung der Anfrage für den generalisierten Datensatz erfolgt, auch eine Abbildung der Ergebnisse beider Wege erzielt werden kann. Demnach ist die Generalisierung unter den in [Sch20] vorgeschlagenen Methoden die einzige Möglichkeit zur Intensionalisierung von Datensätzen innerhalb einer Provenance-unterstützten Datenanalyse.

Das Konzept für die Generalisierung von Datensätzen im Ablauf einer Provenance-unterstützten Datenanalyse wird an dieser Stelle noch nicht weiter diskutiert, sondern auf den Abschnitt 4.4 verschoben. Die entsprechenden notwendigen Anpassungen der Auswertungsanfrage Q, zur Formulierung einer mit der Generalisierung kompatiblen Auswertungsanfrage Q', werden ebenfalls auf einen weiteren Abschnitt 4.5 ausgelagert. Im folgenden Abschnitt 4.3 erfolgen jedoch erst einmal wichtige Überlegungen zur Formulierung des Datensatzes D, der die wichtige zweite Hälfte der Eingabe darstellt.

4.3. Datensatz D

Neben einer dem Privacy-Konzept entsprechenden Auswertungsanfrage Q stellt der Datensatz D den zweiten wichtigen Teil der Eingabe einer Provenance-unterstützten Datenanalyse dar. In diesem Abschnitt werden Überlegungen zur Form eines Datensatzes der Eingabe und den in ihm enthaltenen Relationen angestellt. Wie bereits festgestellt, hängt die Form dieser Eingabe von der jeweiligen Formulierung der Auswertungsanfrage Q ab, weswegen diese auch primär behandelt wurde. Abbildung 4.6 zeigt noch einmal den betrachteten Teil der Datenanalyse.

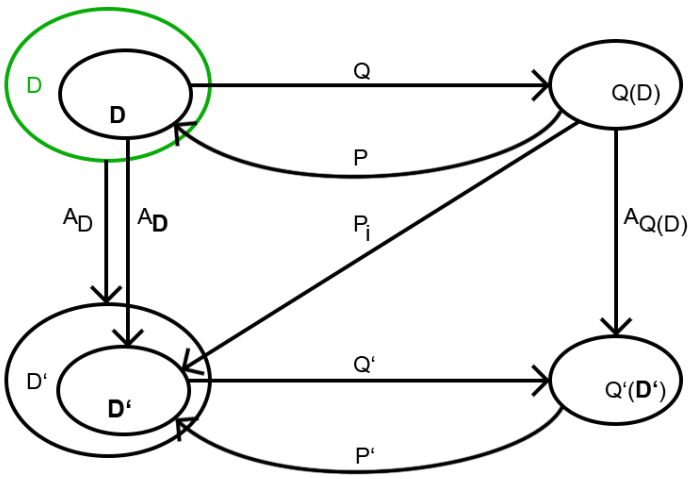

Abbildung 4.6.: Datensatz D

Ein Datensatz sollte für die spätere einfache Umsetzung immer eine einzelne Datenbank mit einer für die Auswertungsanfrage mindestens benötigten Anzahl an Relationen darstellen. Der erste ausschlaggebende Faktor für die Anzahl der Relationen in der Datenbank ist die Form der Auswertungsanfrage Q. Um nicht jede der möglichen Auswertungsanfragen erneut nennen zu müssen, ordnen wir alle Möglichkeiten einer Auswertungsanfragen Q den folgenden zwei Klassen zu.

- **Auswertungsanfragen ohne natürlichen Verbund:** In diese Klassen fallen die folgenden, bisher umsetzbaren und Privacy-gerechten Anfragen einer Provenance-unterstützten Datenanalyse: P, A, SP, SA. Diese Auswertungsanfragen benötigen zur Auswertung von Q jeweils nur eine Tabelle. Der Datensatz D muss demnach mindestens eine Relation enthalten, die hinter `FROM` in der Auswertungsanfrage steht. Man beachte dabei, dass die angegebene Relation die für die Projektion, Aggregation und Selektion benötigten kompatiblen Attribute bzw. Attributwerte enthalten muss.

- **Auswertungsanfragen mit natürlichem Verbund:** In diese Klassen fallen die folgenden, bisher erlaubten Anfragen für die Gestaltung einer Provenance-unterstützten Datenanalyse: PJ, JA, SPJ, SJA. Auswertungsanfragen dieser Klasse benötigen mindestens zwei Relationen, die über ein gleichnamiges Verbundattribut mit gleichen Domänen verbunden werden können. Der Datensatz D muss demnach mindestens zwei Tabellen enthalten, wobei für mögliche Kombinationen mit einer Selektion, Projektion oder Aggregation die entsprechenden Attribute und Attributwerte in mindestens einer der beiden Relationen vorkommen müssen. Die beiden Tabellennamen des Datensatzes stehen in der SQL-Anweisung hinter dem `FROM` und sind durch ein `NATURAL JOIN` voneinander getrennt, was den natürlichen Verbund angibt.

Neben der Anzahl an Relationen, bedingt durch die Auswertungsanfrage Q, muss die Datenbank für das Konzept einer SQL-basierten Generalisierung in Abschnitt 4.4 zusätzlich noch für jedes zu generalisierende Attribut eine minimale-konzeptkonforme Konzeptrelation beinhalten. Die Formulierung einer solchen Konzeptrelation wird ebenfalls im folgenden Abschnitt 4.4 besprochen, der auch die Umsetzung der Generalisierung erläutert.

4.4. Generalisierung

In diesem Abschnitt wird die Formulierung von Konzepthierarchien für die Generalisierung verschiedener Attributwerte erläutert, die ähnlich zu Abschnitt 3.3.1 in verschiedenen Typen zugeordnet werden. Die Anpassung der Auswertungsanfrage Q zu Q', welche in Abschnitt 4.5 beschrieben wird, erfordert jedoch eine bestimmte Form einer Konzepthierarchie für bestimmte Anfragen, die zur Bildung konzeptkonformer Konzepthierarchien in Abschnitt 4.4.2 führen.

Eine rein SQL-basierte Generalisierung eines Attributs, also die Generalisierung einer einzelnen Spalte einer Relation, benötigt für eine relationale Umsetzung jedoch keine Konzepthierarchie, sondern eine Konzeptrelation, mit deren Formulierung sich der Abschnitt 4.4.3 auseinandersetzt. Für eine ressourcensparende relationale Umsetzung reicht bei der Erstellung einer Konzeptrelation die Kombination zweier Stufen in einer minimalen konzeptkonformen Konzeptrelation aus. Die Formulierung solcher Konzeptrelationen werden in Abschnitt 4.4.4 aber noch näher beschrieben. Abbildung 4.7 ordnet die Betrachtungen dieses Abschnitts noch einmal in den Gesamtablauf einer Provenance-unterstützten Datenanalyse ein.

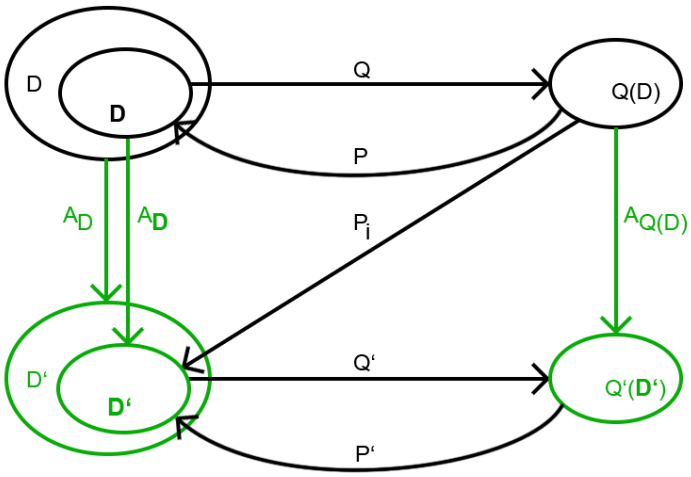

Abbildung 4.7.: Generalisierung von Datensätzen

4.4.1. Konzepthierarchie und Attributtypen

In Abschnitt 3.3.1 wurde bereits eine Einteilung von Attributen zu verschiedenen Typen vorgenommen, die deren Handhabung bei der Formulierung von Konzepthierarchien beschreiben. Diese Typen waren Attribute mit kategorisierbaren numerischen Werten, Attribute mit nicht numerischen Werten ohne sinnvolle Hierarchie und kategorisierbare Attribute mit nicht numerischen Werten. Formuliert wird für ein Attribut des ersten genannten Typs nach [Sva16] einmal eine Konzepthierarchie, beispielsweise für das Attribut **note** der `NOTEN`-Tabelle aus dem Beispieldatensatz. Eine mögliche Konzepthierarchie für die kategorisierbaren numerischen Werte des Attributs **note** ist dann in Abbildung 4.8 zu sehen.

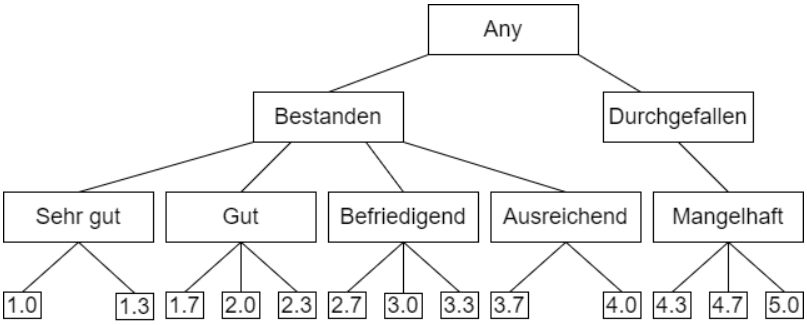

Abbildung 4.8.: Konzepthierarchie für das Attribut **note**

Eine Generalisierung mittels einer solchen Hierarchie ist allerdings nur dann sinnvoll, wenn auf dem Attribut **note** im Anschluss keine Aggregation, mit Ausnahme von COUNT, durchgeführt werden soll. Eine Umsetzung von SUM, AVG, MIN und MAX ließe sich nämlich nicht mehr mit den resultierenden nicht-numerischen Datentypen des generalisierten Attributs durchführen. Auch eine Selektion mit den Vergleichsoperatore $<, \leq, \geq, >$ benötigt auch nach der Generalisierung noch numerische Werte! Abhilfe schafft eine konzeptkonforme Konzeptrelation, siehe Abschnitt 4.4.2.

4.4.2. Konzeptkonformität

Um einen Abbruch einer Provenance-unterstützten Datenanalyse bei der Anpassung von Q zu Q' entsprechend zu vermeiden, muss die Typeinteilung von [Sva16] für eine relationale Umsetzung angepasst

werden. Eine solche Anpassung der Typen für eine relationale Umsetzung der Generalisierung mittels Konzeptrelationen resultiert dann in den folgenden vier Typen:

- 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

Für das Attribut **note** aus Abbildung 4.8 wurde also eine Konzepthierarchie vom neuen Typ 2 erstellt. Die gezeigte Konzepthierarchie ist auch durchaus plausibel, sofern die Eingabe der Datenanalyse eine Auswertungsanfrage ohne Aggregation und ohne Selektion mit den Vergleichsoperatoren $<, \leq, \geq, >$ ist. Soll das Attribut **note** jedoch aggregiert oder mittels einem der Vergleichsoperatoren $<, \leq, \geq, >$ selektiert werden, so muss eine Konzepthierarchie den adäquaten numerischen Wertebereich erhalten, also dem neuen Typ 1 entsprechen. Eine Konzepthierarchie ist in Abbildung 4.9 dargestellt.

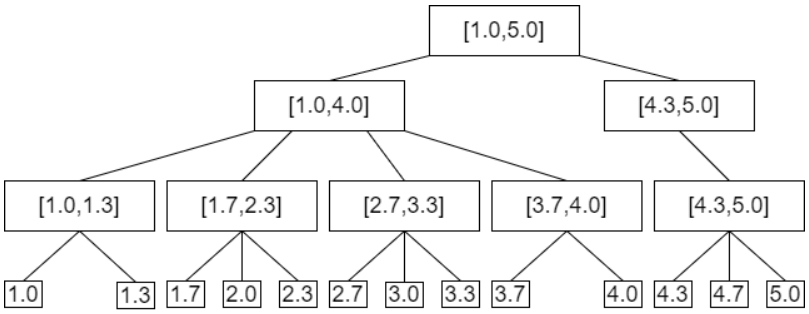

Abbildung 4.9.: Konzepthierarchie für das Attribut **note** vom Typ 1

Man beachte, dass Attribute vom Typ 1 also nur auf Intervalle gleicher numerischer Werte generalisiert werden können. Die Generalisierung erfolgt für Attribute vom Typ 1 also immer auf Intervalle, die durch die Kombination der höchsten Ober- und der niedrigsten Untergrenze aller Kinderknoten der Konzepthierarchie entstehen. Die Wurzel einer solchen Konzepthierarchie enthält als Untergrenze folglich den niedrigsten Attributwert und als Obergrenze den höchsten Attributwert des Attributs. Formulierte Intervalle dürfen demnach keinesfalls überlappen und müssen eine Monotonie aufweisen, dürfen also immer nur größere Werte als das vorherige Intervall und niedrigere Werte als das nachfolgende Intervall enthalten.

Typ 2 umfasst nun alle Attribute, die nicht in den vorher genannten Operationen vorkommen und für die eine sinnvolle Hierarchie erstellt werden kann. Eine dem Typ 2 entsprechende Konzepthierarchie wurde bereits in Abbildung 4.8 für das Attribut **note** erstellt.

Typ 3 umfasst nun alle Attribute, die in keine sinnvolle Hierarchie eingeordnet werden können, aber trotzdem generalisiert werden sollen. Die Konzepthierarchie für solche Attribute enthält immer eine oder zwei zusätzliche Stufen. Die Stufe, die dabei immer vorkommen muss, ist die Wurzel, die alle Attributwerte auf dasselbe anonyme Token oder einen Oberbegriff bzw. Platzhalter abbildet. Eine weitere optionale Stufe kann durch zufällige anonyme Platzhalter erstellt werden, um einzelne Entitäten unterscheiden zu können oder um Korrelationen zu erhalten. Für die Attribute **nachname** und **vorname** der Tabelle `STUDIERENDE` aus dem Beispieldatensatz kann z.B. eine Konzepthierarchie für den Typ 3 erstellt werden. Die erstellten Konzepthierarchien lassen sich dabei in linearer Weise darstellen:

nachname — $*$

vorname — `random` — `any`

Typ 4 stellt zuletzt diejenigen Attribute dar, die originaler Form auch bei einer Veröffentlichung kein Privacy-Problem darstellen und zusätzlicher Aufwand der Generalisierung gespart werden soll. Demnach wird für Attribute vom Typ 4 einfach keine Hierarchie angegeben.

Attribute vom Typ 4 werden bei einer Generalisierung also einfach übersprungen. Eine Generalisierung von Attributen des Typs 2 oder 3 hingegen bildet immer genau einen originalen Attributwert auf genau einen generalisierten Attributwert ab. Um eine Aggregation mit SUM, AVG, MIN, MAX oder eine Selektion mit $\leq, \geq$ auf generalisierten numerischen Werten zu ermöglichen, erfolgt eine Generalisierung des Attributs vom Typ 1 allerdings auf die Unter- und Obergrenze der in der Konzepthierarchie definierten Intervalle, also immer auf zwei generalisierte Attributwerte und somit auch auf zwei Spalten. Die genaue Umsetzung einer Aggregation auf solchen Intervallen wird in Abschnitt 4.5 zur angepassten Auswertungsanfrage erläutert. Entspricht eine Konzepthierarchie für ein Attribut auch dem entsprechen Typ, so steht einer Ausführung der Generalisierung so gut wie nichts mehr im Wege, sie sind also *konzeptkonform*. Eine relationale Umsetzung kann mit Bäumen jedoch relativ wenig anfangen, weswegen der folgende Abschnitt 4.4.3 sich mit der Generierung von Konzeptrelationen aus den besprochenen konzeptkonformen Konzepthierarchien beschäftigt.

4.4.3. Konzeptrelation

Bei einer relationalen Umsetzung einer Generalisierung bringt die übliche Darstellungsform der Konzepthierarchie nur wenig. Soll ein Attribut mit Hilfe von SQL-Anfragen generalisiert werden, spezifischer gesagt durch einen natürlichen Verbund mit anschließender Löschung der alten Attribute, so muss auch die Konzepthierarchie in die Form einer Relation gebracht werden. Bedeutsamer ist dann die Konzepthierarchie des Attributs **note** in Abbildung 4.8, bei der die Konzepthierarchie nicht für eine Aggregation bzw. Selektion mittels $<, \leq, \geq, >$ formuliert wurde. Eine Konzeptrelation wird durch die Generierung eines Tupels für jeden möglichen Ast der Hierarchie erstellt, wobei die Anzahl der Attribute eines Tupels von der Anzahl der Stufen der Hierarchie abhängt. Die Konzeptrelation enthält somit dieselbe Anzahl an Attributen, wie es Stufen in der Konzepthierarchie gibt. Die Anzahl der Einträge hängt von der Anzahl der Blattknoten der Konzepthierarchie ab. Eine solche Konzeptrelation, die für eine relationale Umsetzung der Generalisierung des Attributs **note** genutzt werden könnte, ist in Tabelle 4.2 dargestellt.

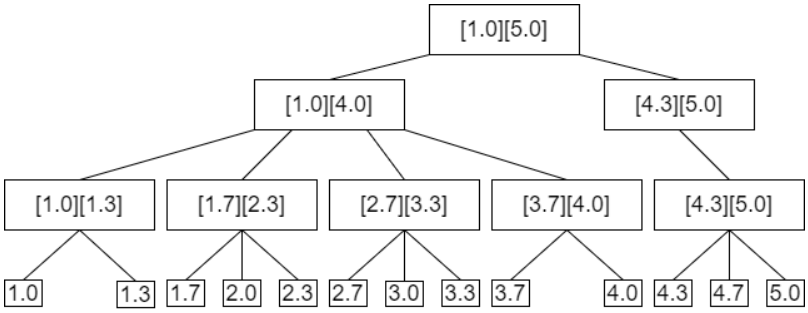

Abbildung 4.10.: Angepasste Konzepthierarchie für das Attribut **note** Typ 1.2

Für ein numerisches Attribut vom Typ 1, welches im Verlauf einer Provenance-unterstützten Analyse jegliche Formen einer Anfrage durchlaufen kann, muss auch die Konzepthierarchie für eine relationale Umsetzung entsprechend umformuliert werden. Aus der Abbildung 4.9 ist zu erkennen, dass bis auf die ursprünglichen Attributwerte nur noch Intervalle von Zahlenwerten in allen weiteren Stufen der Konzepthierarchie dargestellt sind. Da eine Aggregation nur auf Spalten mit numerischen Attributwerten funktioniert, muss eine durch Intervalle dargestellte Generalisierungsstufe auf die numerischen Unter- und Obergrenzen umformuliert werden. Abbildung 4.10 zeigt eine solche Konzepthierarchie.

note	noteGen1	noteGen2	noteGen3
1.0	Sehr Gut	Bestanden	Any
1.3	Sehr Gut	Bestanden	Any
1.7	Gut	Bestanden	Any
2.0	Gut	Bestanden	Any
2.3	Gut	Bestanden	Any
2.7	Befriedigend	Bestanden	Any
3.0	Befriedigend	Bestanden	Any
3.3	Befriedigend	Bestanden	Any
3.7	Ausreichend	Bestanden	Any
4.0	Ausreichend	Bestanden	Any
4.3	Mangelhaft	Durchgefallen	Any
4.7	Mangelhaft	Durchgefallen	Any
5.0	Mangelhaft	Durchgefallen	Any

Tabelle 4.2.: Konzeptrelation für das Attribut **note** vom Typ 2

Eine Konzeptrelation wird dann wieder durch die Formulierung eines Tupels für jeden möglichen Ast des durch die neue Konzepthierarchie aufgespannten Baumes umgesetzt. Für jede Stufe der Konzepthierarchie, die durch Intervalle dargestellt wird, enthält das Tupel nun zwei zusätzliche Attribute. Eine so entstandene Konzeptrelation, wie sie in Abbildung 4.3 für das Attribut **note** zu sehen ist, enthält also eine Spalte mit den originalen Attributwerten und für jede weitere Generalisierungsstufe der Hierarchie eine Spalte für die Untergrenze und die Obergrenze des Intervalls.

note	noteUG1	noteOG1	noteUG2	noteOG2	noteUG3	noteOG3
1.0	1.0	1.3	1.0	4.0	1.0	5.0
1.3	1.0	1.3	1.0	4.0	1.0	5.0
1.7	1.7	2.3	1.0	4.0	1.0	5.0
2.0	1.7	2.3	1.0	4.0	1.0	5.0
2.3	1.7	2.3	1.0	4.0	1.0	5.0
2.7	2.7	3.3	1.0	4.0	1.0	5.0
3.0	2.7	3.3	1.0	4.0	1.0	5.0
3.3	2.7	3.3	1.0	4.0	1.0	5.0
3.7	3.7	4.0	1.0	4.0	1.0	5.0
4.0	3.7	4.0	1.0	4.0	1.0	5.0
4.3	4.3	5.0	4.0	5.0	1.0	5.0
4.7	4.3	5.0	4.0	5.0	1.0	5.0
5.0	4.3	5.0	4.0	5.0	1.0	5.0

Tabelle 4.3.: Konzeptrelation für das Attribut **note** vom Typ 1

4.4.4. Minimale Konzeptrelation

Die in Abschnitt 4.4.3 propagierte Formulierung einer Konzeptrelation zu einer vorgegebenen Konzepthierarchie umfasst allerdings neben den zwei notwendigen Anteilen, dem eigentlichen Attribut und der Stufe auf die generalisiert werden soll, alle anderen Attribute nicht benötigter Stufen. Bei einer relationalen Umsetzung mittels natürlichem Verbund im generalisierten Datensatz müssen somit zusätzliche Löschungen der Attribute der irrelevanten Stufen vorgenommen werden, was zusätzliche Rechenzeit in Anspruch nimmt. Da die Stufe, auf die generalisiert werden soll, vom Autor der Konzepthierarchie ausgewählt wird, kann diese Auswahl direkt bei der Erstellung einer solchen Konzeptrelation beachtet werden, wodurch eine minimale konzeptkonforme Konzeptrelation abgeleitet werden kann. Dafür ist es notwendig, nur das ursprüngliche Attribut und die Attribute der auf die zu generalisierenden Stufe in der Konzeptrelation unterzubringen, was z.B. durch eine Projektion auf einer der bereits erstellten Konzeptrelation geschehen kann. Für Attribute vom Typ 1 bedeutet das somit ein Vorhandensein des eigentlichen At-

tributs und der Unter- und Obergrenze der ausgewählten Generalisierungsstufe, also genau drei Spalten in einer Relation. Für alle anderen Konzepthierarchien enthält eine minimale Konzeptrelation nur das ursprüngliche Attribut und die generalisierte Form dieses Attributs in einer weiteren Spalte (eventuell auch Intervalle, sofern keine weitere Aggregation oder Selektion mit bestimmten Vergleichsoperatoren erfolgen soll). Sollen aber alle Attributwerte durch ein einzelnes Anonymitätstoken oder einen einzigen Oberbegriff ersetzt werden, so reicht für diesen Sonderfall auch ein einzelnes Tupel mit nur einer Spalte aus, in dem dann dieses Token bzw. der Oberbegriff steht. Ein Beispiel für eine minimale Konzeptrelation ist für das Attribut **note** vom Typ 1 in Tabelle 4.4, eine minimale Konzeptrelation für das Attribut **note** als Typ 2 in Tabelle 4.6 dargestellt. Eine minimale Konzeptrelation für das Attribut **nachname** vom Typ 3 ist abschließend in Tabelle 4.5 dargestellt.

note	**noteUG**	**noteOG**
1.0	1.0	1.3
1.3	1.0	1.3
1.7	1.7	2.3
2.0	1.7	2.3
2.3	1.7	2.3
2.7	2.7	3.3
3.0	2.7	3.3
3.3	2.7	3.3
3.7	3.7	4.0
4.0	3.7	4.0
4.3	4.3	5.0
4.7	4.3	5.0
5.0	4.3	5.0

Tabelle 4.4.: Minimale Konzeptrelation für das Attribut **note** vom Typ 1

nachname
*

Tabelle 4.5.: Minimale Konzeptrelation für das Attribut **nachname** vom Typ 3

note	**noteGen**
1.0	Sehr Gut
1.3	Sehr Gut
1.7	Gut
2.0	Gut
2.3	Gut
2.7	Befriedigend
3.0	Befriedigend
3.3	Befriedigend
3.7	Ausreichend
4.0	Ausreichend
4.3	Mangelhaft
4.7	Mangelhaft
5.0	Mangelhaft

Tabelle 4.6.: Minimale Konzeptrelation für das Attribut **note** vom Typ 2

4.4.5. Auswirkungen auf den Datensatz D der Eingabe

Um eine Generalisierung der in der Auswertungsanfrage Q benötigten Ausgangsrelationen zu ermöglichen, muss der Datensatz D neben den genannten Tabellen auch für jedes einzelne Attribut, sofern dieses anonymisiert werden soll, eine gleichnamige, minimale konzeptkonforme Konzeptrelation beinhalten. Neben

der `NOTEN`-Tabelle müssen für die Generalisierung die minimalen konzeptkonformen Konzeptrelationen `kurs_nr`, `student_id`, `semester` und `note` in dem für die Eingabe formulierten Datensatz D vorhanden sein. Um eine einheitliche Umsetzung der Generalisierung zu ermöglichen, müssen die Attribute angegebener Konzeptrelationen auch entsprechend benannt werden. Das ursprüngliche Attribut behält dabei seinen Namen. Die Untergrenze einer Konzeptrelation für Attribute vom Typ 1 trägt dann den Namen des ursprünglichen Attributs plus die Endung "UG", für die Obergrenze geschieht dies analog mit der Endung "OG". Die minimale konzeptkonforme Konzeptrelation `note` enthält also die Attribute **note**, **noteUG** und **noteOG** mit den entsprechenden Tupeln. Bei Konzeptrelationen für Attribute vom Typ 2 erhält der ursprüngliche Attributname in der generalisierten Spalte nur die Endung "Gen". Für das Attribut **semester** der `NOTEN`-Tabelle würde die dazugehörige minimale konzeptkonforme Konzeptrelation `semester` die Attribute **semester** und **semesterGen** enthalten. Attribute vom Typ 3, die auf einen zufälligen Platzhalter zur Beibehaltung von Korrelation generalisiert werden, werden wie Typ 2 gehandhabt. Attribute vom Typ 3, die jedoch auf ein anonymes Token bzw. einen Oberbegriff generalisiert werden, reicht eine dem Attribut gleichbenannte Spalte aus. Für das Attribut **nachname** vom Typ 3 muss also eine minimale konzeptkonforme Konzeptrelation `nachname` mit dem Attribut **nachname** vorliegen. Fehlt eine minimale konzeptkonforme Konzeptrelation für ein Attribut des Typs 4, so werden die enthaltenen Attributwerte für eine Veröffentlichung entsprechend auch nicht generalisiert, was je nach Anwendungsfall ein Privacy-Risiko darstellen kann und demzufolge betrachtet werden muss.

Die Generalisierung erfordert neben der Anpassung des Datensatzes D aber auch eine Anpassung der Auswertungsanfrage Q zu Q' im weiteren Verlauf der Provenance-unterstützten Datenanalyse, um die Kompatibilität der in der Eingabe gewählten Anfrage mit dem generalisierten Datensatz zu gewährleisten. Abschnitt 4.5 befasst sich entsprechend mit dieser Anpassung bzw. Umformulierung der Auswertungsanfrage Q für die Kompatibilität mit dem generalisierten Datensatz D' bzw. **D'**.

4.5. Angepasste Auswertungsanfrage Q'

Der Schritt der Generalisierung für eine bessere Möglichkeit der Veröffentlichung relevanter Datensätze unter Privacy-Aspekten, hat den großen Nachteil, dass die ursprünglich formulierte Auswertungsanfrage Q nicht mehr mit dem generalisierten Datensatz D' bzw. dem relevanten generalisierten Teildatensatz **D'** kompatibel sein muss, z.B. wenn der Attributwert in einer Selektionsbedingung durch die Generalisierung auf einmal nicht mehr existiert. In diesem Abschnitt erfolgen entsprechende Betrachtungen, die sich damit befassen, ob und wie eine Umformulierung der Auswertungsanfrage Q zum Erhalt der Kompatibilität mit generalisierten Datensätzen zu erfolgen hat. Begonnen wird analog zu den Betrachtungen in Abschnitt 4.1 mit den einzelnen Operationen Selektion, Projektion, Verbund und Aggregation, während im Anschluss daran die möglichen Kombinationen dieser Operationen diskutiert werden. Abbildung 4.11 zeigt zu diesem Zweck noch einmal den Teil aktueller Betrachtungen im Ablauf einer Provenance-unterstützten Datenanalyse. Fällt in den folgenden Ausführungen der Begriff Konzeptrelation, so ist immer eine minimale konzeptkonforme Konnzeptrelation gemeint.

- **Selektion:** Bei einer Selektion der Form "`SELECT * FROM tabelle WHERE selektionsbedingung`" sorgt eine Generalisierung für eine Inkompatibilität der Selektionsbedingung $spalte\,\theta\,value$ mit dem generalisierten Datensatz. Wird eine Generalisierung des in der Selektionsbedingung befindlichen Attributs wie in Abschnitt 4.4 vorgenommen, so muss auch eine Generalisierung der in der Selektionsbedingung enthaltenen Bestandteile vorgenommen werden. Das betrifft die Bezeichnung der Spalte, den Vergleichsoperator und den übergebenen Attributwert. Diese Anpassung erfolgt dabei dem Typ des generalisierten Attributs entsprechend, das in *spalte* festgelegt wurde. Liegt für das

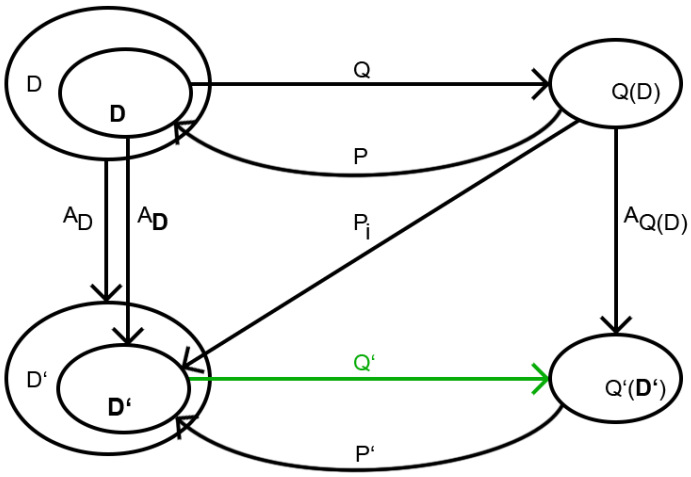

Abbildung 4.11.: Auswertungsanfrage Q’

Attribut jedoch keine Konzeptrelation vor, so muss die Selektionsbedingung auch nicht angepasst werden.

Ist das Attribut vom Typ 2 oder 3 und es liegt eine Konzeptrelation mit 2 Spalten vor, so kann θ nur von der Form $=$ sein. Das Attribut in *spalte* wird dann durch den Ausdruck *spalteGen* und der *value* entsprechend durch das Ergebnis des Ausdrucks “`SELECT spalteGen FROM spalte WHERE spalte=value`” ersetzt.

Ist das Attribut jedoch vom Typ 1 und liegt eine Konzeptrelation mit 3 Spalten vor, so hängen die Anpassungen von dem gewählten Vergleichsoperator θ ab. Ist für θ der Vergleichsoperator $<$ genutzt worden, so muss θ selbst auf $\leq$ generalisiert werden, um bei der Generalisierung auf Intervalle nicht alle in dem Intervall enthaltenen Werte aus dem Ergebnis zu entfernen. Für ein θ der Form $>$ geschieht dies aus demselben Grund zu $\geq$. Eine Anpassung von θ bei den vorliegenden Formen $\leq$, $=$ und $\geq$ ist im Umkehrschluss nicht notwendig. Da entweder die Unter- oder Obergrenze ein Intervall eindeutig identifizieren, reicht die Anpassung des Ausdrucks in *spalte* zu *spalteUG* entsprechend aus. Der Attributwert in der Variable *value* wird dann durch das Ergebnis des Ausdrucks “`SELECT spalteUG FROM spalte WHERE spalte='value'`” ersetzt.

- **Projektion:** Eine Projektion der Form “`SELECT spaltennamen FROM tabelle`” ist leicht an eine Generalisierung anzupassen. Kommen in der Konzeptrelation eines in *spaltennamen* definierten Attributes *spalte* 3 Spalten vor, so wird *spalte* einfach durch $spalteUG, spalteOG$ im Ausdruck *spaltennamen* ersetzt. Für ein Attribut *spalte* mit einer Konzeptrelation mit 2 Spalten erfolgt eine Anpassung der Anfrage mittels Austausch des Ausdrucks *spalte* durch *spalteGen* in den *spaltennamen*. Liegt für ein Attribut keine Konzeptrelation oder eine Konzeptrelation mit nur einer Spalte vor, so wird der Ausdruck *spalte* auch nicht in den *spaltennamen* verändert.

- **Natürlicher Verbund:** Ein natürlicher Verbund kann mit einem SQL-Ausdruck der Form “`SELECT * FROM tabelle1 NATURAL JOIN tabelle2`” auch auf generalisierten Datensätzen ausgeführt werden, weil eine solche Formulierung eines natürlichen Verbundes ebenso wie die Relationennamen unabhängig von der Generalisierung sind. Auch werden die bereits gleich benannten Verbundattribute durch eine einzelne Konzeptrelation auf dieselbe Stufe generalisiert und verlieren die Verbundpartner dadurch nicht.

- **Aggregation:** Eine Aggregation lässt sich durch einen SQL-Ausdruck der Form “`SELECT AGG(spalte) FROM tabelle`” darstellen. Ist $AGG(spalte)$ als COUNT(*) formuliert, so ist keine

Anpassung der Anfrage an den generalisierten Datensatz notwendig. Ist AGG jedoch eines der Schlüsselworte SUM, AVG, MIN, oder MAX, so muss der Ausdruck $AGG(spalte)$ dupliziert werden. Im ersten Ausdruck $AGG(spalte)$ wird das Attribut *spalte* durch $spalteUG$ ersetzt. Im zweiten Ausdruck wird das Attribut *spalte* durch die Obergrenze $spalteOG$ ersetzt. Anschließend wird der ursprüngliche Ausdruck $AGG(spalte)$ durch eine mit Komma getrennte Konkatenation der Aggregationen auf Unter- und Obergrenze $AGG(spalteUG), AGG(spalteOG)$ ersetzt.

Aus diesen Anpassungen der Standardoperationen der relationalen Algebra lassen sich nun die Anpassungen aller möglichen Auswertungsanfragen ableiten. Man beachte jedoch auch an dieser Stelle, dass nicht jede Möglichkeit einer Auswertungsanfrage Q auch umsetzbar ist bzw. der Privacy entsprechen muss, siehe 4.1. Ausgeschlossen werden hierbei die Möglichkeiten der Kombination von Projektion und Aggregation, da eine Anpassung auf demselben Teil der formulierten Auswertungsanfrage Q erfolgen müsste, was nicht möglich ist. Betrachtungen zu den anderen möglichen Kombinationen der Standardoperationen erübrigen sich durch die Anpassung verschiedener Teile in der Anfrage, welches somit keine Konflikte hervorruft. Nachdem nun alle wichtigen Bausteine für eine Datenanalyse erläutert wurden, können diese nunmehr durch Provenance-Informationen nachvollziehbarer und verständlicher gestaltet werden. Abschnitt 4.6 stellt zu diesem Zweck eine Möglichkeit der Provenance-Notation vor und wie weitere Anpassungen einer Datenanalyse für einen Umgang mit dieser Provenance umzusetzen sind.

4.6. ID-basierte Provenance-Notation

An dieser Stelle soll nun eine Lösung für eine toolunabhängige, relationale Provenance-Notation in Form einer eindeutigen ID vorgestellt werden. Abschnitt 4.6.1 geht als erstes auf die Generierung einer benötigten eindeutigen ID inklusive der Möglichkeit zur ID-basierten Permutation ein. Abschnitt 4.6.2 beschreibt abschließend die Umsetzung der Provenance-Notation für die verschiedenen möglichen Anfragetypen. Diese Provenance-Notation wird im Verlauf einer Provenance-unterstützten Datenanalyse auch dazu genutzt, den relevanten Teil eines Datensatzes mittels einer entsprechenden Provenance-Anfrage zu ermitteln. Die Formulierung einer solchen Provenance-Anfrage auf Basis der in diesem Abschnitt vorgestellten Notation der Provenance wird aber erst in Abschnitt 4.7 erläutert.

4.6.1. ID-basierte Permutation

Wie bereits in Kapitel 4.2 beschrieben, verändert eine tupelweise Permutation einen Datensatz D nicht. Gleichsam ist die Ausführung einer solchen primären Permutation sinnvoll, da trotz Erhalt sensibler Informationen in späteren zu entwickelnden Komponenten wie Q(D) oder **D'** einer Unsorted-Matching-Attacke entgegengewirkt werden kann. Eine Permutation sollte demnach sowohl vor der Ausführung von Q, als auch nach der Anonymisierung von **D** ausgeführt werden, um einen erhöhten Schutz vor einer Unsorted-Matching-Attacke in für die Veröffentlichung bestimmten Datensätzen zu gewährleisten. Die Permutation einer Relation kann einfach mittels einer zufällig generierten ID-Spalte umgesetzt werden. Moderne Programmiersprachen und sogar Datenbankmanagementsysteme erlauben die Generierung großer zufälliger Zahlen, wie z.B. eine UUID. Wird ein Datensatz dann nach den in der zufällig generierten Spalte vorhandenen IDs sortiert, kommt das einer Permutation des Datensatzes gleich. Abbildung 4.12 verdeutlicht noch einmal den eben erläuterten Prozess.

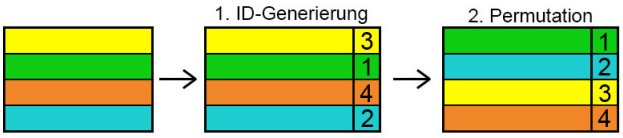

Abbildung 4.12.: ID-basierte Permutation

4.6.2. ID-basierte Provenance und Auswertungsanfrage Q

Warum nun aber eine zu generierende ID benutzen, wenn moderne Datenbankmanagementsysteme bereits Methoden für eine Permutation enthalten und das zufällige Generieren einer ID zusätzliche Rechenleistung kostet? Der Vorteil liegt dabei in der für eine im Ablauf der Datenanalyse notwendigen Provenance-Notation. Betrachten wir hierzu die verschiedenen Standardoperationen und eventuell notwendige Anpassungen in der Auswertungsanfrage zum Umgang mit einer zusätzlichen ID-Spalte, bevor in Abschnitt 4.6.3 der Erhalt der Why-Provenance aus der ID-Notation für die verschiedenen Operationen angegeben wird.

- **Selektion:** Da eine Selektion immer Tupel mit allen Attributen zurückgibt, ist eine Anpassung der Auswertungsanfrage Q nach der Generierung einer zusätzlichen ID-Spalte nicht notwendig. Man beachte jedoch die Ausgabe von Originaltupeln bei einer einzelnen Selektion.

- **Projektion:** Bei einer Projektion werden nur bestimmte Spalten bzw. in der Projektion definierte Attribute zurückgegeben. Um die Rückgabe des zu einem Ergebnis gehörenden ID-Wertes zu garantieren, muss eine Projektionsbedingung immer um die zusätzliche ID-Spalte erweitert werden. Durch die Eindeutigkeit der ID-Werte gleicht das Ergebnis einer Projektion inklusive ID dann dem Ergebnis einer Projektion ohne Duplikateliminierung.

- **Natürlicher Verbund:** Um bei einem Verbund keine zusätzlichen Verbundbedingungen durch gleich benannte ID-Spalten zu schaffen, wird eine einheitliche Benennung der ID-Spalten durchgeführt. Die ID-Spalte für eine Tabelle trägt dann den Namen der *tabelle* plus die Endung "ID". Ein Konflikt entsteht somit nur, wenn zwei zu verbindende Tabellen den gleichen Namen besitzen, was bei der Angabe einer einzelnen Datenbank aber bis auf einen Verbund derselben Tabelle nicht möglich ist. Dies stellt im übrigen keinen praktischen Anwendungsfall dar. Da ein natürlicher Verbund in einer Provenance-unterstützen Datenanalyse durch den Ausdruck "`tabelle1 NATURAL JOIN tabelle2`" angegeben wird, verbleiben beide ID-Spalten im Ergebnis. Die zwei IDs eines Ergebnistupels verweisen dann auf die Originaltupel der Eingaberelationen, deren Kombination für das Ergebnistupel verantwortlich ist. Man beachte auch wieder das Vorliegen von Originaltupelteilen im Ergebnis.

- **Aggregagtion:** Bei der Aggregation werden durch die Art der Formulierung "`SELECT AGG(spalte)`" keine ID-Spalte außer dem Aggregationsergebnis übernommen. Da eine Aggregation immer alle Tupel eines Datensatzes mit in die Ergebnisberechnung mit einbezieht, kann für eine Berechnung der Provenance eine einfache Projektion auf die ID-Spalte der Ausgangsrelation vorgenommen werden, an deren Tupel anschließend das Aggregationsergebnis in entsprechend benannten Spalten angehängt wird. Bei der COUNT-Funktion enthält das Ergebnis somit 2 Spalten, bei der Ausführung von SUM, AVG, MIN und MAX auf einem generalisierten Datensatz wird entsprechend das Ergebnis der Aggregation an die jeweilige Unter- und Obergrenze angehängt, wodurch das Ergebnis 3 Spalten umfasst.

Da die Darstellung der Provenance, die resultierende Anpassung der Anfrage und die später formulierte Provenance-Anfrage stark von der Art der Anfrage abhängt, müssen auch alle Kombinationsmöglichkeiten betrachteter Standardoperationen auf notwendige Anpassungen untersucht werden.

- **SP:** Eine Selektion erfordert keine Anpassung der Anfrage, für die Projektion muss also der Name der ID-Spalte hinzugefügt werden. Die Einträge der im Ergebnis enthaltenen ID-Spalte sind dann durch die Selektion bestimmt.

- **SJ:** Bei einer Kombination von Selektion und natürlichem Verbund muss keine Anpassung vorgenommen werden, da sowohl die Selektion als auch der natürliche Verbund die ID-Spalten der Ausgangsrelationen beibehalten. Im Ergebnis stehen entsprechend 2 ID-Spalten. Man beachte jedoch, dass eine Kombination aus Selektion und natürlichem Verbund Originaltupelteile im Ergebnis darstellen.

- **SA:** Eine Kombination aus Selektion und Aggregation benötigt keine Anpassung der Anfrage, jedoch wird das Ergebnis, bestehend aus Provenance-Spalte mit angehängtem Aggregationsergebnis, anders berechnet als noch bei einer einzelnen Aggregation. Die relevanten Tupel werden nämlich durch die Selektion bestimmt. Somit wird die Spalte mit den relevanten IDs durch eine Projektion auf das Selektionsergebnis erhalten. Die Anzahl der im Ergebnis enthaltenen Spalten ist analog zur Aggregation erklärbar.

- **PJ:** Da eine Projektion nach einem natürlichen Verbund ausgeführt wird, muss eine Anpassung der Projektion beide ID-Spalten der Ausgangsrelationen umfassen. Im Ergebnis charakterisieren die Kombination zweier IDs die relevanten Tupel der Ausgangsrelationen, deren Kombination für das Ergebnis verantwortlich ist.

- **JA:** Bei der Kombination von einem natürlichem Verbund und einer Aggregation werden die relevanten Tupel durch den Verbund bestimmt. Die im Ergebnis vom Verbund vorkommenden ID-Spalten sind an dieser Stelle Ausgang der Erstellung des Ergebnisses, an die das Aggregationsergebnis für jedes Tupel angehängt wird.

- **SPJ:** Eine Anpassung der Anfrage an eine Selektion ist nicht notwendig. Entsprechend sind die Anpassungen analog zu einer PJ-Anfrage vorzunehmen. Eine Projektion muss also die beiden ID-Spalten der Ausgangstabellen umfassen.

- **SJA:** Bei einer solchen Kombination ist der natürliche Verbund in Kombination mit der Selektion für die Generierung der relevanten ID-Spalten zuständig. Eine Ausführung der gesamten Anfrage mit Aggregation liefert entsprechend das Aggregationsergebnis, welches dann für jedes Tupel der relevanten ID-Spalten angehängt wird.

- **PA/SPA/PJA/SPJA:** PA-, SPA-, PJA- und SPJA-Anfragen sind alle vom ProjAgg-Problem betroffen und demnach nicht in einer Provenance-unterstützten Datenanalyse umsetzbar.

Das Konzept der Provenance-Notation durch eine eindeutige ID ist also ohne umfangreiche Anpassungen umsetzbar. Doch inwiefern stellt die präsentierte Provenance-Notation die in Abschnitt 2.3 dargestellte Why-Provenance dar?

4.6.3. ID-basierte Why-Provenance

Eine Umformung der ID-basierten Provenance zur Darstellung der Why-Provenance (Zeugenbasis) ist leicht erklär- und umsetzbar. Auch hierfür werden wieder die verschiedenen Formen von Auswertungsanfragen betrachtet.

- **Selektion:** Eine einzelne Selektion hat im Ergebnis nur eine ID-Spalte. Die Einträge dieser ID-Spalte stellen jeweils den einzigen Zeugen der Zeugenbasis eines Ergebnistupels dar, mit der Annahme, dass die Ausgangsrelation duplikatfrei ist.
- **Projektion:** Eine einzelne Projektion hat im Ergebnis ebenfalls nur eine ID-Spalte. Da das Ergebnis einer Projektion inklusive ID aber einer Projektion ohne Duplikateliminierung gleichkommt, stellt eine ID nicht den einzigen, sondern nur einen der Zeugen der Zeugenbasis eines Ergebnistupels (ohne ID) dar.
- **Natürlicher Verbund:** Beim natürlichen Verbund bleiben die ID-Spalten beider Ausgangsrelationen erhalten. Entsprechend bildet eine Kombination zweier IDs immer einen Zeugen der Zeugenbasis eines Ergebnistupels (ohne die IDs).
- **Aggregagtion:** Bei einer Aggregation wird das Ergebnis aus allen relevanten Tupeln berechnet. Also stellen alle IDs in Kombination den einzigen Zeugen der Zeugenbasis des Aggregationsergebnisses dar.

Nach den Standardoperationen folgen nun die möglichen Kombinationen dieser Operationen.

- **SP:** Da eine Selektion die Anzahl der Tupel der Ausgangsrelation reduziert bleibt die Formulierung der Why-Provenance aus der ID-Spalte im Ergebnis analog zur Projektion. Dies gilt auch für weitere Operationskombinationen mit der Selektion.
- **SJ:** Wie auch bei der SP-Anfrage, ist die Why-Provenance für ein SJ-Ergebnis analog zu einem J-Ergebnis zu formulieren.
- **SA:** Auch bei dieser Anfrage beschränkt die Selektion nur die in die Aggregation eingehenden Tupel. Die ID-Spalte, die durch die Ausführung der Selektion erhalten wird, enthält somit alle Teilzeugen für das Aggregationsergebnis.
- **PJ:** Nach dem natürlichen Verbund liegen ID-Kombinationen als einzelne Zeugen für dessen Ergebnistupel vor. Durch die zusätzliche Projektion muss nur beachtet werden, dass diese Zeugen bei gleichen Tupeln (ohne ID-Spalten) auch eine Zeugenbasis bilden können.
- **JA:** Bei der Kombination von einem natürlichem Verbund und einer Aggregation werden die relevanten Tupel durch den Verbund bestimmt. Die im Ergebnis vom Verbund vorkommenden ID-Spalten sind an dieser Stelle Ausgang der Erstellung des Aggregationsergebnisses, wodurch alle ID-Kombinationen Teil des einzigen Zeugen der Zeugenbasis sind.
- **SPJ:** Eine Selektion schränkt wie bei SP-Anfragen schon erwähnt nur die Ausgangsrelation ein. Die Darstellung der Why-Provenance ist somit analog zu PJ-Anfragen.
- **SJA:** Durch die Natur der Selektion erfolgt auch die Darstellung der Why-Provenance in diesem Anfragetyp analog zur JA-Anfrage.
- **PA/SPA/PJA/SPJA:** PA-, SPA-, PJA- und SPJA-Anfragen sind alle vom ProjAgg-Problem betroffen und demnach nicht in einer Provenance-unterstützten Datenanalyse umsetzbar.

Eine Darstellung der Why-Provenance für verschiedene Anfrageergebnisse ist demnach durch anfragebasierte Anpassungen der ID-basierten Provenance-Notation ohne weitere Komplikationen möglich. Werden keine Anpassungen zur Formulierung einer Why-Provenance vorgenommen, so symbolisieren die IDs immer noch die relevanten Tupel der Ausgangsrelationen ähnlich zur Darstellung der Data Lineage aus Abschnitt 2.3.1. Durch eine Darstellung aller relevanten Tupel in Form der ID lassen sich die Ausgangsrelationen einer Anfrage auf die relevanten Teile einschränken. Die Ermittelung der relevanten Datensatzteile erfolgt dann durch eine entsprechende ID-basierte Provenance-Anfrage, die im folgenden Abschnitt 4.7 beschrieben werden.

4.7. ID-basierte Provenance-Anfrage

In diesem Abschnitt wird erläutert, wie eine Provenance-Anfrage anhand der in Abschnitt 4.6 propagierten ID-basierten Provenance-Notation zur Ermittlung der relevanten Datensatzteile formuliert werden kann. Abbildung 4.13 zeigt den besprochenen Teilschritt noch einmal im gesamten Ablauf eine Provenance-unterstützten Datenanalyse.

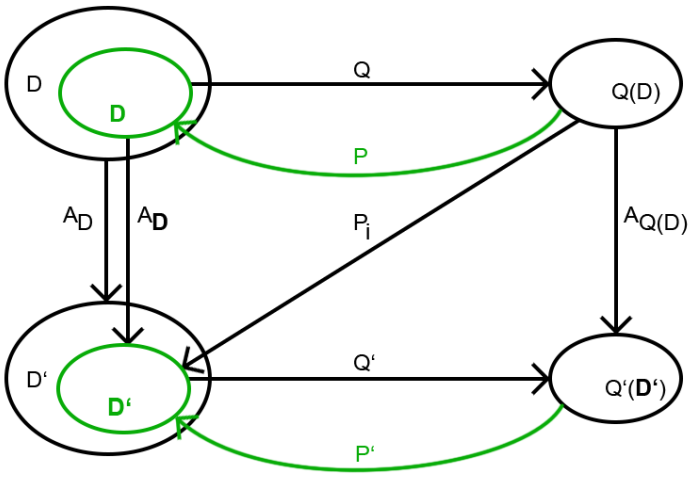

Abbildung 4.13.: Provenance-Anfragen P und P’

Die Formulierung einer Provenance-Anfrage P bzw. P’ ist im Gegensatz zu bereits besprochenen Umsetzungen der Teilschritte einfacher zu erklären. Die Art der Provenance-Anfrage hängt dabei jeweils nur von der Anzahl der im Ergebnis vorkommenden ID-Spalten ab. Eine Provenance-Anfrage muss auch auf dem generalisierten Ergebnis nicht an die durchgeführte Generalisierung angepasst werden, da die ID-Spalten erst nach der Generalisierung hinzugefügt werden und demnach vom Ergebnis ausgeschlossen sind. Dementsprechend unterteilen wir Provenance-Anfragen in zwei Kategorien:

- **Provenance-Anfragen auf Anfrageergebnissen ohne natürlichen Verbund:** Auswertungsanfragen Q ohne natürlichen Verbund enthalten im entsprechenden Anfrageergebnis nur eine einzelne ID-Spalte, die die relevanten Tupel referenzieren. Mit der Annahme, dass eine Auswertungsanfrage ohne natürlichen Verbund auf der in *tabelle* definierten Ausgangsrelation das Ergebnis *result* liefert, so ist das in *tabelle* enthaltene Attribut **tabelleID** ebenfalls in der Ergebnistabelle *result* enthalten. Um die Ausgangsrelation *tabelle* auf den für das Ergebnis relevanten Teil zu kürzen, kann eine SQL-Anfrage der Form “`SELECT * FROM tabelle WHERE tabelleID IN {SELECT tabelleID`

`FROM result}`" genutzt werden. Da die in dieser Arbeit betrachteten Anfragen dieser Kategorie allerdings nur die Operationen Projektion, Aggregation, Selektion und deren Kombinationen umfassen und die Projektion und Aggregation immer den gesamten Datensatz als relevanten Teil haben, kann eine Provenance Anfrage auch durch die einfache Ausführung der isolierten Selektion ermittelt werden, was rechnerisch kostengünstiger ist als die vorher formulierte Anfrage.

- **Provenance-Anfragen auf Anfrageergebnissen mit natürlichem Verbund:** Bei der Provenance-Anfrage auf ein Anfrageergebnis, das durch eine Auswertungsanfrage mit einem natürlichen Verbund entstanden ist, sieht die Provenance-Anfrage entsprechend des Vorkommens zweier ID-Spalten anders aus. Bei analoger Annahme, dass die in einer Anfrage definierten Ausgangsrelationen *tabelle*1 und *tabelle*2 für das Ergebnis *result* sorgen, sind auch die ID-Spalten $tabelle1ID$ und $tabelle2ID$ im Anfrageergebnis *result* enthalten. Da es sich um zwei Ausgangsrelationen handelt, müssen auch zwei Anfragen zur Ermittlung der relevanten Teile beider Relationen ausgeführt werden. Die Erste ist durch einen SQL-Ausdruck der Form "`SELECT * FROM tabelle1 WHERE tabelle1ID IN (SELECT tabelle1ID FROM result)`", die Zweite durch einen Ausdruck der Form "`SELECT * FROM tabelle2 WHERE tabelle2ID IN (SELECT tabelle2ID FROM result)`" umsetzbar.

Mit einer solchen Provenance-Anfrage lassen sich also die relevanten Teile für verschiedene Ergebnisse von Auswertungsanfragen ermitteln. Da nur die Veröffentlichung eines generalisierten relevanten Datensatzes dem Privacy-Konzept entspricht, kann eine intensionale Beantwortung einer solchen Provenance-Anfrage erfolgen, die den relevanten Teildatensatz auch in generalisierter Form zurückgibt. Das Konzept einer solchen intensionalen Provenance-Anfrage unter Nutzung der ID-basierten Provenance-Notation erfolgt im nächsten Abschnitt 4.8.

4.8. Intensionale Provenance-Anfrage

Eine intensionale Provenance-Anfrage ist dazu da, die Ausgabe des relevanten Teildatensatzes in anonymisierter Form bereitzustellen, also eine Beschreibung des für die Auswertungsanfrage Q relevanten Datensatzes zu berechnen, die den Privacy-Ansprüchen des Urhebers genügt. Abschnitt 4.2 beschrieb verschiedene Möglichkeiten für eine solche Intensionalisierung, wobei sich nur die Generalisierung als praktikabel für den Verlauf einer provenance-unterstützten Datenanalyse herausstellte. An dieser Stelle wird also eine Provenance-Anfrage aus Abschnitt 4.7 mit der Anonymisierungsmethode des Generalisierens aus Abschnitt 4.4 verbunden, um einen relevanten anonymen Datensatz bereitzustellen, der wiederum für eine abgeschwächte Nachvollziehbarkeit unter Einhaltung von Privacy-Aspekten sorgt. Abbildung 4.14 ordnet die aktuellen Betrachtungen wieder in den Gesamtablauf einer Provenance-unterstützten Datenanalyse ein.

Die Formulierung einer intensionalen Provenance-Anfrage und die Bildung einer entsprechenden intensionalen Provenance-Antwort kann auf verschiedene Arten erfolgen. Eine erste Möglichkeit zur Bildung intensionaler Provenance-Antworten beschrieb Jan Svacina in [Sva16], deren grundlegende Ideen in Abschnitt 3.3.1 bereits erläutert wurden. Zusammenfassend bewirbt Svacina in seiner Arbeit die Formulierung einer intensionalen Provenance-Antwort durch die Ausführung einer Provenance-Anfrage mit anschließender Generalisierung, angelehnt an [YP99]. Für die Formulierung der Data Provenance bedient sich der Ansatz aus [Sva16] jedoch einem speziellen Tool, was an dieser Stelle nicht für eine Konzeption in Betracht gezogen wird, um eine relationale Lösung so universell wie möglich zu gestalten und entsprechend mögliche Implementierungen in verschiedenen Systemen nicht von vornherein auszuschließen. Eine toolunabhängige, eigene Lösung für die Notation der Data Provenance ist dafür in Abschnitt 4.6 schon

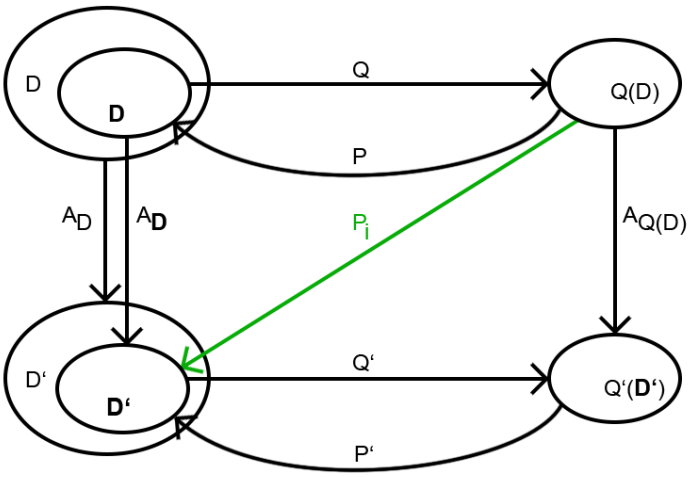

Abbildung 4.14.: Intensionale Provenance-Anfrage

erläutert worden. Auch das eigene Konzept zur Formulierung einer Provenance-Anfrage für die verschiedenen Standardoperationen und die propagierte Notation der Provenance wurde bereits in einem weiteren Abschnitt 4.7 ausführlich beschrieben. In Kombination mit der selbst entwickelten toolunabhängigen Umsetzung der Generalisierung mittels Konzeptrelationen in 4.4, kann eine Formulierung einer intensionalen Provenance-Antwort und deren Beantwortung auch für die eigenen Konzepte abgeleitet werden, alle notwendigen Bausteine wurden ja schon ausführlich dokumentiert.

Da eine intensionale Provenance-Anfrage ebenfalls von der Art der Auswertungsanfrage Q abhängt, folgt nun eine Betrachtung zur Berechnung einer intensionalen Provenance-Antwort, in gewohnter Weise zuerst für die verschiedenen Standardoperationen und anschließend für deren Kombinationen.

- **Selektion nach Svacina:** Handelt es sich bei der Auswertungsanfrage Q um eine reine Selektion, so ist eine Formulierung der intensionale Provenance-Antwort zwar möglich, aber unnötig, da das Ergebnis einer Selektion immer auch den relevanten Teil des Originaldatensatzes darstellt und die Privacy verletzt. Soll trotzdem eine intensionale Provenance-Antwort generiert werden, so reicht eine Generalisierung des Anfrageergebnisses entsprechend aus.

- **Projektion nach Svacina:** Handelt es sich bei der Auswertungsanfrage Q um eine reine Projektion, so ist eine Provenance-Anfrage nur zusätzliche Berechnungszeit, da alle Tupel der Ausgangsrelation relevant den relevanten Teildatensatz darstellen. Die Generierung einer intensionalen Provenance-Antwort kann demnach durch die Generalisierung des gesamten Datensatzes D erhalten werden.

- **Natürlicher Verbund nach Svacina:** Bei der Ausführung eines natürlichen Verbundes ist es notwendig, zuerst die relevanten Tupel in den Ausgangsrelationen zu ermitteln, wofür die in Abschnitt 4.7 definierte Provenance-Anfrage für Anfragen mit Verbund genutzt wird. Anschließend werden die neuen Relationen, die nur die relevanten Tupel enthalten, generalisiert und bilden so die intensionale Provenance-Antwort. Man bedenke aber, dass der Verbund ebenfalls wieder Originaltupelanteile im Ergebnis enthält!

- **Aggregation nach Svacina:** Wie auch bei der Projektion sind bei einer Aggregation alle Ausgangstupel relevant. Eine intensionale Provenance-Antwort kann demnach wieder ohne die zusätzliche Ausführung einer Provenance-Anfrage durch die simple Generalisierung der Ausgangsrelation erhalten werden.

An dieser Stelle erfolgt nun die Betrachtung zur Formulierung einer intensionalen Provenance-Antwort basierend auf dem Ansatz von [Sva16] für komplexere Auswertungsanfragen.

- **SP/SA nach Svacina:** Eine Selektion mit anschließender Projektion oder Aggregation nutzt an dieser Stelle auch wieder das Ergebnis der einzelnen Selektion zur Ermittlung der IDs der relevanten Ausgangstupel, da das unnötigen Rechenaufwand vermeidet. Eine Generalisierung des Ergebnisses der Selektion liefert dann die intensionale Provenance-Antwort für eine SP- oder SA-Anfrage.
- **SJ/PJ/JA/SPJ/SJA nach Svacina:** Für alle Anfragen mit einem natürlichen Verbund wird eine in Abschnitt 4.7 definierte Provenance-Anfrage ausgeführt, um den relevanten Teil der Ausgangsrelationen zu ermitteln. Eine anschließende Generalisierung der relevanten Ausgangsrelationen bildet dann die intensionale Provenance-Antwort. Man beachte auch an dieser Stelle wieder, dass eine SJ-Anfrage Teile der Ausgangsrelationen im Ergebnis enthält, die die Privacy verletzen.
- **PA/SPA/PJA/SPJA nach Svacina:** PA-, SPA-, PJA- und SPJA-Anfragen sind alle vom ProjAgg-Problem betroffen und demnach nicht in einer Provenance-unterstützten Datenanalyse umsetzbar.

Eine intensionale Provenance-Antwort ist im Gesamtablauf einer Provenance-unterstützten Analyse als **D'** dargestellt und wird oft auch einfach generalisierter bzw. anonymisierter relevanter Teildatensatz genannt. Betrachtet man allerdings die möglichen Wege, um einen solchen Datensatz zu erhalten, so fällt auf, dass die Generierung einer intensionalen Provenance-Antwort auch auf andere Art und Weise erfolgen kann. Im Gegensatz zum Ansatz nach Svacina wird die intensionale Provenance-Antwort beim alternativen Weg durch die Generalisierung des gesamten Datensatzes D, einer anschließenden Ausführung der angepassten Auswertungsanfrage Q' und der abschließenden Provenance-Anfrage P' ermittelt, der im weiteren Verlauf als Ansatz nach Lamster bezeichnet wird. Der Unterschied der beiden Ansätze ist noch einmal grafisch in Abbildung 4.15 dargestellt.

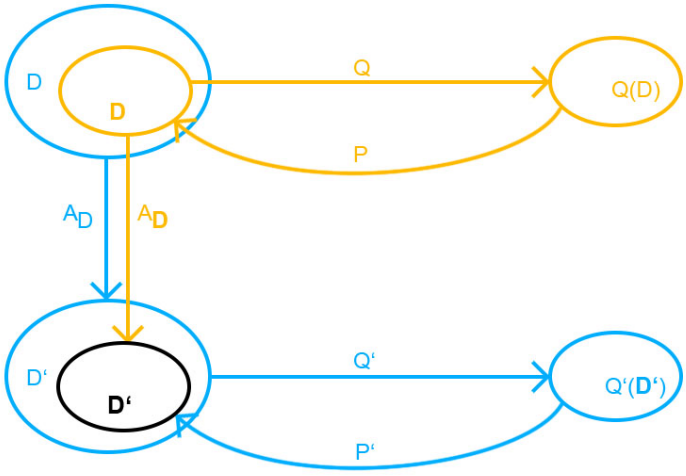

Abbildung 4.15.: Berechnung der intensionalen Provenance-Antwort nach Svacina und Lamster

An dieser Stelle erfolgt nun die Präsentation der jeweiligen Konzepte für die Generierung einer intensionalen Provenance-Antwort nach Lamster. Begonnen wird wieder mit den Standardoperationen und danach erfolgt die Diskussion zur möglichen Umsetzung der Kombinationen.

- **Selektion nach Lamster:** Bei einer Selektion nach Lamster wird zuerst der gesamte Datensatz D generalisiert, entsprechend muss auch die Selektionsbedingung der Anfrage Q an die Selektionsbedingung in Q' angepasst werden. Die Ausführung von Q' liefert dann das generalisierte Selektionsergebnis. Wie bei dem Ansatz nach Svacina auch, stellt das Selektionsergebnis bereits den relevanten

Teil des Datensatzes dar, nur eben des generalisierten Datensatzes, was dementsprechend bereits die intensionale Provenance-Antwort ist. Durch die primäre Generalisierung umfasst der relevante anonyme Datensatzteil **D'** nach Lamster mehr anonyme, aber auch irrelevante Tupel als der Ansatz nach Svacina. Vor allem in Kombinationen mit anderen Standardoperationen, bei denen der Einschluss irrelevanter Tupel für eine Nachvollziehbarkeit nicht ins Gewicht fällt, kann eine Selektion nach Lamster für mehr Privacy sorgen, da selbst bei bekannter Auswertungsanfrage Q nicht auf diejenigen Tupel geschlossen werden kann, die die Selektionsbedingung eigentlich erfüllen. Im Gegensatz dazu eignet sich dieser Ansatz nicht in Kombination mit einer Aggregationsoperation, da die zusätzlichen irrelevanten Tupel das Ergebnis verfälschen können.

- **Projektion nach Lamster:** Da eine Projektion immer alle Ausgangstupel verarbeitet, unterscheidet sich die intensionale Provenance-Antwort nach Lamster auch nicht von der nach Svacina. Auch bei diesem Ansatz reicht dementsprechend die Generalisierung der Ausgangsrelation aus, um den relevanten anonymisierten Datensatz **D'** bzw. die intensionale Provenance-Antwort zu berechnen.

- **Natürlicher Verbund nach Lamster:** Beim natürlichen Verbund nach Lamster werden die Ausgangsrelationen zuerst generalisiert und anschließend per angepasster Anfrage Q' verbunden. Durch die primäre Generalisierung ist auch eine Generalisierung von Verbundattributen möglich, was zusätzliche Verbundbedingungen und somit zusätzliche Ergebnistupel nach sich ziehen kann, was in Kombination mit einer Selektion z.B. für eine erhöhte Privacy sorgen kann, da die Selektionsbedingung durch den Einschluss irrelevanter Tupel nicht immer wahr sein muss, wie es beim Ansatz nach Svacina der Fall ist. Eine Provenance-Anfrage auf dem generalisierten Ergebnis liefert dann die entsprechende intensionale Provenance-Antwort. Man beachte, dass durch die Generalisierung bei der Ausführung von Q' im schlimmsten Fall das Kreuzprodukt aller generalisierten Tupel der Ausgangsrelationen entstehen kann. Beim Ansatz nach Svacina hingegen kann höchstens das Kreuzprodukt aller relevanten Tupel der Ausgangsrelationen entstehen, was bei großen Datensätzen eine deutlich bessere Performance aufweisen sollte.

- **Aggregation nach Lamster:** Wie auch bei Svacina gehen alle Ausgangstupel in die Berechnung einer Aggregation mit ein, entsprechend sind auch die relevanten anonymen Datensätze **D'** bzw. die intensionale Provenance-Antwort beider Ansätze gleich. Die intensionale Provenance-Antwort kann demnach wieder durch die direkte Generalisierung der Ausgangsrelation erstellt werden.

Nach den Standardoperationen erfolgt nun die Beschreibung zur Berechnung einer intensionalen Provenance-Antwort für komplexere Anfragen.

- **SP nach Lamster:** Eine Selektion mit anschließender Projektion nach Lamster beginnt immer mit der Generalisierung des Datensatzes D und der Anpassung der Selektionsbedingung für eine angepasste Auswertungsanfrage Q'. Da die Projektion selbst keine Einschränkungen der Ausgangsrelation vornimmt, ist somit nur die angepasste Selektion für die Ermittlung der relevanten Tupel zuständig. Eine intensionale Provenance-Antwort lässt sich also alleine durch die isolierte angepasste Selektion auf dem generalisierten Datensatz ermitteln, eine Provenance-Anfrage ist entsprechend nur zusätzlicher Verbauch von Ressourcen. Durch Generalisierung der Selektionsbedingung kann die intensionale Provenance-Antwort mehr Tupel enthalten als die vom Svacina-Ansatz, was bei konkreten Selektionsbedingungen allerdings für eine erhöhte Privacy sorgen kann.

- **SA nach Lamster:** Eine Selektion mit anschließender Aggregation nach Lamster beginnt immer mit der Generalisierung des Datensatzes D und der Anpassung der Selektionsbedingung für eine angepasste Auswertungsanfrage Q'. Da eine Aggregation selbst keine Einschränkungen des Originaldatensatzes vornimmt, ist nur die angepasste Selektion für die Ermittlung der relevanten Tupel zuständig. Eine intensionale Provenance-Antwort lässt sich also alleine durch die isolierte angepasste

Selektion auf dem generalisierten Datensatz ermitteln. Durch Generalisierung der Selektionsbedingung kann die intensionale Provenance-Antwort allerdings mehr Tupel enthalten als das Ergebnis vom Svacina-Ansatz. Dies wird an dieser Stelle auch zu einem Problem, da der Einschluss von für Q(D) irrelevanten Tupeln ein Aggregationsergebnis verfälscht, sodass dieses nicht mehr durch Vergleichsoperatoren nachvollziehbar sein muss. Die Berechnung einer intensionalen Provenance-Antwort für eine Selektion in Kombination mit einer Aggregation ist deshalb mit dem Ansatz von Svacina vorzunehmen.

- **SJ/PJ/SPJ nach Lamster:** Bei einer Selektion oder Projektion in Kombination mit einem natürlichen Verbund ist der Ablauf zur Berechnung der intensionalen Provenance-Antwort wie in Abbildung 4.15 gezeigt. Die Generalisierung des Datensatzes D liefert D', die Anpassung der Auswertungsanfrage Q liefert die angepasste Auswertungsanfrage Q', deren Ausführung zum generalisierten Ergebnis Q(**D'**) führt. Eine Provenance-Anfrage auf diesem Ergebnis liefert anschließend die intensionale Provenance-Antwort, die durch die Generalisierung wieder mehr Tupel bei der Selektion und dem natürlichen Verbund mit einschließt, was in bestimmten Fällen der Privacy dienlich sein kann.

- **JA/SJA nach Lamster:** Da eine primäre Generalisierung bei einem Verbund zu zusätzlichen Verbundbedingungen führen kann, was wiederum zu zusätzlichen Tupeln im Ergebnis und schlussendlich zum Einschluss eigentlich irrelevanter Tupel in der intensionalen Provenance-Antwort führt, ist eine Umsetzung der Kombination von einem natürlichem Verbund und einer Aggregation nicht erlaubt. Durch den Einschluss irrelevanter Tupel kann das Ergebnis der Aggregation nämlich entsprechend verfälscht werden, weswegen eine solche Umsetzung einer Anfrage unter Privacy-Aspekten nicht mehr nachvollziehbar wäre. Das Problem des generalisierten Verbunds tritt auch im Svacina-Ansatz auf, wenn die Generalisierung des Ergebnisses Q(D) durch die Auswertung von Q' validiert werden soll, denn auch dort kann ein Verbund auf dem generalisierten für zusätzliche Tupel und somit ein falsches Ergebnis der Aggregation führen. Dieses Problem wird im weiteren Verlauf als JoinAgg-Problem bzw. JA-Problem bezeichnet, welches im Gesamtablauf noch zum Ausschluss einiger weiterer Anfragetypen im Svacina-Ansatz führen wird, obwohl eine Formulierung einer intensionalen Provenance-Antwort an sich möglich ist.

- **PA/SPA/PJA/SPJA nach Svacina:** PA-, SPA-, PJA- und SPJA-Anfragen sind alle vom ProjAgg-Problem betroffen und demnach nicht in einer Provenance-unterstützten Datenanalyse umsetzbar.

Die verschiedenen Ansätze zur Generierung einer intensionalen Provenance-Antwort sorgen in einer Provenance-unterstütztem Datenanalyse also für unterschiedliche Ausprägungen des Kompromisses von Nachvollziehbarkeit und Privacy. Auch die Umsetzbarkeit der Anfragetypen variiert mit der Form des Ansatzes. Die Wahl des Ansatzes zur Generierung einer intensionalen Provenance-Antwort in einer Provenance-unterstützten Datenanalyse ist demnach auch wieder abhängig von der in der Eingabe festgelegten Auswertungsanfrage Q. Der nächste Abschnitt 4.9 betrachtet die verschiedenen Auswertungsanfragen im Kontext eines Konzeptablaufs der zu gestaltenden Datenanalyse und argumentiert auftretende Privacy-Risiken und Probleme zur Umsetzbarkeit für einzelne Eingaben.

4.9. Konzept eines Gesamtablaufs

In diesem Abschnitt werden alle Erkenntnisse aus den vorherigen Abschnitten zusammengetragen, um eine Provenance-unterstützte Datenanalyse für jede der in dieser Arbeit besprochenen Auswertungsanfragen zu

formulieren, sofern dies denn möglich ist. Für jede dieser Anfragen werden die verschiedenen Möglichkeiten zur Formulierung der intensionalen Provenance-Antwort in Bezug auf Privacy und Nachvollziehbarkeit diskutiert. Angefangen wird in gewohnter Weise mit den Standardoperationen, bevor ein Gesamtablauf für komplexere Anfragekombinationen diskutiert wird. Abbildung 4.16 zeigt noch einmal den gesamten Ablauf einer Provenance-unterstützten Datenanalyse.

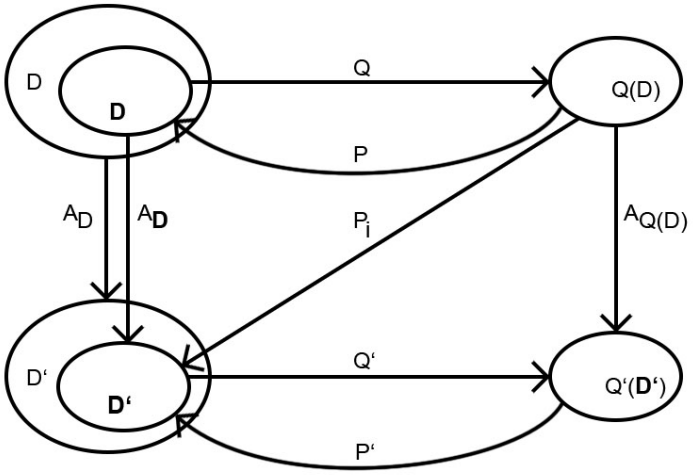

Abbildung 4.16.: Provenance-unterstützte Datenanalyse

- **Ablauf einer Selektion:** Das das Ergebnis einer Selektion immer Originaltupel enthält, ist eine Provenance-unterstützende Datenanalyse zur Verbesserung der Privacy sinnlos. Der für das Ergebnis relevante Teil des Datensatzes ist immer auch das Ergebnis der Anfrage, eine Provenance-Anfrage ist somit eine Identitätsabbildung und trägt nicht sonderlich zur Nachvollziehbarkeit bei. Die Nachvollziehbarkeit ist nämlich alleine durch die Veröffentlichung von Q(D) = D und Q gegeben, was, wie schon oft erwähnt, gegen die Privacy verstößt.

- **Ablauf einer Projektion:** Vor der Ausführung einer Projektion wird der Datensatz durch eine ID-basierte Permutation gegen Unsorted-Matching-Angriffe vorbereitet. Die Ausführung der Anfrage Q liefert dann das Ergebnis Q(D), wobei darauf geachtet werden muss, dass eine Projektion nur auf dem Privacy-Konzept entsprechenden Attributen ausgeführt werden darf. Die Ausführung einer Provenance-Anfrage ist nur zusätzlicher Aufwand, da immer die gesamte Ausgangsrelation bei einer isolierten Projektion den relevanten Teil darstellt. Durch die Angabe von minimalen konzeptkonformen Konzeptrelationen im Datensatz D, kann dann die in der Auswertungsanfrage Q definierte Ausgangsrelation anschließend generalisiert werden, um die intensionale Provenance-Antwort zu formulieren. Die intensionalen Provenance-Antworten nach Svacina und Lamster fallen durch die Relevanz des gesamten Datensatzes D für eine Projektion gleich aus. Eine Anpassung der Auswertungsanfrage Q an die Generalisierung erlaubt anschließend die Berechnung eines generalisierten Ergebnisses aus der intensionalen Provenance-Antwort. Wird das Ergebnis Q(D) nun auch mittels der minimalen konzeptkonformen Konzeptrelationen generalisiert, so entsteht dasselbe Ergebnis wie durch die Anfrage Q', wodurch sich das eigentliche Ergebnis Q(D) validieren lässt. Die Projektion ist somit eine Auswertungsanfrage, die in Kombination mit der Generalisierung kommutativ ist. Um eine Nachvollziehbarkeit zu gewährleisten, ist somit nur die intensionale Provenance-Antwort, das Ergebnis Q(D) und die Auswertungsanfrage Q, sowie die für die Generalisierung benötigten Konzeptrelationen notwendig. Um keine Rückschlüsse auf konkrete Werte vom eigentlichen zum generalisierten Ergebnis zu ermöglichen, kann nach der Generalisierung eine weitere ID-basierte Permutation mit einer neuen ID durchgeführt werden. Dadurch geht allerdings die Referenz vom Ergebnis Q(D) auf die intensionale Provenance-Antwort **D'** verloren.

- **Ablauf eines natürlichen Verbundes:** Der Ablauf eines natürlichen Verbundes enthält Kombinationen von Originaltupeln, weswegen eine Umsetzung in einer Provenance-unterstützten Analyse zur Steigerung der Privacy sinnlos ist. Mittels einer ID-basierten Provenance-Notation und der in Abschnitt 4.7 definierten Provenance-Anfrage für Verbünde lassen sich aber dennoch die relevanten Teile der Ausgangsrelationen ermitteln. Die Veröffentlichung von Q, D und Q(D) erlauben dann eine gesteigerte Nachvollziehbarkeit, die allerdings mit einer Verletzung der Privacy einhergeht.

- **Ablauf einer Aggregation:** Der Ablauf einer Aggregation beginnt mit der ID-basierten Permutation. Die Auswertungsanfrage Q liefert dann das Ergebnis Q(D), also alle IDs der Ausgangsrelation mit einer angehängten Spalte, in der für jedes Tupel der Wert der Aggregatfunktion steht. Da bei einer Aggregation der gesamte Datensatz D den relevanten Teil darstellt, ist keine Provenance-Anfrage nötig. Die Generalisierung von D liefert also die intensionale Provenance-Antwort **D'** und ist für beide Ansätze (Svacina und Lamster) demnach gleich. Eine erneute ID-basierte Permutation muss nach der Generalisierung nicht vorgenommen werden, da keine Rückschlüsse von Q(D) auf **D'** bei normalen Werteverteilungen möglich sind. Eine Anpassung der Auswertungsanfrage Q erlaubt dann die Möglichkeit zur Berechnung des generalisierten Ergebnisses Q(D'(**D'**). Bei der Aggregation entfällt der Schritt der Generalisierung von Q(D) zu dessen Validierung, da eine Aggregation im Falle von COUNT denselben Wert und im Fall der anderen Funktionen SUM, AVG, MIN und MAX jeweils ein Intervall im generalisierten Ergebnis liefert, in denen der Wert in Q(D) enthalten ist. Entsprechend wird für die Nachvollziehbarkeit einer Aggregation nur die Auswertungsanfrage Q', das Ergebnis Q(D) und die intensionale Provenance-Antwort benötigt. Ist die Funktion SUM, AVG, MIN oder MAX, so lässt sich das Ergebnis mittels $\geq$ *spalteUG* und $\leq$ *spalteOG* validieren. Das Ergebnis von COUNT lässt sich entsprechend mit dem Vergleichsoperator = nachvollziehen.

Doch warum ist eine Ausführung besagter Aggregatfunktionen auf Intervallen überhaupt möglich? Sind die Intervalle wie in Abschnitt 4.4 für Attribute vom Typ 1 so gewählt, dass diese sich nicht überlappen und eine Monotonie aufweisen, so können entsprechende Aggregatfunktionen auf den Ober- und Untergrenzen ausgeführt werden. Dass der größte Attributwert entsprechend auch im größten Intervall jeder Stufe einer Konzepthierarchie liegt und somit auch die größte Untergrenze und die größte Obergrenze besitzt ist unschwer nachvollziehbar, was die Funktion MAX auf generalisierten Datensätzen möglich macht. Für die Funktion MIN gilt das analog für den niedrigsten Attributwert und die niedrigsten Grenzen. Aber warum liegen die Ergebnisse für SUM und AVG immer innerhalb der generalisierten Grenzen und sind dadurch nachvollziehbar? Die Lösung liegt dabei in der Monotonie der Intervalle und der Bedingung, dass diese sich außerdem nicht überlappen dürfen. Für jeden Attributwert x kommt somit nur ein Untergrenze $u \leq x$ und eine Obergrenze $o \geq x$ in Frage. Unter der Annahme, dass die Summe aller Attributwerte $x_1, ..., x_n$ mit $n \in \mathbb{N}$ durch die Bedingung $\forall x_i$ mit $i \in \mathbb{N} : u_i \leq x_i$, so gilt auch dass die Summe der Untergrenzen $u_1, ..., u_n$ für dasselbe n kleiner sein muss als die Summe von $x_1, ..., x_n$. Analog gilt die Betrachtung für die Obergrenze. Auch bei der Funktion AVG wird die Monotonie und die nicht vorhandene Überlappung der Intervalle ausgenutzt. Sei x ein Attributwert und $u \leq x$ die dazugehörige Untergrenze und $o \geq x$ die dazugehörige Obergrenze, so ist der Mittelwert aller Untergrenzen $u_1, ..., u_n$ kleiner als der Mittelwert der Attributwerte $x_1, ..., x_n$ mit $n \in \mathbb{N}$, wenn $\forall x_i$ mit $i \in \mathbb{N} : u_i \leq x_i$, was durch die Definition der Intervallgrenzen wahr ist. Entsprechend gilt diese Betrachtung auch für die Obergrenze, jedoch mit $\geq$.

Aus den Betrachtungen zu den einzelnen Standardoperationen ist zusammengefasst erkennbar, dass die intensionale Provenance-Antwort für eine Projektion und Aggregation direkt mittels Generalisierung der Ausgangsrelation berechnet werden kann und die Ausführung einer Provenance-Anfrage für diese Operationen demnach überflüssig ist. Gleichfalls ist erkennbar, dass eine Selektion und ein natürlicher Verbund in isolierter Form nicht der Privacy entsprechen, aber durch zusätzliche

Provenance-Informationen auch in einer koventionellen Datenanalyse nachvollziehbarer gestaltet werden können.

Da die Selektion und der natürliche Verbund bei einer Ausführung in Kombination mit einer Aggregation oder Projektion eher mit dem Konzept der Privacy vereinbar sind, werden auch an dieser Stelle wieder die einzelnen Kombinationen der Standardoperatoren zur Formulierung einer komplexeren Auswertungsanfrage in einem Gesamtablauf einer Provenance-unterstützten Datenanalyse als Eingabe betrachtet.

- **Ablauf SP:** Im Gegensatz zu einer einzelnen Selektion ist für eine Kombination aus Selektion und Projektion eine Provenance-unterstützte Datenanalyse möglich. Die Ausgangsrelation innerhalb des Datensatzes wird zuerst permutiert und die Projektionsbedingung in der Auswertungsanfrage Q entsprechend um die ID-Spalte erweitert. Die Auswertung der Anfrage Q liefert dann das Anfrageergebnis Q(D). Für die Bestimmung des relevanten Teildatensatzes **D** kann danach eine Provenance-Anfrage ausgeführt werden. Die Ausnutzung der Selektionsbedingung für die Bestimmung von **D** ist jedoch ressourcensparender, weil die Projektion die ID-Spalte nicht weiter verändert. Eine anschließende Generalisierung des relevanten Teildatensatzes liefert dann die intensionale Provenance-Antwort **D'** nach Svacina, die auch nur solche Tupel in generalisierter Form enthält, deren projizierte Teile auch im Ergebnis Q(D) vorkommen. In diesem Fall ist es wichtig zu beachten, dass nach der Generierung von **D'** wieder eine neue ID-Generierung und Permutation vorgenommen werden sollte, da sonst Teile des Ergebnisses Q(D) in **D'** wieder eingesetzt werden können. Die an die Generalisierung angepasste Auswertungsanfrage Q' liefert dann das generalisierte Ergebnis Q'(**D'**). Das Ergebnis der Generalisierung von Q(D) ist dann gleich dem Ergebnis Q'(**D'**), was eine Nachvollziehbarkeit und Validierung von Q(D) ermöglicht. Eine SP-Anfrage in Kombination mit der Generalisierung ist entsprechend kommutativ.

  Wird die intensionale Provenance-Antwort jedoch nach Lamster berechnet, so entfällt der Schritt der Provenance-Anfrage P. Der gesamte Datensatz D wird als erstes ID-basiert permutiert und dann generalisiert. Der generalisierte Datensatz D' geht dann in die angepasste Auswertungsanfrage Q' ein. Durch die in Q' generalisierte Selektionsbedingung kann das Ergebnis Q'(**D'**) entsprechend zusätzliche irrelevante Tupel enthalten, die im Vergleich zum Svacina-Ansatz die Kommutativität der Anfrage und der Generalisierung verletzen. Die Nachvollziehbarkeit wird entsprechend auf eine Teilmengenbeziehung bei Generalisierung von Q(D) reduziert, sodass Generalisierung(Q(D)) $\subset$ Q'(**D'**), aber nicht mehr gleich ist. Im Ansatz nach Lamster wird erst anschließend durch eine Provenance-Anfrage P' die intensionale Provenance-Antwort **D'** berechnet, in diesem Fall reicht allerdings auch wieder die angepasste Selektionsbedingung aus Q' dafür aus. Die intensionale Provenance-Antwort **D'** enthält im Vergleich zum Svacina-Ansatz auch wieder mehr irrelevante Tupel, was der Privacy z.B. bei sehr spezifischen Selektionen zu Gute kommen kann. Dafür wird die Nachvollziehbarkeit jedoch auf eine Teilmengenbeziehung abgeschwächt.

- **Ablauf SA:** Eine Provenance-unterstützte Datenanalyse für eine Kombination aus Selektion und Aggregation beginnt mit einer ID-basierten Permutation der Ausgangsrelation. Anschließend wird die Anfrage Q ausgewertet, um das Ergebnis Q(D) zu erhalten. Eine Provenance-Anfrage P auf die im Ergebnis enthaltenen IDs liefert dann den relevanten Datensatzteil **D**, der dann zur intensionalen Provenance-Antwort **D'** generalisiert wird. Die Generierung einer neuen ID und eine weitere ID-basierte Permutation sind unnötig, da außer dem Aggregationsergebnis, welches in jedem Ergebnistupel steht, keine anderen Rückschlüsse möglich sind. Die angepasste Auswertungsanfrage Q') liefert dann das generalisierte Ergebnis Q'(**D'**), welches im Fall von COUNT dasselbe Ergebnis ist wie das in Q(D). In allen anderen Fällen kann das Ergebnis Q(D) durch Vergleichsoperatoren validiert werden, wie schon bei der Aggregation beschrieben. Eine Generalisierung des Ergebnisses Q(D) ist deshalb auch nicht notwendig und würde nur die Nachvollziehbarkeit gefährden. Eine

Kommutativität von Generalisierung und Auswertungsanfrage ist an dieser Stelle somit gar nicht erwünscht, aber auch nicht schädlich für die Nachvollziehbarkeit. Der geschilderte Ablauf entspricht dabei dem Ansatz nach Svacina. Eine Umsetzung nach Lamster ist für SA-Anfragen nicht möglich, da die Generalisierung der Selektionsbedingung ohne vorherige Bestimmung der relevanten Tupel auch irrelevante Tupel in die Aggregation mit einbezieht, was das Ergebnis verfälschen und die Nachvollziehbarkeit gefährden kann.

- **Ablauf SJ:** Wie auch eine einzelne Selektion und ein einzelner natürlicher Verbund nicht in einer Provenance-unterstützten Datenanalyse realisiert werden können, so auch nicht deren Kombination, da Originaltupelkombinationen im Ergebnis gegen grundlegende Privacy-Aspekte verstoßen. Um eine solche Anfrage in einer konventionelle Datenanalyse nachvollziehbarer zu gestalten, können aber auch an dieser Stelle Provenance-Informationen im Ergebnis bereitgestellt werden. Mit einer dazu entsprechenden Provenance-Anfrage P kann dann der relevante Teildatensatz D ermittelt werden, der für die Nachvollziehbarkeit mindestens veröffentlicht werden muss.

- **Ablauf PJ/SPJ:** Wie auch bei der Selektion kann die Projektion genutzt werden um einen natürlichen Verbund privater zu gestalten und eine Umsetzung in einer Provenance-unterstützten Datenanalyse zu erlauben. Die Ausgangsrelationen im Datensatz D werden entsprechend ID-basiert permutiert. Anschließend wird die Anfrage Q ausgewertet, um das Ergebnis Q(D) zu erhalten. Um den Ansatz von Svacina umzusetzen, kann nun eine Provenance-Anfrage ausgeführt werden, deren Ergebnis anschließend generalisiert und permutiert wird (2 Relationen). Diese beiden generalisierten relevanten Teilrelationen stellen dann die intensionale Provenance-Antwort dar. Die Anpassung an die Generalisierung in Q' bewirkt, sofern auch Verbundattribute generalisiert wurden, dass im Ergebnis zusätzliche Tupel vorhanden sein können. Im Ansatz nach Svacina kann dies im schlimmsten Fall das Ausmaß eines Kreuzprodukts der relevaten Teilrelationen annehmen. Eine Nachvollziehbarkeit von Q(D) ist entsprechend nur durch eine Teilmengenbeziehung zu Q'(**D'**) nach der Generalisierung möglich. Der Ansatz nach Lamster unterscheidet sich darin, dass das Ausmaß des generalisierten Ergebnisses im schlimmsten Fall das Kreuzprodukt aller Ausgangstupel annehmen kann. Dies geschieht durch die primäre Generalisierung nach der ID-basierten Permutation und der anschließenden Auswertung von Q', bei der nicht nur die relevanten Tupel, sondern alle Tupel mit einbezogen werden. Eine zusätzliche Selektion sorgt in den Teilschritten für weitere Anpassungen und beschränkt weiterhin die Anzahl der relevanten Tupel, der Ablauf der Datenanalyse verläuft aber gleich. Eine Kommutativität solcher Anfragen und der Generalisierung ist entsprechend nicht gegeben und eine Nachvollziehbarkeit ist nur durch eine Teilmengenbeziehung möglich.

- **Ablauf JA/SJA:** Da ein natürlicher Verbund auf generalisierten Daten mit generalisierten Verbundattributen in der Auswertungsanfrage Q' zusätzliche Tupel durch den Verbund generiert, kann das Aggregationsergebnis entsprechend verfälscht werden. Diese Problem wird auch als JoinAgg-Problem bezeichnet. Es lässt sich aber z.B. dadurch lösen, indem das Ergebnis des natürlichen Verbundes als neue Ausgangsrelation materialisiert wird und die Anfragen anschließend um den natürlichen Verbund erleichtert werden. Die alternativen ausführbaren Anfragen in einer Provenance-unterstützte Datenanalyse sind demnach wieder eine A- bzw. SA-Anfrage.

- **Ablauf PA/SPA/PJA/SPJA:** Ein Ablauf für diese Anfragetypen lässt sich aufgrund des ProjAgg-Problems nicht generalisieren. Alternative Anfragen ohne Projektion sind demzufolge A, SA und SJA, wobei SJA-Anfragen wieder dem JoinAgg-Problem unterliegen und ebenfalls auf SA reduziert werden müssen.

Es ist also möglich, eine Projektions-, Aggregations-, SP-, SA-, PJ- und SPJ-Anfrage als Eingabe einer Provenance-unterstützten Datenanalyse zur Steigerung der Privatsphäre zu übergeben. Die Tabellen 4.7

und 4.8 fassen noch einmal die bis hierher gewonnenen Erkenntnisse zur Umsetzbarkeit, Privacy, den möglichen Ansätzen und Alternativen noch einmal zusammen.

Anfrage	S	P	J	A	SP	SJ	SA	PJ	PA	JA	SPJ	SPA	SJA	PJA	SPJA
Umsetzbarkeit	✓	✓	✓	✓	✓	✓	✓	✓	X	X	✓	X	X	X	X
Privacy	X	✓	X	✓	✓	X	✓	✓			✓				
Ansatz		Sv=L		Sv=L	Sv/L		Sv	Sv/L			Sv/L				
Problem	O		O			O			PA	JA		PA	JA	PA/JA	PA/JA
Alternative									A	A		SA	SA	A	SA

Tabelle 4.7.: Übersicht komplexerer Auswertungsanfragen

Abkürzung	Bedeutung
Ansatz Sv	Umsetzung nach Ansatz Svacina
Ansatz Sv=L	Umsetzung nach Ansatz Svacina und Lamster gleich
Ansatz Sv/L	Umsetzung nach Ansatz Svacina oder Lamster möglich
Problem O	Privacy-Problem: Ergebnis enthält Originaltupel
Problem PA	Projektion in Kombination mit Aggregation nicht ausführbar
Problem JA	Natürlicher Verbund nach Lamster oder Svacina keine Grundlage für nachvollziehbare Aggregation
Problem PA/JA	**PA** und **JA**

Tabelle 4.8.: Übersicht komplexerer Auswertungsanfragen

5. Umsetzung und Implementierung

In diesem Kapitel erfolgt die Vorstellung der Programmierung, die die theoretischen Überlegungen der letzten Kapitel implementiert und die Ausführung einer Provenance-unterstützten Datenanalyse auf Basis einer konventionellen Datenanalyse zur Steigerung der Privacy ermöglicht. Als erstes erläutert Abschnitt 5.1 die Wahl der Programmiersprache und des Datenbankmanagementsystems. Daraufhin wird in Abschnitt 5.2 der Aufbau des Programms und ein resultierender Programmablauf beschrieben. Abschnitt 5.3 behandelt anschließend die vom Prototypen akzeptierten Eingaben und die entsprechend zu setzenden Argumente. Der letzte Abschnitt 5.4 verweist dann nur noch auf den Programmcode.

5.1. Programmiersprache und Datenbankmanagementsystem

Das Konzept für eine Umsetzung einer Provenance-unterstützten Datenanalyse in Kapitel 4 wurde mit Absicht so formuliert, dass eine Implementierung mit verschiedenen Programmiersprachen und verschiedenen Datenbankmanagementsysteme vorgenommen werden kann. Die einzige Anforderung ist die Möglichkeit zur Kommunikation beider Teile miteinander, eine ausreichende Mächtigkeit der Programmiersprache und ein Datenbankmanagementsystem, das die notwendigen Operationen auch ausführen kann.

Für eine prototypische Umsetzung des Konzeptes fiel die Wahl dabei auf die Programmiersprache **Python** und das Datenbankmanagementsystem **SQLite**. Python erfüllt die zuvor genannten Anforderungen. Zum einen handelt es sich um eine Script-basierte Programmiersprache, die eine Aufteilung der einzelnen Anwendungsfälle in verschiedene Scripte ermöglicht und eine entsprechend einfache Möglichkeit zur Generierung eines Prototyps darstellt. Auch beinhaltet Python alle nötigen Schnittstellen, darunter auch eine Anbindung zu SQLite mit umfangreicher Funktionalität. Außerdem ist Python für alle geläufigen Betriebssysteme verfügbar. SQLlite ist ein einfach zu benutzendes Datenbankmanagementsystem, das die benötigten Operationen zur Umsetzung des Konzepts enthält. Die einfache Anbindung einer SQLite-Datenbank innerhalb eines Python-Scripts macht SQLite zur perfekten Ergänzung für eine prototypische Umsetzung.

5.2. Programmaufbau und -ablauf

Das Programm ist zur besseren Übersichtlichkeit in 8 Scripte unterteilt. An dieser Stelle erfolgt eine Beschreibung der Funktion der einzelnen Scripte.

- **main.py** In diesem Script sind die Argumente definiert, die für eine Ausführung gesetzt werden müssen. Die Angabe der verschiedenen Argumente variiert je nach Anwendung. Durch Fallunterscheidungen mittels dieser Argumente erfolgt dann die Ausführung des zu den Argumenten passenden Scripts.

- **example.py** Dieses Script kann aufgerufen werden, um einen Datensatz entsprechend des Studentenbeispiels aus Abschnitt 1.3 in einer SQLite-Datenbank zu erstellen. Die Generierung umfasst neben den Tabellen `NOTEN` und `STUDIERENDE` auch die konzeptkonformen Konzeptrelationen für jedes Attribut in einer zusätzlichen Tabelle.
- **projection.py:** Dieses Script enthält die Umsetzung einer Projektion in einer Provenance-unterstützten Datenanalyse. Es wird das Anfrageergebnis Q(D), die intensionale Provenance-Antwort **D'** und das Ergebnis von Q' als Tabelle in der übergebenen SQLite-Datenbank erzeugt.
- **aggregation.py:** Dieses Script enthält die Umsetzung einer Aggregation für die Funktionen SUM, AVG, MIN, MAX und COUNT. Neben den Anfrageergebnissen wird eine zusätzliche Provenance-Tabelle der Ergebnisse im Ergebnis berechnet. Die Ausgabe entspricht also 5 anstatt 3 Tabellen.
- **sp.py** Dieses Script enthält die Umsetzung einer Projektion mit dem Zusatz der Selektion, bestehend aus Attributname, Vergleichsoperator und zu selektierendem Attributwert. Nach der Ausführung enthält die übergebene Datenbank wieder das Anfrageergebnis Q(D), die intensionale Provenance-Antwort **D'** und das Ergebnis der angepassten Anfrage Q'.
- **sa.py** Dieses Script enthält die Umsetzung einer SA-Anfrage für eine Provenance-unterstützte Datenanalyse. Ähnlich zur Aggregation werden zwei zusätzliche Tabellen für das Aggregationsergebnis erstellt.
- **jp.py** Dieses Script enthält die Umsetzung einer JP-Anfrage. Neben den bekannten Tabellen für Q(D) und dem Anfrageergebnis von Q' werden für die beiden Verbundtabellen auch zwei Tabellen in der intensionalen Provenance-Antwort generiert.
- **sjp.py** Das letzte Script vereinigt die Umsetzungen von Selektion, natürlichem Verbund und Projektion. Die Ausgabe enthält entsprechend eine Tabelle für Q(D) und Q'(**D'**), sowie zwei weitere Tabellen der intensionalen Provenance-Antwort.

Die setzbaren Argumente bei Start von main.py sind die folgenden:

- **m mode:** Dieses Argument muss immer gesetzt werden und enthält den Typ der Auswertungsanfrage, die als Eingabe definiert wird. Entsprechend kann dieses Argument nur einen der Werte P, A, SP, SA, JP oder SJP annehmen.
- **d database:** Dieses Argument muss auf den Pfad der für die Analyse genutzten SQLite-Datenbankdatei gesetzt werden.
- **t table:** An dieser Stelle muss immer ein Relationenname gesetzt werden, der in der übergebenen Datenbank vorkommt und in der Auswertungsfrage hinter dem `FROM` gesetzt wird.
- **sa selection_attribute:** Sofern eine Selektion in der Anfrage vorkommt, muss dieses Argument auf den Attributnamen der Selektion gesetzt werden.
- **so selection_operator:** Dieses Argument muss auf einen der Vergleichsoperatoren $<, \leq, ==, \geq, >$ gesetzt werden, sofern die Anfrage eine Selektion enthält.
- **sv selection_value:** Für dieses Argument muss der Attributwert gesetzt werden, der in der Selektionsbedingung hinter dem Vergleichsoperator steht.
- **p projection:** Dieses Argument kann eine Liste von Attributnamen annehmen, auf die projiziert werden soll. Die einzelnen Attribute werden dabei mit einem Space voneinander getrennt.
- **j natural_join** Enthält die Anfrage einen natürlichen Verbund, so muss in diesem Argument der Relationenname der zweiten Relation für einen Verbund angegeben sein.

- **af aggregate_function** Ist eine Aggregation Teil der Anfrage, wird hier die Aggregatfunktion definiert. Diese kann eine der Funktionen AVG, SUM, MIN, MAX oder COUNT sein.
- **aa aggregate_attribute** Dieses Argument muss auf den Attributnamen gesetzt werden, der in der Aggregation definiert ist. Ist das Argument -af allerdings COUNT, so darf dieses Argument nicht gesetzt sein.

Mithilfe dieser Argumente wird entschieden, welches der Scripte für die Umsetzung einer bestimmten Anfrage ausgeführt wird. Ein gesamter Programmablauf wird demnach durch den Aufruf von main.py unter der Angabe benötigter Argumente in Gang gesetzt. Begonnen wird entsprechend auch mit der Ausführung vom Script main.py, das eine optionale Ausführung von example.py enthält und mit der Ausführung eines den Argumenten entsprechenden Anfrage-Scripts endet.

5.3. Studentenbeispiel und Anfragen

Anstatt die Implementierung ausführlich zu erklären, soll an dieser Stelle eine Vorstellung der ausführbaren Anfragen und der zu setzenden Parameter erfolgen. Der Programmcode ist zudem gut kommentiert und wurde genau nach Konzept implementiert. Neben den möglichen Anfragen werden noch getestete Argumentkombinationen aufgezählt, die auch für das Studentenbeispiel erstellt wurden, um den Funktionsumfang des Prototyps zu testen. Nun soll mit der Aufzählung der im Prototyp umsetzbaren Auswertungsanfragen und den benötigten Argumenten in Tabelle 5.1 und 5.2 begonnen werden.

Anfrage	Argumente	Bemerkung
P	-m -d -t -p	
A	-m -d -t -af -aa	-aa nicht, wenn -af COUNT
SP	-m -d -t -p -sa -so -sv	
SA	-m -d -t -af -aa -sa -so -sv	-aa nicht, wenn -af COUNT
JP	-m -d -t -p -j	
SJP	-m -d -t -p -j -sa -so -sv	

Tabelle 5.1.: Übersicht implementierter Auswertungsanfragen

Args	Anleitung	Bemerkung
-m	Name der Anfrage	Auswahl: P,A,SP,SA,JP,SJP
-d	Pfad zur Datenbank	nur ein Pfad möglich
-t	Name der Tabelle hinter `FROM`	nur eine Tabelle möglich
-p	Attributnamen für Projektion	mehrere mit Space trennen
-j	Name der Tabelle hinter `NATURAL JOIN`	nur eine Tabelle möglich
-af	Aggregationsfunktion	Auswahl: AVG, SUM, MIN, MAX, COUNT
-aa	Attribut der Aggregation	nur ein Attribut
-sa	Attribut der Selektion	nur ein Attribut
-so	Vergleichsoperator der Aggregation	Auswahl: $<, <=, ==, >=, >$
-sv	Attributwert der Selektion	nur ein Attributwert

Tabelle 5.2.: Übersicht der Argumente

Die im Prototyp ausführbaren Anfragen entsprechen also den umsetzbaren und Privacy-konformen Anfragetypen aus Tabelle 4.7. Mit den vorgestellten Argumenten lassen sich eine Vielzahl verschiedener Anfragen für jeden Typ formulieren. Einschränkungen des Konzeptes gibt es bis auf die Definition einer einzelnen Selektionsbedingung dabei nicht. Um den Funktionsumfang zu testen, wurden weitere Auswertungsanfragen für das Studentenbeispiel formuliert, die in Tabelle 5.3 durch die jeweiligen Argumente angegeben sind.

Getestete Argumentkombinationen
-m P -d -t STUDIERENDE -p studiengang
-m P -d -t NOTEN -p note
-m A -d -t STUDIERENDE -af COUNT
-m A -d -t NOTEN -af COUNT
-m A -d -t NOTEN -af SUM -aa note
-m A -d -t NOTEN -af AVG -aa note
-m A -d -t NOTEN -af MIN -aa note
-m A -d -t NOTEN -af MAX -aa note
-m SP -d -t NOTEN -p semester note -sa note -so < -sv 2.0
-m SP -d -t NOTEN -p semester note -sa note -so = -sv 1.0
-m SP -d -t NOTEN -p semester note -sa semester -so == -sv WS 15/16
-m SP -d -t NOTEN -p note -sa note -so > -sv 3.0
-m SP -d -t STUDIERENDE -p student_id -sa studiengang -so == -sv Mathematik
-m SA -d -t NOTEN -af AVG -aa note -sa note -so < -sv 3.0
-m SA -d -t NOTEN -af MIN -aa note -sa note -so >= -sv 1.0
-m SA -d -t NOTEN -af SUM -aa note -sa note -so <= -sv 3.0
-m SA -d -t NOTEN -af COUNT -sa note -so < -sv 3.0
-m SA -d -t STUDIERENDE -af COUNT -sa studiengang -so = -sv Informatik
-m JP -d -t STUDIERENDE -j NOTEN -p studiengang semester note
-m JP -d -t STUDIERENDE -j NOTEN -p studiengang
-m SJP -d -t STUDIERENDE -j NOTEN -p studiengang -sa note -so < -sv 3.0
-m SJP -d -t STUDIERENDE -j NOTEN -p vorname nachname note -sa note -so == -sv 1.0

Tabelle 5.3.: Getestete Argumentkombinationen

Für das Argument -d wurde immer ein Pfad angegeben, der auf eine leere SQLite-Datenbankdatei verweist. Das Script example.py sorgte dann für die Erstellung der notwendigen Tabellen des Studentenbeispiels und der Konzeptrelationen für alle Attribute. Die Argumentkombination wurden so gewählt, dass für die Argumente, die eine Auswahl verschiedener Werte anbieten, auch jeder mögliche Wert getestet wurde.

Zur besseren Veranschaulichung wird an dieser Stelle die Ausgabe für die letzte Argumentkombination aus Tabelle 5.3 in Abbildung 5.1 gezeigt. Die aus den Parametern ableitbare SQL-Anfrage lautet "`SELECT vorname,nachname,note FROM STUDIERENDE NATURAL JOIN NOTEN WHERE note = 1.0`".

```
Creating example dataset.
Regular result SJP_QofD:
[('Anne', 'Altmann', 1.0, 5724698925010316547, -6812372115592583316)]
Generalized result SJP_genQofD:
[('x', 'x', 1.0, 1.5, 3665911636702759906, -9091564041974166302)]
Intensional provenance-answer SJP_gen_id_STUDIERENDE and SJP_gen_id_NOTEN:
[('x', 'x', 'x', 'Science', 3665911636702759906)]
[('x', 'x', 'WS', 1.0, 1.5, -9091564041974166302)]
```

Abbildung 5.1.: Ausgabe des Prototypen

Die Ausgabe enthält also das Ergebnis Q(D), die intensionale Provenance-Antwort **D'** und das generalisierte Ergebnis der Auswertungsanfrage Q'. Die dazugehörigen Tabellen lauten SJP_QofD, SJP_gen_id_STUDIERENDE und SJP_gen_id_NOTEN als intensionale Provenance-Antwort und SJP_genQofD für das Ergebnis von Q'. Zusätzlich wird in der Ausgabe die angepasste Auswertungsanfrage Q' angegeben. Diese war in der Abbildung für das Beispiel jedoch zu lang und wurde deswegen herausgeschnitten.

Die prototypische Implementierung ist nur für den Ansatz nach Svacina vorgenommen worden, da der

Ansatz nach Lamster nur für SP-, PJ- und SPJ-Anfragen ein potentiell unterschiedliches Ergebnis liefert bzw. umsetzbar ist. Eine Umsetzung nach Svacina ist als Proof of Concept jedoch ausreichend und eine Anpassung für den Ansatz nach Lamster ist leicht vorzunehmen. Durch die Funktionalität der Svacina-Implementierung und die leichten Abweichungen im Lamster-Ansatz sollte eine solche Umsetzung ebenfalls funktionieren.

5.4. Implementierung und GIT

Der Programmcode der prototypischen Implementierung ist in Anhang A aufgelistet. Für den Programmcode wurde aber zusätzlich ein Repository auf GitHub angelegt. Für weitere Informationen siehe Anhang B.2.

6. Fazit und Ausblick

In diesem Abschnitt soll abschließend ein Fazit gezogen werden, das nochmal die Erkenntnisse dieser Arbeit zusammenfasst. Zu diesem Zweck enthält Abschnitt 6.1 das besagte Fazit und Abschnitt 6.2 gibt danach noch einen Ausblick auf mögliche Forschungsthemen in diesem Bereich an.

6.1. Fazit

Betrachtungen von Vorarbeiten und auch dieser Arbeit haben ergeben, dass eine Provenance-Notation nicht immer sinnvoll für die Einhaltung der Privacy ist. Ebenfalls wurde herausgestellt, dass auch andere Teile einer Datenanalyse, wie z.B. die Auswertungsanfrage Q oder der relevante Datensatzteil **D** bei Veröffentlichung ein Privacy-Risiko darstellen können. Um den relevanten Teil einer Datenbank zu ermitteln kann eine Provenance-Anfrage genutzt werden, die, wenn sie intensional beantwortet wird, auch den Privacy-Ansprüchen genügt. Eine intensionale Beantwortung von Provenance-Anfragen wurde auf Basis der Generalisierung untersucht, wie sie in [Sva16] konzipiert wurde. Da eine Umsetzung vorher aber noch nicht erfolgt ist und das Konzept auf der Nutzung eines Tools basiert, wurde dieses Konzept für eine toolunabhängige, relationale Umsetzung angepasst. Da ein anonymisierter bzw. generalisierter Datensatz nicht mehr alleine für die Nachvollziehbarkeit und Reproduzierbarkeit eines Ergebnisses ausreicht, wurde ein Ablauf einer Provenance-unterstützten Datenanalyse konzipiert, der sowohl für die Nachvollziehbarkeit als auch die Erhaltung der Privacy sorgen kann. Für diese neue Datenanalyse wurde die Umsetzbarkeit verschiedener Auswertungsanfragen untersucht. Die Auswertungsanfragen sind die Standardoperationen Selektion, Projektion, der natürliche Verbund und die Aggregation (SUM, AVG, MIN, MAX und COUNT) sowie deren Kombinationen. Durch die festgesetzte Notation in dieser Arbeit erwiesen sich jedoch nur sechs Anfragen als umsetzbar und Privacy-konform. Diese sind P-, A-, SP-, SA-, JP- und SJP-Anfragen. Eine intensionale Beantwortung der Provenance-Anfrage erforderte ebenfalls eine Anpassung der Auswertungsanfrage. Eine minimale Konzeptrelation mit Beachtung von Attributtypen erlaubte dann eine relationale, toolunabhängige Umsetzung dieser Generalisierung. Die Ergebnisse von P-, SP-, JP- und SJP-Anfragen mussten für eine Nachvollziehbarkeit ebenfalls generalisiert und dann mit dem anonymen Ergebnis der angepassten Auswertungsanfrage verglichen werden. Die Ergebnisse von A- und SA-Anfragen benötigten für eine Nachvollziehbarkeit hingegen keine Anonymisierung. Eine Nachvollziehbarkeit durch die Äquivalenz der Ergebnisse ist für die Anfragen P und SP (nach Svacina) möglich. Eine Nachvollziehbarkeit durch eine Teilmengenbeziehung ist für die Operationen SP (nach Lamster), JP und SJP möglich. Aggregationsergebnisse der Anfragen A und SA lassen sich hingegen durch einfache Vergleiche mittels Vergleichsoperatoren validieren.

Eine Implementierung dieser Konzepte hat ergeben, dass eine Provenance-unterstützte Datenanalyse auf Basis einer konventionellen Datenanalyse umgesetzt werden kann, sofern diese als Eingabe eine der kompatiblen Anfragen und im Datensatz die entsprechenden Konzeptrelationen für eine Generalisierung enthielt. Die relationale Formulierung einer solchen Datenanalyse mittels ID-basierter Provenance, Generalisierung via Konzeptrelationen und der Anpassung eines konventionellen Ablaufs für eine gesteigerte Privacy unter Erhalt zumindest abgeschwächter Nachvollziehbarkeit zeigte, dass eine solche Umsetzung

die Privacy erhöhen kann. Die prototypische Implementierung erfolgte dabei mittels Python und SQLite.

6.2. Ausblick

Nach den durchgeführten Betrachtungen stellt sich vor allem die Frage, wie sich auch bisher nicht umsetzbare Auswertungsanfragen in dieser neuen Art der Datenanalyse realisieren lassen. Eine Möglichkeit zur Behebung des ProjAgg-Problems, welches nur durch die feste Formulierung der Aggregation entstanden ist, wäre z.B. die Erweiterung von SPJA-Anfragen um das Statement `GROUP BY`, das oft zur Kombination von Aggregations und Projektionsoperationen genutzt wird. Solche Erweiterungen müssen natürlich auch immer auf die Privacy untersucht werden. Für weitere Operationen stellt sich also immer die Frage der Umsetzbarkeit und Privacy.

Eine weitere Frage ist, wie mit den Auswertungsanfragen umgegangen werden kann, die von sich aus gegen die Privacy verstoßen. Als Beispiel hierfür sind noch einmal die Anfragen S und SJ genannt. Da die Ergebnisse dieser Anfragen große Teile originaler Daten enthalten, muss für diese ein anderer Weg der Nachvollziehbarkeit unter Privacy-Aspekten entwickelt werden.

Eine weitere Frage ist die mögliche Anwendung im Bereich Big Data. Besonders kritisch sind hierbei die großen Datenmengen anzusehen, deren Inhalt die unterschiedlichsten Formen haben kann. Entsprechend ist eine Anpassung der Konzepte an spezifischere Formen von Daten und Informationen interessant. Auch eine Optimierung und Anpassung des Konzepts für riesige Datenmengen ist nicht von der Hand zu weisen. Eine eventuelle Parallelisierung einer solchen Datenanalyse ist demnach ebenfalls ein weiteres interessantes Forschungsthema.

Literaturverzeichnis

[AL12] Alharbi, Khalid N. ; Lin, Xiaodong: PDP: A Privacy-Preserving Data Provenance Scheme. In: *32nd International Conference on Distributed Computing Systems Workshops (ICDCS 2012 Workshops), Macau, China, June 18-21, 2012*, IEEE Computer Society, 2012, 500–505

[ASH21] Auge, Tanja ; Scharlau, Nic ; Heuer, Andreas: Privacy Aspects of Provenance Queries. In: *CoRR* abs/2101.04432 (2021). `https://arxiv.org/abs/2101.04432`

[Aug17] Auge, Tanja: *Masterarbeit: Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen.* 2017. – Universität Rostock, Fakultät für Elektrotechnik und Informatik

[Aug20] Auge, Tanja: Extended Provenance Management for Data Science Applications. In: Abedjan, Ziawasch (Hrsg.) ; Hose, Katja (Hrsg.): *Proceedings of the VLDB 2020 PhD Workshop co-located with the 46th International Conference on Very Large Databases (VLDB 2020), ONLINE, August 31 - September 4, 2020* Bd. 2652, CEUR-WS.org, 2020 (CEUR Workshop Proceedings)

[BGH+06] Braun, Uri ; Garfinkel, Simson ; Holland, David A. ; Muniswamy-Reddy, Kiran-Kumar ; Seltzer, Margo I.: Issues in Automatic Provenance Collection. In: Moreau, Luc (Hrsg.) ; Foster, Ian (Hrsg.): *Provenance and Annotation of Data.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2006. – ISBN 978–3–540–46303–0, S. 171–183

[BKT01] Buneman, Peter ; Khanna, Sanjeev ; Tan, Wang C.: Why and Where: A Characterization of Data Provenance. In: *Proceedings of the 8th International Conference on Database Theory.* Berlin, Heidelberg : Springer-Verlag, 2001 (ICDT '01). – ISBN 3540414568, S. 316–330

[CCT09] Cheney, James ; Chiticariu, Laura ; Tan, Wang-Chiew: Provenance in Databases: Why, How, and Where. In: *Found. Trends Databases* 1 (2009), April, Nr. 4, 379–474. `http://dx.doi.org/10.1561/1900000006`. – DOI 10.1561/1900000006. – ISSN 1931–7883

[Che11] Cheney, James: A Formal Framework for Provenance Security. In: *Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium.* USA : IEEE Computer Society, 2011 (CSF '11). – ISBN 9780769543659, 281–293

[Cod70] Codd, Edgar F.: A Relational Model of Data for Large Shared Data Banks. In: *Communications of the ACM* 13 (1970), Juni, Nr. 6, 377-387. `http://dl.acm.org/citation.cfm?id=362685`

[CW00] Cui, Y. ; Widom, J.: Practical lineage tracing in data warehouses. In: *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*, 2000, S. 367–378

[CWW00] Cui, Yingwei ; Widom, Jennifer ; Wiener, Janet L.: Tracing the Lineage of View Data in a Warehousing Environment. In: *ACM Trans. Database Syst.* 25 (2000), Juni, Nr. 2, 179–227. `http://dx.doi.org/10.1145/357775.357777`. – DOI 10.1145/357775.357777. – ISSN 0362–5915

[CY20] Can, Özgü ; Yilmazer, Dilek: A novel approach to provenance management for privacy preservation. In: *J. Inf. Sci.* 46 (2020), Nr. 2. http://dx.doi.org/10.1177/0165551519827882. – DOI 10.1177/0165551519827882

[Dal86] Dalenius, Tore: Finding a needle in a haystack or identifying anonymous census records. In: *Journal of Official Statistics* 2 (1986), Nr. 3315-328. – serial sequence - indexed article

[DKM+06] Dwork, Cynthia ; Kenthapadi, Krishnaram ; McSherry, Frank ; Mironov, Ilya ; Naor, Moni: Our Data, Ourselves: Privacy Via Distributed Noise Generation, 2006. – ISBN 978–3–540–34546–6, S. 486–503

[DKR+11] Davidson, Susan B. ; Khanna, Sanjeev ; Roy, Sudeepa ; Stoyanovich, Julia ; Tannen, Val ; Chen, Yi: On provenance and privacy. In: Milo, Tova (Hrsg.): *Database Theory - ICDT 2011, 14th International Conference, Uppsala, Sweden, March 21-24, 2011, Proceedings*, ACM, 2011, 3–10

[DKT+11] Davidson, Susan B. ; Khanna, Sanjeev ; Tannen, Val ; Roy, Sudeepa ; Chen, Yi ; Milo, Tova ; Stoyanovich, Julia: Enabling Privacy in Provenance-Aware Workflow Systems. In: *Fifth Biennial Conference on Innovative Data Systems Research, CIDR 2011, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, www.cidrdb.org, 2011, 215–218

[Dwo06] Dwork, Cynthia: Differential Privacy. In: *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*. Berlin, Heidelberg : Springer-Verlag, 2006 (ICALP'06). – ISBN 3540359079, 1–12

[Dwo08] Dwork, Cynthia: Differential Privacy: A Survey of Results, 2008. – ISBN 978–3–540–79227–7, S. 1–19

[GH15] Grunert, Hannes ; Heuer, Andreas: Slicing in Assistenzsystemen - Wie trotz Anonymisierung von Daten wertvolle Analyseergebnisse gewonnen werden können. In: Saake, Gunter (Hrsg.) ; Broneske, David (Hrsg.) ; Dorok, Sebastian (Hrsg.) ; Meister, Andreas (Hrsg.): *Proceedings of the 27th GI-Workshop Grundlagen von Datenbanken, Gommern, Germany, May 26-29, 2015* Bd. 1366, CEUR-WS.org, 2015 (CEUR Workshop Proceedings), 24–29

[GKT07] Green, Todd J. ; Karvounarakis, Grigoris ; Tannen, Val: Provenance Semirings. In: *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA : Association for Computing Machinery, 2007 (PODS '07). – ISBN 9781595936851, 31–40

[HDBL17] Herschel, Melanie ; Diestelkämper, Ralf ; Ben Lahmar, Houssem: A Survey on Provenance: What for? What Form? What From? In: *The VLDB Journal* 26 (2017), Dezember, Nr. 6, 881–906. http://dx.doi.org/10.1007/s00778-017-0486-1. – DOI 10.1007/s00778–017–0486–1. – ISSN 1066–8888

[HSS18] Heuer, Andreas ; Saake, Gunter ; Sattler, Kai-Uwe: *Datenbanken - Konzepte und Sprachen, 6. Auflage*. MITP, 2018 https://mitp.de/IT-WEB/Datenbanken/Datenbanken-Konzepte-und-Sprachen-oxid.html. – ISBN 978–3–9584577–6–8

[Kle20] Klein, Eric: *Bachelorarbeit: Parallele Anonymisierung von großen Datenbeständen*. 2020. – Universität Rostock, Fakultät für Elektrotechnik und Informatik

[LLV07] Li, Ninghui ; Li, Tiancheng ; Venkatasubramanian, Suresh: t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In: *2007 IEEE 23rd International Conference on Data Engineering* (2007), S. 106–115

[LLZM12] Li, Tiancheng ; Li, Ninghui ; Zhang, Jian ; Molloy, Ian: Slicing: A New Approach for Privacy Preserving Data Publishing. In: *IEEE Transactions on Knowledge and Data Engineering* 24 (2012), Nr. 3, S. 561–574. http://dx.doi.org/10.1109/TKDE.2010.236. – DOI 10.1109/TKDE.2010.236

[MKGV07] Machanavajjhala, Ashwin ; Kifer, Daniel ; Gehrke, Johannes ; Venkitasubramaniam, Muthuramakrishnan: L-Diversity: Privacy beyond k-Anonymity. In: *ACM Trans. Knowl. Discov. Data* 1 (2007), März, Nr. 1, 3–es. http://dx.doi.org/10.1145/1217299.1217302. – DOI 10.1145/1217299.1217302. – ISSN 1556–4681

[Mot94] Motro, Amihai: Intensional answers to database queries. In: *IEEE Transactions on Knowledge and Data Engineering* 6 (1994), S. 444–454

[SBS18] Sanchez, Jose Luis C. ; Bernabé, Jorge B. ; Skarmeta, Antonio F.: Towards privacy preserving data provenance for the Internet of Things. In: *4th IEEE World Forum on Internet of Things, WF-IoT 2018, Singapore, February 5-8, 2018*, IEEE, 2018, 41–46

[Sch20] Scharlau, Nic: *Bachelorarbeit: Provenance und Privacy in ProSA*. 2020. – Universität Rostock, Fakultät für Elektrotechnik und Informatik

[Sva16] Svacina, Jan: *Bachelorarbeit: Intensional Answers for Provenance Queries in Big Data Analytics*. 2016. – Universität Rostock, Fakultät für Elektrotechnik und Informatik

[Swe00] Sweeney, Latanya: *Simple Demographics Often Identify People Uniquely*. Version: 2000. http://dataprivacylab.org/projects/identifiability/

[Swe02] Sweeney, Latanya: k-Anonymity: A Model for Protecting Privacy. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10 (2002), Nr. 5, 557-570. http://dblp.uni-trier.de/db/journals/ijufks/ijufks10.html#Sweene02

[WDA+16] Wilkinson, Mark D. ; Dumontier, Michel ; Aalbersberg, IJsbrand J. ; Appleton, Gabrielle ; Axton, Myles ; Baak, Arie ; Blomberg, Niklas ; Boiten, Jan-Willem ; Silva Santos, Luiz B. ; Bourne, Philip E. ; Bouwman, Jildau ; Brookes, Anthony J. ; Clark, Tim ; Crosas, Mercè ; Dillo, Ingrid ; Dumon, Olivier ; Edmunds, Scott ; Evelo, Chris T. ; Finkers, Richard ; Gonzalez-Beltran, Alejandra ; Gray, Alasdair J. ; Groth, Paul ; Goble, Carole ; Grethe, Jeffrey S. ; Heringa, Jaap ; Hoen, Peter A. ; Hooft, Rob ; Kuhn, Tobias ; Kok, Ruben ; Kok, Joost ; Lusher, Scott J. ; Martone, Maryann E. ; Mons, Albert ; Packer, Abel L. ; Persson, Bengt ; Rocca-Serra, Philippe ; Roos, Marco ; Schaik, Rene van ; Sansone, Susanna-Assunta ; Schultes, Erik ; Sengstag, Thierry ; Slater, Ted ; Strawn, George ; Swertz, Morris A. ; Thompson, Mark ; Lei, Johan van d. ; Mulligen, Erik van ; Velterop, Jan ; Waagmeester, Andra ; Wittenburg, Peter ; Wolstencroft, Katherine ; Zhao, Jun ; Mons, Barend: The FAIR Guiding Principles for scientific data management and stewardship. 3 (2016), Nr. 1, 160018. http://dx.doi.org/10.1038/sdata.2016.18. – DOI 10.1038/sdata.2016.18. – ISSN 2052–4463

[Web85] Webber, B.: Questions, Answers and Responses: Interacting with Knowledge-Base Systems. In: *On Knowledge Base Management Systems*, 1985

[WS97] Woodruff, A. ; Stonebraker, M.: Supporting fine-grained data lineage in a database visualization environment. In: *Proceedings 13th International Conference on Data Engineering*, 1997, S. 91–102

[YP99] Yoon, Suk-Chung ; Park, E.K.: An approach to intensional query answering at multiple abstraction levels using data mining approaches. In: *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers* Bd. Track6, 1999, S. 9 pp.–

Anfragenverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

A. Programmcode

A.1. main.py

```
import sqlite3

from example import create_example
from jp import jp
from projection import projection
from aggregation import aggregation
from sa import sa
from sjp import sjp
from sp import sp
import argparse



parser = argparse.ArgumentParser()

parser.add_argument("-m", "--mode", required=True, choices=["P", "A", "SP", "SA", "JP", "
    SJP"], type=str,
                    help="Select the type of query you wanna execute.")
parser.add_argument("-d", "--database", required=True, type=str,
                    help="Provide a SQLite file which will be used for the run. Original
                        data will be untouched.")
parser.add_argument("-t", "--table", type=str, required=True,
                    help="All modess: Name of the base relation defined in FROM-clause.")
parser.add_argument("-sa", "--selection_attribute", type=str, metavar='',
                    help="Mode SP, SA, SJP: Name of the attribute in the selection.")
parser.add_argument("-so", "--selection_operator", choices=["<", "<=", "==", ">=", ">"],
    metavar='', type=str,
                    help="Mode SP, SA, SJP: Comparison operator in the selection.")
parser.add_argument("-sv", "--selection_value", type=str, metavar='',
                    help="Mode SP, SA, SJP: Single value to compare with.")
parser.add_argument("-p", "--projection", type=str, nargs='+', metavar='',
                    help="Mode P, SP, JP, or SJP: List of the attributes to project,
                        separated by spaces.")
parser.add_argument("-j", "--natural_join", type=str,
                    help="Mode JP or SJP: Name of the second relation for a natural join
                        .")
parser.add_argument("-af", "--aggregate_function", choices=["AVG", "SUM", "MIN", "MAX", "
    COUNT"], metavar='', type=str,
                    help="Mode A or SA:. Name of the aggregation function.")
parser.add_argument("-aa", "--aggregate_attribute", type=str, metavar='',
                    help="Mode A or SA and -af not being COUNT: Name of the attribute to
                        aggregate.")

args = parser.parse_args()
args1 = vars(args)

if __name__ == '__main__':
    # uncomment if you want to create the example database
    try:
```

```
        create_example()
    except Exception:
        pass

    # basic check for empty values in the argument --projection for mode P
    if args.mode == "P":
        if len(args.projection) == 0:
            raise ValueError("No attributes set for projection. Type -h for help.")
        database_str: str = args.database
        table_str: str = args.table
        projection_list = args.projection

        #delete prior results in the dataset
        con = sqlite3.Connection(database=database_str)
        try:
            con.execute(f"DROP TABLE gen_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE P_gen_id_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE P_QofD")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE P_genQofD")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE id{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE prep_{table_str}")
        except Exception:
            pass
        con.commit()
        con.close()

        #execute projection
        projection(database_str, table_str, projection_list)

    # basic check for empty or too many values in the argument --aggregate_function for
        mode P
    # check for empty --aggregate_attribute if --aggregate_function = COUNT -> --
        aggregate_attribute = "*"
    # check for --aggregate_attribute if --aggregate_function != COUNT
    if args.mode == "A":

        database_str: str = args.database
        table_str: str = args.table
        projection_list = args.projection

        # delete prior results in the dataset
        con = sqlite3.Connection(database=database_str)
        try:
            con.execute(f"DROP TABLE gen_{table_str}")
        except Exception:
            pass
```

```
    try:
        con.execute(f"DROP TABLE A_gen_id_{table_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE A_QofD")
        con.execute(f"DROP TABLE A_QofD_Prov")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE A_genQofD")
        con.execute(f"DROP TABLE A_genQofD_Prov")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE id{table_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE prep_{table_str}")
    except Exception:
        pass
    con.commit()
    con.close()

    if len(args.aggregate_function) == 0:
        raise ValueError("No function set for aggregate_function. Type -h for help.")

    if args.aggregate_function == "COUNT":
        if args.aggregate_attribute != None:
            raise ValueError("No aggregate_attribute allowed for COUNT-function. Type
                -h for help.")

        aggregation(args.database, args.table, "COUNT", "*")

    if args.aggregate_function != "COUNT":
        if len(args.aggregate_attribute) == 0:
            raise ValueError("No aggregate_attribute set for aggregate_function. Type
                -h for help.")

        aggregation(args.database, args.table, args.aggregate_function, args.
            aggregate_attribute)

# basic check for empty values in the argument --projection
# basic check for emtpy or too many values in --selection_attribute, --
    selection_operator and --selection_value
# for mode SP
if args.mode == "SP":
    if len(args.selection_attribute) == 0:
        raise ValueError("No value set for argument selection_attribute. Type -h for
            help.")

    if len(args.selection_operator) == 0:
        raise ValueError("No value set for argument selection_operator. Type -h for
            help.")

    if len(args.selection_value) == 0:
        raise ValueError("No value set for argument selection_value. Type -h for help
            .")

    if len(args.projection) == 0:
```

```
            raise ValueError("No attributes set for projection. Type -h for help.")

        database_str: str = args.database
        table_str: str = args.table
        projection_list = args.projection

        # delete prior results in the dataset
        con = sqlite3.Connection(database=database_str)
        try:
            con.execute(f"DROP TABLE gen_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE SP_gen_id_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE SP_QofD")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE SP_genQofD")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE id{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE prep_{table_str}")
        except Exception:
            pass
        con.commit()
        con.close()

        sp(args.database, args.table, args.selection_attribute, args.selection_operator,
            args.selection_value, args.projection)


    # basic checks of previous arguments from previous operations combined for mode SA
    if args.mode == "SA":

        database_str: str = args.database
        table_str: str = args.table
        projection_list = args.projection

        # delete prior results in the dataset
        con = sqlite3.Connection(database=database_str)
        try:
            con.execute(f"DROP TABLE gen_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE SA_gen_id_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE SA_QofD")
            con.execute(f"DROP TABLE SA_QofD_Prov")
        except Exception:
            pass
```

```
    try:
        con.execute(f"DROP TABLE SA_genQofD")
        con.execute(f"DROP TABLE SA_genQofD_Prov")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE id{table_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE prep_{table_str}")
    except Exception:
        pass
    con.commit()
    con.close()

    if len(args.selection_attribute) == 0:
        raise ValueError("No value set for argument selection_attribute. Type -h for
            help.")

    if len(args.selection_operator) == 0:
        raise ValueError("No value set for argument selection_operator. Type -h for
            help.")

    if len(args.selection_value) == 0:
        raise ValueError("No value set for argument selection_value. Type -h for help
            .")

    if len(args.aggregate_function) == 0:
        raise ValueError("No function set for aggregate_function. Type -h for help.")

    if args.aggregate_function == "COUNT":
        if args.aggregate_attribute != None:
            raise ValueError("No aggregate_attribute allowed for COUNT-function. Type
                -h for help.")

        sa(args.database, args.table, args.selection_attribute, args.
            selection_operator, args.selection_value, "COUNT", "*")

    if args.aggregate_function != "COUNT":
        if args.aggregate_attribute == None:
            raise ValueError("No aggregate_attribute set for aggregate_function. Type
                -h for help.")
        sa(args.database, args.table, args.selection_attribute, args.
            selection_operator, args.selection_value, args.aggregate_function, args.
            aggregate_attribute)


if args.mode == "JP":
    if len(args.projection) == 0:
        raise ValueError("No attributes set for projection. Type -h for help.")

    if len(args.natural_join) == 0:
        raise ValueError("No second relation set for argument natural_join. Type -h
            for help.")

    if args.table == args.natural_join:
        raise ValueError("Value for argument table must be different from the one in
            natural_join. Type -h for help.")

    database_str: str = args.database
```

```
        table_str: str = args.table
        projection_list = args.projection
        join_str: str = args.natural_join

        # delete prior results in the dataset
        con = sqlite3.Connection(database=database_str)
        try:
            con.execute(f"DROP TABLE gen_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE gen_{join_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE JP_gen_id_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE JP_gen_id_{join_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE JP_QofD")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE JP_genQofD")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE id{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE id{join_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE prep_{table_str}")
        except Exception:
            pass
        try:
            con.execute(f"DROP TABLE prep_{join_str}")
        except Exception:
            pass
        con.commit()
        con.close()

        jp(args.database, args.table, args.projection, args.natural_join)

    # basic checks like in jp plus s
    if args.mode == "SJP":
        if len(args.projection) == 0:
            raise ValueError("No attributes set for projection. Type -h for help.")

        if len(args.natural_join) == 0:
            raise ValueError("No second relation set for argument natural_join. Type -h
                for help.")

        if len(args.selection_attribute) == 0:
```

```
        raise ValueError("No value set for argument selection_attribute. Type -h for
            help.")

    if len(args.selection_operator) == 0:
        raise ValueError("No value set for argument selection_operator. Type -h for
            help.")

    if len(args.selection_value) == 0:
        raise ValueError("No value set for argument selection_value. Type -h for help
            .")

    database_str: str = args.database
    table_str: str = args.table
    projection_list = args.projection
    join_str: str = args.natural_join

    # delete prior results in the dataset
    con = sqlite3.Connection(database=database_str)
    try:
        con.execute(f"DROP TABLE gen_{table_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE gen_{join_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE SJP_gen_id_{table_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE SJP_gen_id_{join_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE SJP_QofD")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE SJP_genQofD")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE id{table_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE id{join_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE prep_{table_str}")
    except Exception:
        pass
    try:
        con.execute(f"DROP TABLE prep_{join_str}")
    except Exception:
        pass
    con.commit()
    con.close()
```

```
        sjp(args.database, args.table, args.projection, args.natural_join,
            args.selection_attribute, args.selection_operator, args.selection_value)

    print("End of program.")
```

A.2. example.py

```
import sqlite3

def create_example():
    print("Creating example dataset.")
    newpath = (r'C:\Users\Maex\PycharmProjects\pythonProject\EXAMPLE.db')
    NEWDATABASE = sqlite3.connect(database=newpath)
    NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS STUDIERENDE (student_id INTEGER
        PRIMARY KEY, nachname TEXT,"
                        f" vorname TEXT, studiengang TEXT)")
    NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS NOTEN (kurs_nr TEXT, student_id
        INTEGER,"
                            f" semester TEXT, note REAL)")
    NEWDATABASE.commit()

    insert_studierende = (f"INSERT INTO STUDIERENDE (student_id, nachname, vorname,
        studiengang) VALUES (?, ?, ?, ?)")
    s1 = (1, "Muster", "Max", "Informatik")
    s2 = (2, "Neuhaus", "Nancy", "Informatik")
    s3 = (3, "Altmann", "Anne", "Mathematik")
    s4 = (4, "Fern", "Friedrich", "Informatik")
    s5 = (5, "Unruh", "Uwe", "Mathematik")
    NEWDATABASE.execute(insert_studierende, s1)
    NEWDATABASE.execute(insert_studierende, s2)
    NEWDATABASE.execute(insert_studierende, s3)
    NEWDATABASE.execute(insert_studierende, s4)
    NEWDATABASE.execute(insert_studierende, s5)
    NEWDATABASE.commit()
    stringdb1 = NEWDATABASE.execute(f"SELECT * FROM STUDIERENDE").fetchall()
    print(stringdb1)

    insert_noten = ("INSERT INTO NOTEN (kurs_nr, student_id, semester, note) VALUES (?,
        ?, ?, ?)")
    n1 = ("001", 1, "SS 16", 1.7)
    n2 = ("001", 2, "SS 16", 1.3)
    n3 = ("001", 3, "SS 16", 2.3)
    n4 = ("001", 4, "SS 16", 3.3)
    n5 = ("002", 1, r"WS 15/16", 3.0)
    n6 = ("002", 3, r"WS 15/16", 1.0)
    n7 = ("002", 4, r"WS 15/16", 2.7)
    n8 = ("002", 5, r"WS 15/16", 1.7)
    NEWDATABASE.execute(insert_noten, n1)
    NEWDATABASE.execute(insert_noten, n2)
    NEWDATABASE.execute(insert_noten, n3)
    NEWDATABASE.execute(insert_noten, n4)
    NEWDATABASE.execute(insert_noten, n5)
    NEWDATABASE.execute(insert_noten, n6)
    NEWDATABASE.execute(insert_noten, n7)
    NEWDATABASE.execute(insert_noten, n8)
    NEWDATABASE.commit()
    stringdb2 = NEWDATABASE.execute(f"SELECT * FROM NOTEN").fetchall()
    print(stringdb2)
```

```
NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS student_id (student_id TEXT)")
NEWDATABASE.execute(f"INSERT INTO student_id (student_id) VALUES ('x')")
NEWDATABASE.commit()
stringdb3 = NEWDATABASE.execute(f"SELECT * FROM student_id").fetchall()
print(stringdb3)

NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS vorname (vorname TEXT)")
NEWDATABASE.execute(f"INSERT INTO vorname (vorname) VALUES ('x')")
NEWDATABASE.commit()
stringdb4 = NEWDATABASE.execute(f"SELECT * FROM vorname").fetchall()
print(stringdb4)

NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS nachname (nachname TEXT)")
NEWDATABASE.execute(f"INSERT INTO nachname (nachname) VALUES ('x')")
NEWDATABASE.commit()
stringdb5 = NEWDATABASE.execute(f"SELECT * FROM nachname").fetchall()
print(stringdb5)

NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS studiengang (studiengang TEXT,
    studiengangGen TEXT)")
insert_studiengang = (f"INSERT INTO studiengang (studiengang, studiengangGen) VALUES
    (?,?)")
sg1 = ("Informatik", "Science")
sg2 = ("Mathematik", "Science")
NEWDATABASE.execute(insert_studiengang, sg1)
NEWDATABASE.execute(insert_studiengang, sg2)
NEWDATABASE.commit()
stringdb6 = NEWDATABASE.execute(f"SELECT * FROM studiengang").fetchall()
print(stringdb6)

NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS kurs_nr (kurs_nr TEXT)")
NEWDATABASE.execute(f"INSERT INTO kurs_nr (kurs_nr) VALUES ('x')")
NEWDATABASE.commit()
stringdb7 = NEWDATABASE.execute(f"SELECT * FROM kurs_nr").fetchall()
print(stringdb7)

NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS semester (semester TEXT, semesterGen
    TEXT)")
insert_semester = (f"INSERT INTO semester (semester, semesterGen) VALUES (?,?)")
ss1 = ("SS 16", "SS")
ss2 = (r"WS 15/16", "WS")
NEWDATABASE.execute(insert_semester, ss1)
NEWDATABASE.execute(insert_semester, ss2)
NEWDATABASE.commit()
stringdb8 = NEWDATABASE.execute(f"SELECT * FROM semester").fetchall()
print(stringdb8)

NEWDATABASE.execute(f"CREATE TABLE IF NOT EXISTS note (note REAL, noteUG REAL, noteOG
    REAL)")
insert_note = (f"INSERT INTO note (note, noteUG, noteOG) VALUES (?,?,?)")
nn1 = (1.0, 1.0, 1.5)
nn2 = (1.3, 1.0, 1.5)
nn3 = (1.7, 1.5, 2.5)
nn4 = (2.0, 1.5, 2.5)
nn5 = (2.3, 1.5, 2.5)
nn6 = (2.7, 2.5, 3.5)
nn7 = (3.0, 2.5, 3.5)
nn8 = (3.3, 2.5, 3.5)
nn9 = (3.7, 3.5, 4.0)
nn10 = (4.0, 4.0, 5.0)
```

```
    nn11 = (4.3, 4.0, 5.0)
    nn12 = (4.7, 4.0, 5.0)
    nn13 = (5.0, 4.0, 5.0)
    NEWDATABASE.execute(insert_note, nn1)
    NEWDATABASE.execute(insert_note, nn2)
    NEWDATABASE.execute(insert_note, nn3)
    NEWDATABASE.execute(insert_note, nn4)
    NEWDATABASE.execute(insert_note, nn5)
    NEWDATABASE.execute(insert_note, nn6)
    NEWDATABASE.execute(insert_note, nn7)
    NEWDATABASE.execute(insert_note, nn8)
    NEWDATABASE.execute(insert_note, nn9)
    NEWDATABASE.execute(insert_note, nn10)
    NEWDATABASE.execute(insert_note, nn11)
    NEWDATABASE.execute(insert_note, nn12)
    NEWDATABASE.execute(insert_note, nn13)
    NEWDATABASE.commit()
    stringdb9 = NEWDATABASE.execute(f"SELECT * FROM note").fetchall()
    print(stringdb9)
    print("Example dataset created")
    print()
    print()
```

A.3. projection.py

```
import sqlite3
from pathlib import Path

def projection(database: str, table: str, projection: list):
    # convert path string to path
    db_file = Path(database)
    # create database connection
    DATABASE = sqlite3.connect(database=db_file)
    # define table_id
    table_id: str = table + 'ID'
    # concatenate projection attribute-list to string
    projection_string_wo_id: str = ','.join(projection)
    # add table_id to projection string
    projection_string: str = projection_string_wo_id + ',' + table_id

    #copy the table, add a random id, permutate via the new id
    id_table: str = 'id' + table
    DATABASE.execute(f"CREATE TABLE {id_table} AS SELECT *, random() AS {table_id} FROM {
        table} ORDER BY {table_id}")

    # calculating Q(D) with ID
    DATABASE.execute(f"CREATE TABLE P_QofD AS SELECT {projection_string} FROM {id_table
        }")

    # create list of column names in table
    cursor = DATABASE.execute(f"SELECT * FROM {table} LIMIT 1")
    columnNames: list = [description[0] for description in cursor.description]

    # create new generalisation entry table
    DATABASE.execute(f"CREATE TABLE prep_{table} AS SELECT * FROM {table}")
    DATABASE.commit()
    # generate projection string for generalised attributes to create generalized dataset
    # add generalized attributes to different projection list for Q' if they appear in Q
    projection_list_generalisation: list = []
```

```
projection_generalized_wo_id: list = []
columns_to_generalise: list = []
for i in columnNames:
    try:
        cursor2 = DATABASE.execute(f"SELECT * FROM {i} LIMIT 1")
        tuple = cursor2.fetchone()
        colCount: int = len(tuple)
    except Exception:
        projection_list_generalisation.append(f"{i}")
        if i in projection:
            projection_generalized_wo_id.append(f"{i}")
    else:
        if colCount == 1:
            token: str = tuple[0]
            projection_list_generalisation.append(f"{i}")
            columns_to_generalise.append(f"{i}")
            DATABASE.execute(f"ALTER TABLE prep_{table} RENAME COLUMN {i} TO {i}old")
            DATABASE.execute(f"ALTER TABLE prep_{table} ADD COLUMN {i}")
            DATABASE.execute(f"UPDATE prep_{table} SET {i} = '{token}'")
            if i in projection:
                projection_generalized_wo_id.append(f"{i}")
        if colCount == 2:
            projection_list_generalisation.append(f"{i}Gen")
            columns_to_generalise.append(f"{i}")
            if i in projection:
                projection_generalized_wo_id.append(f"{i}Gen")
        if colCount == 3:
            columns_to_generalise.append(f"{i}")
            projection_list_generalisation.append(f"{i}UG")
            projection_list_generalisation.append(f"{i}OG")
            if i in projection:
                projection_generalized_wo_id.append(f"{i}UG")
                projection_generalized_wo_id.append(f"{i}OG")
projection_string_generalisation: str = ','.join(projection_list_generalisation)

# generate join string for generalisation
generalisingJoin: str = ' NATURAL JOIN '.join(columns_to_generalise)

# generalisation of D - creation of D' without IDs
DATABASE.execute(f"CREATE TABLE gen_{table} AS SELECT {
    projection_string_generalisation} FROM prep_{table} "
                 f"NATURAL JOIN {generalisingJoin}")

# id generation and permutation for D'
DATABASE.execute(f"CREATE TABLE P_gen_id_{table} AS SELECT *, random() AS {table_id}"
                 f" FROM gen_{table} ORDER BY {table_id}")

# add ID to generalized projection list, form list to string, execution of Q'
projection_generalized_wo_id.append(f"{table_id}")
projection_generalized: str = ','.join(projection_generalized_wo_id)
DATABASE.execute(f"CREATE TABLE P_genQofD AS SELECT {projection_generalized} FROM
    P_gen_id_{table}")
DATABASE.commit()
DATABASE.close()

DATABASE = sqlite3.connect(database=db_file)
# deletion of prep_table
# deletion of gen_table without ID
DATABASE.execute(f"DROP TABLE {id_table}")
DATABASE.execute(f"DROP TABLE prep_{table}")
DATABASE.execute(f"DROP TABLE gen_{table}")
```

```
    DATABASE.commit()

    QofD = DATABASE.execute("SELECT * FROM P_QofD").fetchall()
    genQofD = DATABASE.execute("SELECT * FROM P_genQofD").fetchall()
    genD = DATABASE.execute(f"SELECT * FROM P_gen_id_{table}").fetchall()
    print(f"Regular result P_QofD:")
    print(QofD)
    print(f"Generalized result P_genQofD:")
    print(genQofD)
    print(f"Intensional provenance-answer P_gen_id_{table}:")
    print(genD)
    print("Generalized query Q':")
    print(f"CREATE TABLE P_genQofD AS SELECT {projection_generalized} FROM P_gen_id_{
        table}")
    DATABASE.close()

    print("Projection finished.")
```

A.4. aggregation.py

```
import sqlite3
from pathlib import Path

def aggregation(database: str, table: str, function: str, attribute: str):
    # convert path string to path
    db_file = Path(database)
    # create database connection
    DATABASE = sqlite3.connect(database=db_file)
    # define table_id
    table_id: str = table + 'ID'
    #agg_string
    agg_function = f"{function}({attribute})"
    agg_string = f"'{function}({attribute})'"

    #agg_string_generalized
    if function == 'COUNT':
        agg_string_generalized = f'"{function}({attribute})"'
        agg_function_generalized = f"{function}({attribute})"
    else:
        agg_ug_function = f"{function}({attribute}UG)"
        agg_og_function = f"{function}({attribute}OG)"
        agg_ug_string = f'"{function}({attribute}UG)"'
        agg_og_string = f'"{function}({attribute}OG)"'
        agg_function_generalized = f"{agg_ug_function},{agg_og_function}"
        agg_string_generalized = f'{agg_ug_string},{agg_og_string}'

    # copy the table, add a random id, permutate via the new id
    id_table: str = 'id' + table
    DATABASE.execute(f"CREATE TABLE {id_table} AS SELECT *, random() AS {table_id} FROM {
        table} ORDER BY {table_id}")

    # calculating Q(D) with ID
    DATABASE.execute(f"CREATE TABLE A_QofD AS SELECT {agg_function} FROM {id_table}")
    DATABASE.commit()

    #save agg value to string
    agg_value = DATABASE.execute(f"SELECT * FROM A_QofD").fetchone()[0]

    #create provenance table
```

```
DATABASE.execute(f"CREATE TABLE A_QofD_Prov AS SELECT {agg_value} AS {agg_string}, {
    table_id} FROM {id_table}")

# create list of column names in table
cursor = DATABASE.execute(f"SELECT * FROM {table} LIMIT 1")
columnNames: list = [description[0] for description in cursor.description]

# create new generalisation entry table
DATABASE.execute(f"CREATE TABLE prep_{table} AS SELECT * FROM {table}")
DATABASE.commit()

# generate projection string for generalised attributes to create generalized dataset
projection_list_generalisation: list = []
columns_to_generalise: list = []
agg_ug: str
agg_og: str
for i in columnNames:
    try:
        cursor2 = DATABASE.execute(f"SELECT * FROM {i} LIMIT 1")
        tuple = cursor2.fetchone()
        colCount: int = len(tuple)
    except Exception:
        projection_list_generalisation.append(f"{i}")
    else:
        if colCount == 1:
            token: str = tuple[0]
            projection_list_generalisation.append(f"{i}")
            DATABASE.execute(f"ALTER TABLE prep_{table} RENAME COLUMN {i} TO {i}old")
            DATABASE.execute(f"ALTER TABLE prep_{table} ADD COLUMN {i}")
            DATABASE.execute(f"UPDATE prep_{table} SET {i} = '{token}'")
        if colCount == 2:
            projection_list_generalisation.append(f"{i}Gen")
            columns_to_generalise.append(f"{i}")
        if colCount == 3:
            columns_to_generalise.append(f"{i}")
            projection_list_generalisation.append(f"{i}UG")
            projection_list_generalisation.append(f"{i}OG")
#names of all attributes after generalisation separated by ,
projection_string_generalisation: str = ','.join(projection_list_generalisation)

# generate join string for generalisation
generalisingJoin: str = ' NATURAL JOIN '.join(columns_to_generalise)

# generalisation of D - creation of D' without IDs
DATABASE.execute(f"CREATE TABLE gen_{table} AS SELECT {
    projection_string_generalisation} FROM prep_{table} "
                 f"NATURAL JOIN {generalisingJoin}")
DATABASE.commit()

# id generation and permutation for D'
DATABASE.execute(f"CREATE TABLE A_gen_id_{table} AS SELECT *, random() AS {table_id}"
                 f" FROM gen_{table} ORDER BY {table_id}")
DATABASE.commit()

# create Q'(D')
DATABASE.execute(f"CREATE TABLE A_genQofD AS SELECT {agg_function_generalized} FROM
    A_gen_id_{table}")
DATABASE.commit()

# create provenance table for Q'(D')
if function == 'COUNT':
```

```
        agg_value_generalized = DATABASE.execute(f"SELECT {agg_string_generalized} FROM
            A_genQofD").fetchone()[0]
        DATABASE.execute(f"CREATE TABLE A_genQofD_Prov AS SELECT {agg_value_generalized}
            AS {agg_string}, "
                         f"{table_id} FROM A_gen_id_{table}")
    else:
        agg_value_generalized = DATABASE.execute(f"SELECT {agg_ug_string},{agg_og_string}
             FROM A_genQofD").fetchone()
        agg_value_generalized_ug = agg_value_generalized[0]
        agg_value_generalized_og = agg_value_generalized[1]
        DATABASE.execute(f"CREATE TABLE A_genQofD_Prov AS SELECT '{
            agg_value_generalized_ug}' AS {agg_ug_string}, "
                         f"'{agg_value_generalized_og}' AS {agg_og_string}, {table_id}
                             FROM A_gen_id_{table}")
    DATABASE.close()

    DATABASE = sqlite3.connect(database=db_file)
    # deletion of prep_table
    # deletion of gen_table without ID
    DATABASE.execute(f"DROP TABLE {id_table}")
    DATABASE.execute(f"DROP TABLE prep_{table}")
    DATABASE.execute(f"DROP TABLE gen_{table}")
    DATABASE.commit()

    QofD = DATABASE.execute("SELECT * FROM A_QofD").fetchall()
    QofDprov = DATABASE.execute("SELECT * FROM A_QofD_Prov").fetchall()
    genQofD = DATABASE.execute("SELECT * FROM A_genQofD").fetchall()
    genQofDprov = DATABASE.execute("SELECT * FROM A_genQofD_Prov").fetchall()
    genD = DATABASE.execute(f"SELECT * FROM A_gen_id_{table}").fetchall()
    print(f"Regular result A_QofD:")
    print(QofD)
    print(f"Provenance of A_QofD:")
    print(QofDprov)
    print(f"Generalized result A_genQofD:")
    print(genQofD)
    print(f"Provenance of A_genQofD:")
    print(genQofDprov)
    print(f"Intensional provenance-answer P_gen_id_{table}:")
    print(genD)
    print("Generalized query Q':")
    print(f"CREATE TABLE A_genQofD AS SELECT {agg_string_generalized} FROM A_gen_id_{
        table}")
    DATABASE.close()

    print("Aggregation finished.")
```

A.5. sp.py

```
import sqlite3
from pathlib import Path

def sp(database: str, table: str, attribute: str, operator: str, value: str, projection:
    str):

    # convert path string to path
    db_file = Path(database)
    # create database connection
    DATABASE = sqlite3.connect(database=db_file)
    # define table_id
```

```
table_id: str = table + 'ID'

value_string: str = fr'"{value}"'

#generalize selection_operator
if operator == "<":
    operator_gen = '<='
elif operator == ">":
    operator_gen = ">="
else:
    operator_gen = operator

#selection string
sel_string = attribute + operator + value_string

# concatenate projection attribute-list to string
projection_string_wo_id: str = ','.join(projection)
# add table_id to projection string
projection_string: str = projection_string_wo_id + ',' + table_id

# copy the table, add a random id, permutate via the new id
id_table: str = 'id' + table
DATABASE.execute(f"CREATE TABLE {id_table} AS SELECT *, random() AS {table_id} FROM {
    table} ORDER BY {table_id}")

# calculating Q(D) with ID
DATABASE.execute(f"CREATE TABLE SP_QofD AS SELECT {projection_string} FROM {id_table}
     WHERE {sel_string}")

# create list of column names in table
cursor = DATABASE.execute(f"SELECT * FROM {table} LIMIT 1")
columnNames: list = [description[0] for description in cursor.description]

# create new table with relevant entries for generalisation
DATABASE.execute(f"CREATE TABLE prep_{table} AS SELECT * FROM {table} WHERE {
    sel_string}")
DATABASE.commit()
# generate projection string for generalised attributes to create generalized dataset
# add generalized attributes to different projection list for Q' if they appear in Q
projection_list_generalisation: list = []
projection_generalized_wo_id: list = []
columns_to_generalise: list = []
attribute_gen: str
value_gen: str

for i in columnNames:
    try:
        cursor2 = DATABASE.execute(f"SELECT * FROM {i} LIMIT 1")
        tuple = cursor2.fetchone()
        colCount: int = len(tuple)
    except Exception:
        projection_list_generalisation.append(f"{i}")
        if i in projection:
            projection_generalized_wo_id.append(f"{i}")
        if i == attribute:
            attribute_gen = attribute
            value_gen = value_string
    else:
        if colCount == 1:
            token: str = tuple[0]
            projection_list_generalisation.append(f"{i}")
```

```
                columns_to_generalise.append(f"{i}")
                DATABASE.execute(f"ALTER TABLE prep_{table} RENAME COLUMN {i} TO {i}old")
                DATABASE.execute(f"ALTER TABLE prep_{table} ADD COLUMN {i}")
                DATABASE.execute(f"UPDATE prep_{table} SET {i} = '{token}'")
                if i in projection:
                    projection_generalized_wo_id.append(f"{i}")
                if i == attribute:
                    attribute_gen = attribute
                    value_gen = token
            if colCount == 2:
                projection_list_generalisation.append(f"{i}Gen")
                columns_to_generalise.append(f"{i}")
                if i in projection:
                    projection_generalized_wo_id.append(f"{i}Gen")
                if i == attribute:
                    attribute_gen = attribute + 'Gen'
                    value_gen = DATABASE.execute(f"SELECT {attribute_gen} FROM {i} "
                                                 f"WHERE {sel_string}").fetchone()[0]
            if colCount == 3:
                columns_to_generalise.append(f"{i}")
                projection_list_generalisation.append(f"{i}UG")
                projection_list_generalisation.append(f"{i}OG")
                if i in projection:
                    projection_generalized_wo_id.append(f"{i}UG")
                    projection_generalized_wo_id.append(f"{i}OG")
                if i == attribute:
                    attribute_gen = attribute + 'UG'
                    value_gen = DATABASE.execute(f"SELECT {attribute_gen} FROM {i} "
                                                 f"WHERE {attribute} = {value}").fetchone
                                                     ()[0]
    value_gen_string = rf'"{value_gen}"'
    sel_string_generalized = attribute_gen + operator_gen + value_gen_string
    projection_string_generalisation: str = ','.join(projection_list_generalisation)

    # generate join string for generalisation
    generalisingJoin: str = ' NATURAL JOIN '.join(columns_to_generalise)

    # generalisation of boldD - creation of boldD' without IDs
    DATABASE.execute(f"CREATE TABLE gen_{table} AS SELECT {
        projection_string_generalisation} FROM prep_{table} "
                     f"NATURAL JOIN {generalisingJoin}")

    # id generation and permutation for boldD'
    DATABASE.execute(f"CREATE TABLE SP_gen_id_{table} AS SELECT *, random() AS {table_id
        }"
                     f" FROM gen_{table} ORDER BY {table_id}")

    # add ID to generalized projection list, form list to string, execution of Q'
    projection_generalized_wo_id.append(f"{table_id}")
    projection_generalized: str = ','.join(projection_generalized_wo_id)
    DATABASE.execute(f"CREATE TABLE SP_genQofD AS SELECT {projection_generalized} FROM
        SP_gen_id_{table} "
                     f"WHERE {sel_string_generalized}")
    DATABASE.commit()
    DATABASE.close()

    DATABASE = sqlite3.connect(database=db_file)
    # deletion of prep_table
    # deletion of gen_table without ID
    DATABASE.execute(f"DROP TABLE {id_table}")
    DATABASE.execute(f"DROP TABLE prep_{table}")
```

```
    DATABASE.execute(f"DROP TABLE gen_{table}")
    DATABASE.commit()

    QofD = DATABASE.execute("SELECT * FROM SP_QofD").fetchall()
    genQofD = DATABASE.execute("SELECT * FROM SP_genQofD").fetchall()
    genD = DATABASE.execute(f"SELECT * FROM SP_gen_id_{table}").fetchall()
    print(f"Regular result SP_QofD:")
    print(QofD)
    print(f"Generalized result SP_genQofD:")
    print(genQofD)
    print(f"Intensional provenance-answer SP_gen_id_{table}:")
    print(genD)
    print("Generalized query Q':")
    print(f"CREATE TABLE SP_genQofD AS SELECT {projection_generalized} FROM SP_gen_id_{
        table} "
                    f"WHERE {sel_string_generalized}")
    DATABASE.close()

    print("SP-query finished.")
```

A.6. sa.py

```
import sqlite3
from pathlib import Path

def sa(database: str, table: str, sattribute:str, operator:str, value:str,
       function: str, attribute: str):
    # convert path string to path
    db_file = Path(database)
    # create database connection
    DATABASE = sqlite3.connect(database=db_file)
    # define table_id
    table_id: str = table + 'ID'
    #agg_string
    agg_function = f"{function}({attribute})"
    agg_string = f"'{function}({attribute})'"

    value_string: str = fr'"{value}"'
    operator = operator

    # generalize selection_operator
    if operator == "<":
        operator_gen = '<='
    elif operator == ">":
        operator_gen = ">="
    else:
        operator_gen = operator

    # selection string
    sel_string = sattribute + operator + value_string

    #agg_string_generalized
    if function == 'COUNT':
        agg_string_generalized = f'"{function}({attribute})"'
        agg_function_generalized = f"{function}({attribute})"
    else:
        agg_ug_function = f"{function}({attribute}UG)"
        agg_og_function = f"{function}({attribute}OG)"
        agg_ug_string = f'"{function}({attribute}UG)"'
```

```
        agg_og_string = f'"{function}({attribute}OG)"'
        agg_function_generalized = f"{agg_ug_function},{agg_og_function}"
        agg_string_generalized = f'{agg_ug_string},{agg_og_string}'

    # copy the table, add a random id, permutate via the new id
    id_table: str = 'id' + table
    DATABASE.execute(f"CREATE TABLE {id_table} AS SELECT *, random() AS {table_id} FROM {
        table} ORDER BY {table_id}")

    # calculating Q(D) with ID
    DATABASE.execute(f"CREATE TABLE SA_QofD AS SELECT {agg_function} FROM {id_table}
        WHERE {sel_string}")
    DATABASE.commit()

    #save agg value to string
    agg_value = DATABASE.execute(f"SELECT * FROM SA_QofD").fetchone()[0]

    #create provenance table
    DATABASE.execute(f"CREATE TABLE SA_QofD_Prov AS SELECT {agg_value} AS {agg_string}, {
        table_id} FROM {id_table} "
                     f"WHERE {sel_string}")

    # create list of column names in table
    cursor = DATABASE.execute(f"SELECT * FROM {table} LIMIT 1")
    columnNames: list = [description[0] for description in cursor.description]

    # create new table only with relevant entries for generalization
    DATABASE.execute(f"CREATE TABLE prep_{table} AS SELECT * FROM {table} WHERE {
        sel_string}")
    DATABASE.commit()

    # generate projection string for generalised attributes to create generalized dataset
    projection_list_generalisation: list = []
    columns_to_generalise: list = []
    agg_ug: str
    agg_og: str
    sattribute_gen: str
    value_gen: str
    for i in columnNames:
        try:
            cursor2 = DATABASE.execute(f"SELECT * FROM {i} LIMIT 1")
            tuple = cursor2.fetchone()
            colCount: int = len(tuple)
        except Exception:
            projection_list_generalisation.append(f"{i}")
            if i == sattribute:
                sattribute_gen = sattribute
                value_gen = value_string
        else:
            if colCount == 1:
                token: str = tuple[0]
                projection_list_generalisation.append(f"{i}")
                DATABASE.execute(f"ALTER TABLE prep_{table} RENAME COLUMN {i} TO {i}old")
                DATABASE.execute(f"ALTER TABLE prep_{table} ADD COLUMN {i}")
                DATABASE.execute(f"UPDATE prep_{table} SET {i} = '{token}'")
                if i == sattribute:
                    sattribute_gen = sattribute
                    value_gen = token
            if colCount == 2:
                projection_list_generalisation.append(f"{i}Gen")
                columns_to_generalise.append(f"{i}")
```

```
            if i == sattribute:
                sattribute_gen = sattribute + 'Gen'
                value_gen = DATABASE.execute(f"SELECT {sattribute_gen} FROM {i} "
                                             f"WHERE {sel_string}").fetchone()[0]
        if colCount == 3:
            columns_to_generalise.append(f"{i}")
            projection_list_generalisation.append(f"{i}UG")
            projection_list_generalisation.append(f"{i}OG")
            if i == sattribute:
                sattribute_gen = sattribute + 'UG'
                value_gen = DATABASE.execute(f"SELECT {sattribute_gen} FROM {i} "
                                             f"WHERE {sattribute} = {value}").
                                                 fetchone()[0]

value_gen_string = rf'"{value_gen}"'
sel_string_generalized = sattribute_gen + operator_gen + value_gen_string

#names of all attributes after generalisation separated by ,
projection_string_generalisation: str = ','.join(projection_list_generalisation)

# generate join string for generalisation
generalisingJoin: str = ' NATURAL JOIN '.join(columns_to_generalise)

# generalisation of D - creation of D' without IDs
DATABASE.execute(f"CREATE TABLE gen_{table} AS SELECT {
    projection_string_generalisation} FROM prep_{table} "
                 f"NATURAL JOIN {generalisingJoin}")
DATABASE.commit()

# id generation and permutation for D'
DATABASE.execute(f"CREATE TABLE SA_gen_id_{table} AS SELECT *, random() AS {table_id
    }"
                 f" FROM gen_{table} ORDER BY {table_id}")
DATABASE.commit()

# create Q'(D')
DATABASE.execute(f"CREATE TABLE SA_genQofD AS SELECT {agg_function_generalized} FROM
    SA_gen_id_{table} "
                 f"WHERE {sel_string_generalized}")
DATABASE.commit()

# create provenance table for Q'(D')
if function == 'COUNT':
    agg_value_generalized = DATABASE.execute(f"SELECT {agg_string_generalized} FROM
        SA_genQofD").fetchone()[0]
    DATABASE.execute(f"CREATE TABLE SA_genQofD_Prov AS SELECT {agg_value_generalized}
         AS {agg_string}, "
                     f"{table_id} FROM SA_gen_id_{table} WHERE {
                         sel_string_generalized}")
else:
    agg_value_generalized = DATABASE.execute(f"SELECT {agg_ug_string},{agg_og_string}
         FROM SA_genQofD").fetchone()
    agg_value_generalized_ug = agg_value_generalized[0]
    agg_value_generalized_og = agg_value_generalized[1]
    DATABASE.execute(f"CREATE TABLE SA_genQofD_Prov AS SELECT '{
        agg_value_generalized_ug}' AS {agg_ug_string}, "
                     f"'{agg_value_generalized_og}' AS {agg_og_string}, {table_id}
                         FROM SA_gen_id_{table} "
                     f"WHERE {sel_string_generalized}")
DATABASE.close()
```

```
    DATABASE = sqlite3.connect(database=db_file)
    # deletion of prep_table
    # deletion of gen_table without ID
    DATABASE.execute(f"DROP TABLE {id_table}")
    DATABASE.execute(f"DROP TABLE prep_{table}")
    DATABASE.execute(f"DROP TABLE gen_{table}")
    DATABASE.commit()

    QofD = DATABASE.execute("SELECT * FROM SA_QofD").fetchall()
    QofDprov = DATABASE.execute("SELECT * FROM SA_QofD_Prov").fetchall()
    genQofD = DATABASE.execute("SELECT * FROM SA_genQofD").fetchall()
    genQofDprov = DATABASE.execute("SELECT * FROM SA_genQofD_Prov").fetchall()
    genD = DATABASE.execute(f"SELECT * FROM SA_gen_id_{table}").fetchall()
    print(f"Regular result SA_QofD:")
    print(QofD)
    print(f"Provenance of SA_QofD:")
    print(QofDprov)
    print(f"Generalized result SA_genQofD:")
    print(genQofD)
    print(f"Provenance of SA_genQofD:")
    print(genQofDprov)
    print(f"Intensional provenance-answer P_gen_id_{table}:")
    print(genD)
    print("Generalized query Q':")
    print(f"CREATE TABLE SA_genQofD AS SELECT {agg_function_generalized} FROM A_gen_id_{
        table} "
                    f"WHERE {sel_string_generalized}")
    DATABASE.close()

    print("SA-query finished.")
```

A.7. jp.py

```
import sqlite3
from pathlib import Path

def jp(database: str, table: str, projection: list, join:str):
    # convert path string to path
    db_file = Path(database)
    # create database connection
    DATABASE = sqlite3.connect(database=db_file)
    # define table_id
    table_id: str = table + 'ID'
    #define join_id
    join_id: str = join + 'ID'
    # concatenate projection attribute-list to string
    projection_string_wo_id: str = ','.join(projection)
    # add table_id to projection string
    projection_string: str = projection_string_wo_id + ',' + table_id + ',' + join_id

    #copy the table, add a random id, permutate via the new id
    id_table: str = 'id' + table
    DATABASE.execute(f"CREATE TABLE {id_table} AS SELECT *, random() AS {table_id} FROM {
        table} ORDER BY {table_id}")

    #copy the join table, add a random id, permutate via the new id
    id_join: str = 'id' + join
    DATABASE.execute(f"CREATE TABLE {id_join} AS SELECT *, random() AS {join_id} FROM {
        join} ORDER BY {join_id}")
```

```
# calculating Q(D) with ID
DATABASE.execute(f"CREATE TABLE JP_QofD AS SELECT {projection_string} FROM {id_table}
     NATURAL JOIN {id_join}")

# create list of column names in table
cursort = DATABASE.execute(f"SELECT * FROM {table} LIMIT 1")
columnNamest: list = [description[0] for description in cursort.description]
cursorj = DATABASE.execute(f"SELECT * FROM {join} LIMIT 1")
columnNamesj: list = [description[0] for description in cursorj.description]

cursorx = DATABASE.execute(f"SELECT {table_id} FROM JP_QofD")
col = 0
table_ids_list = [x[col] for x in cursorx]
table_ids = tuple(table_ids_list)
cursory = DATABASE.execute(f"SELECT {join_id} FROM JP_QofD")
join_ids_list = [y[col] for y in cursory]
join_ids = tuple(join_ids_list)

# create tables with relevant entries
if len(table_ids) != 1:
    DATABASE.execute(f"CREATE TABLE prep_{table} AS SELECT * FROM {id_table} WHERE {
        table_id} IN {table_ids}")
else:
    DATABASE.execute(f"CREATE TABLE prep_{table} AS SELECT * FROM {id_table} WHERE {
        table_id} = {table_ids[0]}")
if len(join_ids) != 1:
    DATABASE.execute(f"CREATE TABLE prep_{join} AS SELECT * FROM {id_join} WHERE {
        join_id} IN {join_ids}")
else:
    DATABASE.execute(f"CREATE TABLE prep_{join} AS SELECT * FROM {id_join} WHERE {
        join_id} = {join_ids[0]}")
DATABASE.commit()

# generate projection string for generalised attributes to create generalized table
# add generalized attributes to projection list for Q' if they appear in Q
projection_list_generalisation: list = []
projection_generalized_wo_id: list = []
columns_to_generalise: list = []
for i in columnNamest:
    try:
        cursor2 = DATABASE.execute(f"SELECT * FROM {i} LIMIT 1")
        tuple2 = cursor2.fetchone()
        colCount: int = len(tuple2)
    except Exception:
        projection_list_generalisation.append(f"{i}")
        if i in projection:
            projection_generalized_wo_id.append(f"{i}")
    else:
        if colCount == 1:
            token: str = tuple2[0]
            projection_list_generalisation.append(f"{i}")
            columns_to_generalise.append(f"{i}")
            DATABASE.execute(f"ALTER TABLE prep_{table} RENAME COLUMN {i} TO {i}old")
            DATABASE.execute(f"ALTER TABLE prep_{table} ADD COLUMN {i}")
            DATABASE.execute(f"UPDATE prep_{table} SET {i} = '{token}'")
            if i in projection:
                projection_generalized_wo_id.append(f"{i}")
        if colCount == 2:
            projection_list_generalisation.append(f"{i}Gen")
            columns_to_generalise.append(f"{i}")
```

```
            if i in projection:
                projection_generalized_wo_id.append(f"{i}Gen")
        if colCount == 3:
            columns_to_generalise.append(f"{i}")
            projection_list_generalisation.append(f"{i}UG")
            projection_list_generalisation.append(f"{i}OG")
            if i in projection:
                projection_generalized_wo_id.append(f"{i}UG")
                projection_generalized_wo_id.append(f"{i}OG")
projection_string_generalisation: str = ','.join(projection_list_generalisation)

# generate join string for generalisation of table
generalisingJoin: str = ' NATURAL JOIN '.join(columns_to_generalise)

# generalisation of table' - creation of table' wo IDs
DATABASE.execute(f"CREATE TABLE gen_{table} AS SELECT {
    projection_string_generalisation} FROM prep_{table} "
                 f"NATURAL JOIN {generalisingJoin}")

# id generation and permutation for table'
DATABASE.execute(f"CREATE TABLE JP_gen_id_{table} AS SELECT *, random() AS {table_id
    }"
                 f" FROM gen_{table} ORDER BY {table_id}")

# generate projection string for generalised attributes to create generalized join-
    table
# add generalized attributes to projection list for Q' if they appear in Q
projection_list_generalisation2: list = []
projection_generalized_wo_id2: list = []
columns_to_generalise2: list = []
for i in columnNamesj:
    try:
        cursor3 = DATABASE.execute(f"SELECT * FROM {i} LIMIT 1")
        tuple3 = cursor3.fetchone()
        colCount: int = len(tuple3)
    except Exception:
        projection_list_generalisation2.append(f"{i}")
        if i in projection:
            projection_generalized_wo_id2.append(f"{i}")
    else:
        if colCount == 1:
            token: str = tuple3[0]
            projection_list_generalisation2.append(f"{i}")
            columns_to_generalise2.append(f"{i}")
            DATABASE.execute(f"ALTER TABLE prep_{join} RENAME COLUMN {i} TO {i}old")
            DATABASE.execute(f"ALTER TABLE prep_{join} ADD COLUMN {i}")
            DATABASE.execute(f"UPDATE prep_{join} SET {i} = '{token}'")
            if i in projection:
                projection_generalized_wo_id2.append(f"{i}")
        if colCount == 2:
            projection_list_generalisation2.append(f"{i}Gen")
            columns_to_generalise2.append(f"{i}")
            if i in projection:
                projection_generalized_wo_id2.append(f"{i}Gen")
        if colCount == 3:
            columns_to_generalise2.append(f"{i}")
            projection_list_generalisation2.append(f"{i}UG")
            projection_list_generalisation2.append(f"{i}OG")
            if i in projection:
                projection_generalized_wo_id2.append(f"{i}UG")
                projection_generalized_wo_id2.append(f"{i}OG")
```

```
projection_string_generalisation2: str = ','.join(projection_list_generalisation2)

# generate join string for generalisation of join
generalisingJoin2: str = ' NATURAL JOIN '.join(columns_to_generalise2)

# generalisation of join - creation of join' wo IDs
DATABASE.execute(f"CREATE TABLE gen_{join} AS SELECT {
    projection_string_generalisation2} FROM prep_{join} "
                 f"NATURAL JOIN {generalisingJoin2}")

# id generation and permutation for join'
DATABASE.execute(f"CREATE TABLE JP_gen_id_{join} AS SELECT *, random() AS {join_id}"
                 f" FROM gen_{join} ORDER BY {join_id}")




# add ID to generalized projection list, form list to string, execution of Q'
projection_gen_list = projection_generalized_wo_id + projection_generalized_wo_id2
projection_gen_list.append(f"{table_id}")
projection_gen_list.append(f"{join_id}")
projection_generalized: str = ','.join(projection_gen_list)
DATABASE.execute(f"CREATE TABLE JP_genQofD AS SELECT {projection_generalized} "
                 f"FROM JP_gen_id_{table} NATURAL JOIN JP_gen_id_{join}")
DATABASE.commit()
DATABASE.close()

DATABASE = sqlite3.connect(database=db_file)
# deletion of prep_table
# deletion of gen_table without ID
DATABASE.execute(f"DROP TABLE {id_table}")
DATABASE.execute(f"DROP TABLE {id_join}")
DATABASE.execute(f"DROP TABLE prep_{table}")
DATABASE.execute(f"DROP TABLE prep_{join}")
DATABASE.execute(f"DROP TABLE gen_{table}")
DATABASE.execute(f"DROP TABLE gen_{join}")
DATABASE.commit()

QofD = DATABASE.execute("SELECT * FROM JP_QofD").fetchall()
genQofD = DATABASE.execute("SELECT * FROM JP_genQofD").fetchall()
genD = DATABASE.execute(f"SELECT * FROM JP_gen_id_{table}").fetchall()
genD2 = DATABASE.execute(f"SELECT * FROM JP_gen_id_{join}").fetchall()
print(f"Regular result P_QofD:")
print(QofD)
print(f"Generalized result P_genQofD:")
print(genQofD)
print(f"Intensional provenance-answer JP_gen_id_{table} and JP_gen_id_{join}:")
print(genD)
print(genD2)
print("Generalized query Q':")
print(f"CREATE TABLE JP_genQofD AS SELECT {projection_generalized} "
      f"FROM JP_gen_id_{table} NATURAL JOIN JP_gen_id_{join}")
DATABASE.close()

print("JP-query finished.")
```

A.8. sjp.py

```
import sqlite3
```

```
from pathlib import Path

def sjp(database: str, table: str, projection: list, join:str, attribute:str, operator:
    str, value:str):
    # convert path string to path
    db_file = Path(database)
    # create database connection
    DATABASE = sqlite3.connect(database=db_file)
    # define table_id
    table_id: str = table + 'ID'
    #define join_id
    join_id: str = join + 'ID'
    # concatenate projection attribute-list to string
    projection_string_wo_id: str = ','.join(projection)
    # add table_id to projection string
    projection_string: str = projection_string_wo_id + ',' + table_id + ',' + join_id

    value_string: str = fr'"{value}"'

    # generalize selection_operator
    if operator == "<":
        operator_gen = '<='
    elif operator == ">":
        operator_gen = ">="
    else:
        operator_gen = operator

    # selection string
    sel_string = attribute + operator + value_string


    #copy the table, add a random id, permutate via the new id
    id_table: str = 'id' + table
    DATABASE.execute(f"CREATE TABLE {id_table} AS SELECT *, random() AS {table_id} FROM {
        table} ORDER BY {table_id}")

    #copy the join table, add a random id, permutate via the new id
    id_join: str = 'id' + join
    DATABASE.execute(f"CREATE TABLE {id_join} AS SELECT *, random() AS {join_id} FROM {
        join} ORDER BY {join_id}")

    # calculating Q(D) with ID
    DATABASE.execute(f"CREATE TABLE SJP_QofD AS SELECT {projection_string} FROM {id_table
        } NATURAL JOIN {id_join} WHERE {sel_string}")

    # create list of column names in table
    cursort = DATABASE.execute(f"SELECT * FROM {table} LIMIT 1")
    columnNamest: list = [description[0] for description in cursort.description]
    cursorj = DATABASE.execute(f"SELECT * FROM {join} LIMIT 1")
    columnNamesj: list = [description[0] for description in cursorj.description]

    cursorx = DATABASE.execute(f"SELECT {table_id} FROM SJP_QofD")
    col = 0
    table_ids_list = [x[col] for x in cursorx]
    table_ids = tuple(table_ids_list)
    cursory = DATABASE.execute(f"SELECT {join_id} FROM SJP_QofD")
    join_ids_list = [y[col] for y in cursory]
    join_ids = tuple(join_ids_list)

    # create tables with relevant entries
    if len(table_ids) != 1:
```

```
    DATABASE.execute(f"CREATE TABLE prep_{table} AS SELECT * FROM {id_table} WHERE {
        table_id} IN {table_ids}")
else:
    DATABASE.execute(f"CREATE TABLE prep_{table} AS SELECT * FROM {id_table} WHERE {
        table_id} = {table_ids[0]}")
if len(join_ids) != 1:
    DATABASE.execute(f"CREATE TABLE prep_{join} AS SELECT * FROM {id_join} WHERE {
        join_id} IN {join_ids}")
else:
    DATABASE.execute(f"CREATE TABLE prep_{join} AS SELECT * FROM {id_join} WHERE {
        join_id} = {join_ids[0]}")
DATABASE.commit()

# generate projection string for generalised attributes to create generalized table
# add generalized attributes to projection list for Q' if they appear in Q
projection_list_generalisation: list = []
projection_generalized_wo_id: list = []
columns_to_generalise: list = []
attribute_gen: str
value_gen: str
for i in columnNamest:
    try:
        cursor2 = DATABASE.execute(f"SELECT * FROM {i} LIMIT 1")
        tuple2 = cursor2.fetchone()
        colCount: int = len(tuple2)
    except Exception:
        projection_list_generalisation.append(f"{i}")
        if i in projection:
            projection_generalized_wo_id.append(f"{i}")
        if i == attribute:
            attribute_gen = attribute
            value_gen = value_string
    else:
        if colCount == 1:
            token: str = tuple2[0]
            projection_list_generalisation.append(f"{i}")
            columns_to_generalise.append(f"{i}")
            DATABASE.execute(f"ALTER TABLE prep_{table} RENAME COLUMN {i} TO {i}old")
            DATABASE.execute(f"ALTER TABLE prep_{table} ADD COLUMN {i}")
            DATABASE.execute(f"UPDATE prep_{table} SET {i} = '{token}'")
            if i in projection:
                projection_generalized_wo_id.append(f"{i}")
            if i == attribute:
                attribute_gen = attribute
                value_gen = token
        if colCount == 2:
            projection_list_generalisation.append(f"{i}Gen")
            columns_to_generalise.append(f"{i}")
            if i in projection:
                projection_generalized_wo_id.append(f"{i}Gen")
            if i == attribute:
                attribute_gen = attribute + 'Gen'
                value_gen = DATABASE.execute(f"SELECT {attribute_gen} FROM {i} "
                                             f"WHERE {sel_string}").fetchone()[0]
        if colCount == 3:
            columns_to_generalise.append(f"{i}")
            projection_list_generalisation.append(f"{i}UG")
            projection_list_generalisation.append(f"{i}OG")
            if i in projection:
                projection_generalized_wo_id.append(f"{i}UG")
                projection_generalized_wo_id.append(f"{i}OG")
```

```
            if i == attribute:
                attribute_gen = attribute + 'UG'
                value_gen = DATABASE.execute(f"SELECT {attribute_gen} FROM {i} "
                                             f"WHERE {attribute} = {value}").fetchone
                                                 ()[0]
projection_string_generalisation: str = ','.join(projection_list_generalisation)

# generate join string for generalisation of table
generalisingJoin: str = ' NATURAL JOIN '.join(columns_to_generalise)

# generalisation of table' - creation of table' wo IDs
DATABASE.execute(f"CREATE TABLE gen_{table} AS SELECT {
    projection_string_generalisation} FROM prep_{table} "
                 f"NATURAL JOIN {generalisingJoin}")

# id generation and permutation for table'
DATABASE.execute(f"CREATE TABLE SJP_gen_id_{table} AS SELECT *, random() AS {table_id
    }"
                 f" FROM gen_{table} ORDER BY {table_id}")

# generate projection string for generalised attributes to create generalized join-
    table
# add generalized attributes to projection list for Q' if they appear in Q
projection_list_generalisation2: list = []
projection_generalized_wo_id2: list = []
columns_to_generalise2: list = []
for i in columnNamesj:
    try:
        cursor3 = DATABASE.execute(f"SELECT * FROM {i} LIMIT 1")
        tuple3 = cursor3.fetchone()
        colCount: int = len(tuple3)
    except Exception:
        projection_list_generalisation2.append(f"{i}")
        if i in projection:
            projection_generalized_wo_id2.append(f"{i}")
        if i == attribute:
            attribute_gen = attribute
            value_gen = value_string
    else:
        if colCount == 1:
            token: str = tuple3[0]
            projection_list_generalisation2.append(f"{i}")
            columns_to_generalise2.append(f"{i}")
            DATABASE.execute(f"ALTER TABLE prep_{join} RENAME COLUMN {i} TO {i}old")
            DATABASE.execute(f"ALTER TABLE prep_{join} ADD COLUMN {i}")
            DATABASE.execute(f"UPDATE prep_{join} SET {i} = '{token}'")
            if i in projection:
                projection_generalized_wo_id2.append(f"{i}")
            if i == attribute:
                attribute_gen = attribute
                value_gen = token
        if colCount == 2:
            projection_list_generalisation2.append(f"{i}Gen")
            columns_to_generalise2.append(f"{i}")
            if i in projection:
                projection_generalized_wo_id2.append(f"{i}Gen")
            if i == attribute:
                attribute_gen = attribute + 'Gen'
                value_gen = DATABASE.execute(f"SELECT {attribute_gen} FROM {i} "
                                             f"WHERE {sel_string}").fetchone()[0]
        if colCount == 3:
```

```
            columns_to_generalise2.append(f"{i}")
            projection_list_generalisation2.append(f"{i}UG")
            projection_list_generalisation2.append(f"{i}OG")
            if i in projection:
                projection_generalized_wo_id2.append(f"{i}UG")
                projection_generalized_wo_id2.append(f"{i}OG")
            if i == attribute:
                attribute_gen = attribute + 'UG'
                value_gen = DATABASE.execute(f"SELECT {attribute_gen} FROM {i} "
                                             f"WHERE {attribute} = {value}").fetchone
                                                 ()[0]
projection_string_generalisation2: str = ','.join(projection_list_generalisation2)

# generate join string for generalisation of join
generalisingJoin2: str = ' NATURAL JOIN '.join(columns_to_generalise2)

# generalisation of join - creation of join' wo IDs
DATABASE.execute(f"CREATE TABLE gen_{join} AS SELECT {
    projection_string_generalisation2} FROM prep_{join} "
                 f"NATURAL JOIN {generalisingJoin2}")

# id generation and permutation for join'
DATABASE.execute(f"CREATE TABLE SJP_gen_id_{join} AS SELECT *, random() AS {join_id}"
                 f" FROM gen_{join} ORDER BY {join_id}")

value_gen_string = rf'"{value_gen}"'
sel_string_generalized = attribute_gen + operator_gen + value_gen_string


# add ID to generalized projection list, form list to string, execution of Q'
projection_gen_list = projection_generalized_wo_id + projection_generalized_wo_id2
projection_gen_list.append(f"{table_id}")
projection_gen_list.append(f"{join_id}")
projection_generalized: str = ','.join(projection_gen_list)
DATABASE.execute(f"CREATE TABLE SJP_genQofD AS SELECT {projection_generalized} "
                 f"FROM SJP_gen_id_{table} NATURAL JOIN SJP_gen_id_{join} WHERE {
                     sel_string_generalized}")
DATABASE.commit()
DATABASE.close()

DATABASE = sqlite3.connect(database=db_file)
# deletion of prep_table
# deletion of gen_table without ID
DATABASE.execute(f"DROP TABLE {id_table}")
DATABASE.execute(f"DROP TABLE {id_join}")
DATABASE.execute(f"DROP TABLE prep_{table}")
DATABASE.execute(f"DROP TABLE prep_{join}")
DATABASE.execute(f"DROP TABLE gen_{table}")
DATABASE.execute(f"DROP TABLE gen_{join}")
DATABASE.commit()

QofD = DATABASE.execute("SELECT * FROM SJP_QofD").fetchall()
genQofD = DATABASE.execute("SELECT * FROM SJP_genQofD").fetchall()
genD = DATABASE.execute(f"SELECT * FROM SJP_gen_id_{table}").fetchall()
genD2 = DATABASE.execute(f"SELECT * FROM SJP_gen_id_{join}").fetchall()
print(f"Regular result SJP_QofD:")
print(QofD)
print(f"Generalized result SJP_genQofD:")
print(genQofD)
print(f"Intensional provenance-answer SJP_gen_id_{table} and SJP_gen_id_{join}:")
print(genD)
```

```
print(genD2)
print("Generalized query Q':")
print(f"CREATE TABLE SJP_genQofD AS SELECT {projection_generalized} "
      f"FROM SJP_gen_id_{table} NATURAL JOIN SJP_gen_id_{join} WHERE {
          sel_string_generalized}")
DATABASE.close()

print("SJP-query finished.")
```

B. Repositories

B.1. Quellen

Die Originalliteratur und weitere genutzte Quellen dieser Arbeit sind im dazugehörigen stud.ip dieser Masterarbeit zu finden.

B.2. Repository und Zugriff

Eine Zugriffserlaubnis für das GitHub-Repository kann auf Nachfrage per E-Mail an `tanja.auge@uni-rostock.de` erteilt werden. Der Link zum privaten Repository lautet: https://github.com/maexlamster/PuD

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit 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 31. August 2021