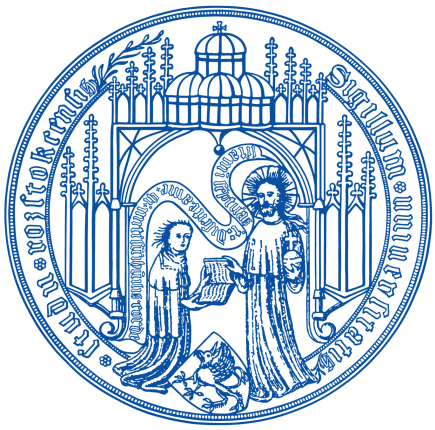
Berechnung von Quasi-Identifikatoren nach Verbundoperationen

Bachelorarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von:	Florian Rose
Matrikelnummer:	215204656
geboren am:	31.05.1996 in Rostock
Erstgutachter:	Prof. Dr. rer. nat. habil. Andreas Heuer
Zweitgutachter:	apl. Prof. Dr.-Ing. habil. Meike Klettke
Betreuer:	Hannes Grunert
Abgabedatum:	18.03.2019

Inhaltsverzeichnis

Kapitel 1

Einleitung

Heutzutage setzen Unternehmen verstärkt auf Cloudlösungen für ihre Dienste ([GS18]), da diese Vorteile in Hinblick auf Skalierbarkeit der Anwendung, Ausfallsicherheit und Kosteneffizienz versprechen. Allerdings geben sie damit auch Daten ihrer Kunden an Dritte weiter, die ggf. sensible Informationen enthalten, wie z. B. Krankheiten, Einkommen oder Ähnliches.

1.1 Motivation

Um aufzuzeigen, wie wichtig Datenschutz ist, betrachten wir das Beispiel eines Assistenzsystems. Systeme dieser Art umgeben uns immer häufiger, z. B. in Form von Fitnessarmbändern oder Smartwatches.

> **Definition:** Assistenzsystem nach [GH14]
> Assistenzsysteme sollen den Nutzer bei der Arbeit (Ambient Assisted Working) und in der Wohnung (Ambient Assisted Living) unterstützen. Durch verschiedene Sensoren werden Informationen über die momentane Situation und die Handlungen des Anwenders gesammelt. Diese Daten werden durch das System gespeichert und mit weiteren Daten, beispielsweise mit dem Facebook-Profil des Nutzers verknüpft. Durch die so gewonnenen Informationen lassen sich Vorlieben, Verhaltensmuster und zukünftige Ereignisse berechnen.

Angenommen, der Anbieter des Assistenzsystems lagert seine Infrastruktur in die Cloud bei einem großen Internetkonzern aus. So wären Daten, welche auf den Cloudservern ausgewertet werden, nicht gegen unerwünschte Analysen durch den Anbieter der Cloud geschützt. Dies ist insbesondere dann problematisch, falls die Daten nach der Erhebung ungefiltert an die Cloud gesendet werden.
Auch das Smartphone ist in diesem Sinn ein Assistenzsystem, da es den Zugang zu Informationen aus dem Internet auch mobil ermöglicht. Dies ist im Arbeitsumfeld von Vorteil, um z. B. E-Mails auch unterwegs zu lesen. Ein Beispiel dafür, dass solche Systeme nicht unbedingt den Datenschutz gewährleisten, war der Fall rund um Facebook und Cambridge Analytica (zusammengefasst in [NPO18]). Hier wurden durch eine Smartphone-App Daten der Nutzer ohne deren Zustimmung gesammelt, um gezielt die Meinungsbildung von Millionen von Facebooknutzern und potenziell sogar Wahlergebnisse zu beeinflussen.
Um sicherzustellen, dass die erhobenen Daten mit Personenbezug wirklich nur für den eigentlich vorgesehenen Verwendungszweck genutzt werden, gibt es mehrere Ansätze. Beispielsweise kann der Datenschutz schon bei der Anwendungsentwicklung durch die Nutzung von Sichtkonzepten und feingranularen Zugriffsrechten Einfluss finden. Häufig werden Daten nachträglich anonymisiert. Hierfür werden Verfahren basierend auf Maßen, wie k-Anonymität, verwendet, welche Quasi-Identifikatoren nutzen, um eine Anonymisierung zu gewährleisten (vgl. [ABA18]).

1.2 Problemstellung

Für die Anwendung von Anonymisierungsverfahren ist es Voraussetzung zu wissen, welche Attribute (beinahe) identifizierend sind. Diese Attribute werden als Quasi-Identifikatoren, im Nachfolgenden auch QI, bezeichnet.

> **Zitat:** Quasi-Identifikator nach [Dal86]
> A P-set of data is defined as unique in the set of N records in the following two cases:
>
> i) there is but one individual with that specific set of data; or
>
> ii) there are a small number k of individuals, say k = 2 or 3, with that specific set of data.

Dabei ist N die Anzahl der Tupel des Datenbestandes. Das *P-set* ist eine Menge von Attributen, für die Folgendes gilt:

> **Zitat:** Eigenschaften P-set [Dal86]
> Data for this set of variables may be in the public domain for a non-negligeable proportion of the census population.

Laut obiger Definition ist ein Quasi-Identifikator eine Menge von Attributen. Das bedeutet, jede Attributkombination ist potenziell ein QI. Folglich entspricht die Menge der QI-Kandidaten einer Tabelle der Potenzmenge ihrer Attributmenge X, mit Ausnahme der leeren Menge. Aus der Kombinatorik ist folgende Formel zur Berechnung der Mächtigkeit der Potenzmenge bekannt:

$$|\mathcal{P}(X)| = 2^{|X|}$$

Daraus folgt, dass die Anzahl der zu überprüfenden QI-Kandidaten exponentiell mit der Anzahl der Attribute wächst. Dies ist insbesondere für Verbundoperationen interessant. Bei Verbunden werden zwei oder mehr Relationen zu einer vereinigt, wodurch neue Quasi-Identifikatoren entstehen können.
Ziel der Arbeit ist es deshalb, Möglichkeiten zu finden, um Aussagen über die Quasi-Identifikatoren der Verbundrelation zu treffen, ohne dass existierende Algorithmen auf dem explodierten Suchraum ausgeführt werden. Dabei soll das Wissen über die Ausgangsrelationen, vor allem deren Quasi-Identifikatoren, genutzt werden, um die neuen Quasi-Identifikatoren zu berechnen bzw. bestimmte Attributkombinationen als QI auszuschließen.

1.3 Gliederung der Arbeit

Dieses Kapitel hat eine kurze Einleitung und Motivation für das Thema der Arbeit gegeben. Im nachfolgenden Kapitel (2) stellen wir kurz Grundlagen vor, welche die Wichtigkeit dieses Themas noch einmal unterstreichen. Anschließend werden in jenem Teil der Arbeit der Stand der Technik bezüglich Quasi-Identifikatoren, funktionalen Abhängigkeiten und Schlüsseleigenschaften vorgestellt. Im dritten Kapitel (3) wird das Konzept für die Lösung des in dieser Arbeit diskutierten Problems erläutert. Dabei werden Ansätze diskutiert, die zur Berechnung von Quasi-Identifikatoren einer Verbundrelation auf Basis der Ausgangsrelationen genutzt werden können. Weiter werden dort auch Ansätze für die Berechnung während des Verbundes skizziert. In Kapitel 4 besprechen wir die Implementation der Algorithmen, die aus dem Konzept hervorgehen. Hierbei werden auch Schwierigkeiten in der Implementierung diskutiert, welche zu alternativen Lösungen führen. Im darauffolgenden Kapitel (5) wird die Implementation anhand eines Beispieldatensatzes ausgewertet. Im letzten Kapitel (6) wird zunächst eine Zusammenfassung der Ergebnisse dieser Arbeit präsentiert. Abschließend werden noch offene Probleme diskutiert und besprochen.

Kapitel 2

Stand der Technik

In diesem Kapitel werden zunächst die Grundlagen des Datenschutzes und der Anonymisierung beschrieben. Im Anschluss werden Techniken zur Berechnung von Quasi-Identifikatoren gezeigt und auf verwandte Themengebiete eingegangen.

2.1 Grundlagen

Im Nachfolgenden werden die Grundlagen des technischen Datenschutzes vorgestellt. Dazu wird sowohl das rechtliche Fundament des Forschungsgebietes als auch eine erste Methodik für die Umsetzung des Datenschutzes erläutert.

2.1.1 Datenschutz

Der Datenschutz ist gesetzlich durch das Bundesdatenschutzgesetz geregelt. In §71 des Bundesdatenschutzgesetzes heißt es:

> **Zitat:** Datenschutz durch Technikgestaltung und datenschutzfreundliche Voreinstellungen [BDS17]
> (1) Der Verantwortliche hat sowohl zum Zeitpunkt der Festlegung der Mittel für die Verarbeitung als auch zum Zeitpunkt der Verarbeitung selbst angemessene Vorkehrungen zu treffen, die geeignet sind, die Datenschutzgrundsätze wie etwa die Datensparsamkeit wirksam umzusetzen, und die sicherstellen, dass die gesetzlichen Anforderungen eingehalten und die Rechte der betroffenen Personen geschützt werden. Er hat hierbei den Stand der Technik, die Implementierungskosten und die Art, den Umfang, die Umstände und die Zwecke der Verarbeitung sowie die unterschiedliche Eintrittswahrscheinlichkeit und Schwere der mit der Verarbeitung verbundenen Gefahren für die Rechtsgüter der betroffenen Personen zu berücksichtigen. Insbesondere sind die Verarbeitung personenbezogener Daten und die Auswahl und Gestaltung von Datenverarbeitungssystemen an dem Ziel auszurichten, so wenig personenbezogene Daten wie möglich zu verarbeiten. Personenbezogene Daten sind zum frühestmöglichen Zeitpunkt zu anonymisieren oder zu pseudonymisieren, soweit dies nach dem Verarbeitungszweck möglich ist.
> (2) Der Verantwortliche hat geeignete technische und organisatorische Maßnahmen zu treffen, die sicherstellen, dass durch Voreinstellungen grundsätzlich nur solche personenbezogenen Daten verarbeitet werden können, deren Verarbeitung für den jeweiligen bestimmten Verarbeitungszweck erforderlich ist. Dies betrifft die Menge der erhobenen Daten, den Umfang ihrer Verarbeitung, ihre Speicherfrist und ihre Zugänglichkeit. Die Maßnahmen müssen insbesondere gewährleisten, dass die Daten durch Voreinstellungen nicht automatisiert einer unbestimmten Anzahl von Personen zugänglich gemacht werden können.

Dabei sollten vor allem die zwei Kernkonzepte des Datenschutzes, Datensparsamkeit und Datenvermeidung, berücksichtigt werden. Konkret bedeutet das nach [GH16], dass

- nur für den Verwendungszweck unabdingbare Daten, die nicht bereits früher ausgewertet werden können, weitergegeben werden,
- nur so wenig personenbezogene Daten wie möglich verwendet werden, und
- diese Daten dann frühestmöglich anonymisiert/pseudonymisiert und wieder gelöscht werden.

In der Praxis ist dies aber nur selten der Fall, besonders im Bereich der mobilen Endgeräte. Smartphones und deren Anwendungen sammeln dauerhaft Daten. Wetter-Apps, deren Funktionalität an eingeschaltete Standorterkennung gebunden ist, sind nur ein Beispiel.

2.1.2 Anonymisierung

Um zu verhindern, dass sensible Daten mit den dazugehörigen Personen in Verbindung gebracht werden können, ist eine Anonymisierung der Daten nötig. Die bekanntesten Anonymisierungskonzepte sind k-Anonymität und seine Erweiterungen.

k-Anonymität

Um k-Anonymität [SS98] zu erreichen, können Daten unterdrückt oder generalisiert werden. Die Attribute werden hierbei in drei Kategorien eingeteilt:

- **Schlüssel**
  Schlüssel sind eindeutig identifizierende Attribute. Schlüssel werden immer aus der Menge der Attribute entfernt, sofern sie nicht für weitere Auswertungen benötigt werden.
- **Quasi-Identifikatoren**
  Quasi-Identifikatoren sind Attribute, die in Kombination mit anderen Daten einzelne Tupel identifizieren können. Die oben genannten Methoden sorgen dafür, dass die Quasi-Identifikatoren diese Eigenschaften im anonymisierten Datensatz verlieren.
- **sensible Attribute**
  Sensible Attribute sind diejenigen, deren Daten nicht mit den dazugehörigen Personen öffentlich einsehbar sein sollen.

Dabei ist es auch möglich, dass diese Kategorien überlappen, das heißt, Attribute können sowohl Quasi-Identifikator als auch sensibles Attribut sein. Betrachten wir zur Veranschaulichung folgendes Beispiel einer fiktiven Patientendatenbank:

Vorname	Nachname	Alter	Geschlecht	Geburtsort	Krankheit
Volker	Ferber	29	männlich	Rostock	Krebs
Edmund	Hertel	29	männlich	Rostock	Herzerkrankung
Arthur	Ross	33	männlich	Schwerin	Knochenfraktur
Jakob	Hagelstein	39	männlich	Schwerin	Herzerkrankung
Natali	Scherzinger	38	weiblich	Wismar	Knochenfraktur
Angelika	Morgentaler	39	weiblich	Wismar	Krebs
Monika	Semmelrogge	34	weiblich	Rostock	Herzerkrankung
Sven	Schnell	37	männlich	Schwerin	unbekannt
Martin	Geier	27	männlich	Rostock	Herzerkrankung
Karin	Essig	34	weiblich	Rostock	Virusinfektion

Tabelle 2.1: Ausgangsrelation der Patientendatenbank;
Schlüssel (rosa), Quasi-Identifikatoren (gelb) und sensible Attribute (violett) farblich hervorgehoben

Diese Tabelle wird nun Schritt für Schritt durch Generalisierung in eine 2-anonyme Form überführt. Offensichtlich sind Vorname und Nachname in diesem Beispiel Schlüssel und werden deshalb entfernt (im Folgenden durch ein * repräsentiert).

Vorname	Nachname	Alter	Geschlecht	Geburtsort	Krankheit
*	*	29	männlich	Rostock	Krebs
*	*	29	männlich	Rostock	Herzerkrankung
*	*	33	männlich	Schwerin	Knochenfraktur
*	*	39	männlich	Schwerin	Herzerkrankung
*	*	38	weiblich	Wismar	Knochenfraktur
*	*	39	weiblich	Wismar	Krebs
*	*	34	weiblich	Rostock	Herzerkrankung
*	*	37	männlich	Schwerin	unbekannt
*	*	27	männlich	Rostock	Herzerkrankung
*	*	34	weiblich	Rostock	Virusinfektion

Tabelle 2.2: Patientenrelation nach Unterdrückung der Schlüssel

Betrachtet man nun die Attribute Alter, Geschlecht und Geburtsort, fällt auf, dass diese immer noch identifizierend sind. Beispielsweise gibt es nur eine Person mit $Alter = 33$. Deshalb wird dieses Attribut nun generalisiert. Um eine Generalisierung durchzuführen, wird eine Generalisierungshierarchie benötigt. Solche Hierarchien hängen immer vom Attribut selbst ab und nicht rein von dem Typ des Attributes. So ließen sich beispielsweise Orte zu Großräumen, Bundesländern und schließlich Regionen generalisieren. Im Fall des Alters werden statt absolute Werte nun Intervalle benutzt, sodass das Alter kein Quasi-Identifikator mehr ist.

Vorname	Nachname	Alter	Geschlecht	Geburtsort	Krankheit
*	*	$Alter < 30$	männlich	Rostock	Krebs
*	*	$Alter < 30$	männlich	Rostock	Herzerkrankung
*	*	$30 <= Alter < 40$	männlich	Schwerin	Knochenfraktur
*	*	$30 <= Alter < 40$	männlich	Schwerin	Herzerkrankung
*	*	$30 <= Alter < 40$	weiblich	Wismar	Knochenfraktur
*	*	$30 <= Alter < 40$	weiblich	Wismar	Krebs
*	*	$30 <= Alter < 40$	weiblich	Rostock	Herzerkrankung
*	*	$30 <= Alter < 40$	männlich	Schwerin	unbekannt
*	*	$Alter < 30$	männlich	Rostock	Herzerkrankung
*	*	$30 <= Alter < 40$	weiblich	Rostock	Virusinfektion

Tabelle 2.3: 2-anonyme Patientenrelation; Äquivalenzklassen farblich hervorgehoben

Die Tabelle ist nun 2-anonym in Bezug auf Geburtsort, Geschlecht und Alter, da es für jede Äquivalenzklasse über diese Attribute mindestens zwei Tupel gibt. Es ist anhand dieser Daten nicht möglich mit einer Sicherheit von mehr als 50% $(1/k)$ zu sagen, welcher Patient zu welchem Datensatz gehört, sofern man nur das Alter, das Geschlecht und den Geburtsort kennt.

Jedoch ist k-Anonymität nicht in jedem Fall ein gutes Mittel zur Anonymisierung. Es gibt Fälle, in denen eine Tabelle k-anonym ist, die sensiblen Daten einer Person jedoch dennoch (mit einer ausreichenden Genauigkeit) bestimmt werden können. Wird beispielsweise das erste Tupel (Volker Ferbel) aus Tabelle 2.3 entfernt, weil dieser in ein anderes Krankenhaus verlegt wird, bleibt die Relation 2-anonym. Allerdings ist das sensible Attribut Krankheit für alle Patienten, die jünger als 30 Jahre sind, gleich. Dadurch kann Edmund Hertel dann doch seine Herzerkrankung zugewiesen werden, sofern man um sein Alter und seinen Aufenthalt in diesem Krankenhaus weiß (Homogenitätsattacke [MGKV06]).
Weiter kann auch Hintergrundwissen genutzt werden, um die Zuordnung einer Person zu einem Datensatz wahrscheinlicher zu machen. Ist beispielsweise bekannt, dass die 38 Jahre alte Natali Scherzinger aus Wismar erst vor kurzem in den Skiurlaub gefahren ist, so wäre es sehr wahrscheinlich, dass sie die Patientin mit Knochenfraktur aus der grünen Äquivalenzklasse ist (Angriff mittels Hintergrundwissen [MGKV06]).

l-Diversität

l-Diversität ist eine Erweiterung der k-Anonymität. Hier soll die Schwäche von k-Anonymität bei Attributwerten, die besonders häufig bzw. selten vorkommen, beseitigt werden. Dazu wird die Verteilung der Werte des sensiblen Attributes in den Äquivalenzklassen (q^*-Blöcke) betrachtet.

> **Zitat:** l-Diversität nach [MGKV06]
> Let us define a q^*-block be the set of tuples in [a table] T^* whose nonsensitive attribute values generalize to q^*. [...] A q^*-block is l-diverse if it contains l "well represented" values for the sensitive attribute S. A table is l-diverse, if every q^*-block is l-diverse.

Dies verbessert besonders die Schwäche der k-Anonymität hinsichtlich der Homogenitätsattacke.

t-Closeness

l-Diversität geht zwar einen wichtigen Schritt über k-Anonymität hinaus, geht aber nicht weit genug. Es verhindert die Zuordnung von Tupeln zu der entsprechenden Person, verhindert aber nicht unbedingt die Zuordnung des sensiblen Attributes zu der dazugehörigen Person (Attribute Disclosure). In [LLV07] werden folgende Schwächen von l-Diversität beschrieben:

- l-Diversität kann schwer zu erreichen und unnötig sein
- l-Diversität ist für das Verhindern von Attribute Disclosure unzulänglich

> **Definition:** Attribut Disclosure nach [MNSS15]
> Attribut Disclosure triff auf, falls sensible Informationen über eine Person aus einem Datenbestand gewonnen werden können, ohne dass ihr Datensatz im Datenbestand identifiziert wird.

Deshalb wird das Konzept t-Closeness eingeführt. Hier wird die Verteilung der Attributwerte beachtet.

> **Zitat:** Das Prinzip t-Closeness nach [LLV07]
> An equivalence class is said to have t-closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than threshold t. A table is said to have t-closeness if all equivalence classes have t-closeness.

Das Abstandsmaß wird dabei wie der Abstand zwischen Wahrscheinlichkeitsverteilungen berechnet, z. B. mit der Kullback-Leibler-Divergenz ([KL51]).

2.2 Finden von funktionalen Abhängigkeiten

Im Folgenden wird vorgestellt, wie funktionale Abhängigkeiten (nachfolgend auch als FDs abgekürzt) auf Basisrelationen gefunden werden können. Dies ist deshalb relevant, weil Quasi-Identifikatoren der Eigenschaft von funktionalen Abhängigkeiten ähneln. Sie erfüllen diese für einen sehr großen Anteil der Tupel, aber nicht für alle. Funktionale Abhängigkeiten sind wie folgt definiert:

Definition: funktionale Abhängigkeiten nach [SSH13]
Funktionale Abhängigkeiten, sind Ausdrücke der Form: $A \rightarrow B$
Dabei sind A und B Mengen von Attributen der Relation R. $A \rightarrow B$ gilt auf einer Instanz r der Relation R, falls für jede Kombination von Attributwerten der Attribute aus A maximal eine Kombination von Attributwerten der Attribute aus B existiert. Formal, ohne Beschränkung der Allgemeinheit, für $\|A\| = 1$:
$\forall a \in A : \|\pi_B(\sigma_{A=a}(r))\| \leq 1$

2.2.1 Auf Basisrelationen

Betrachten wir zunächst den Fall, dass die Tabelle eine Basisrelation ist. Es stehen zum Finden von funktionalen Abhängigkeiten also nur die Informationen zur Verfügung, die aus den Tupeln oder dem Schema der Relation entnommen können werden.

Echte funktionale Abhängigkeiten

Funktionale Abhängigkeiten zwischen Attributen sind in der Regel auf einen semantischen Zusammenhang der Attribute zurückzuführen. Bei der Suche nach funktionalen Abhängigkeiten ist zu beachten, dass man nicht immer davon ausgehen kann, dass die Ausgangsrelation mindestens in der dritten Normalform vorliegt. In [Kle98] werden folgende Heuristiken für das Finden von potenziellen FDs betrachtet:

Zitat: Heuristiken zum Finden von FDs nach [Kle98]

H_F1 [...] Gibt es in einer Relation mehrere Attribute ABC, die als Integer definiert sind so wird überprüft, ob sich die Werte eines Attribute[s] (C) aus zwei anderen Attribute[n] (AB) durch Addition oder Multiplikation errechnen lassen. Ist das der Fall, so wird die funktionale Abhängigkeit $AB \rightarrow C$ vermutet.

H_F2 [...] [Die unbekannten funktionalen Abhängigkeiten] könnten geltende funktionale Abhängigkeiten sein. Es können aber auch negierte funktionale Abhängigkeiten sein, die noch nicht erkannt wurden, weil in den Daten keine Tupel vorhanden sind, die die entsprechende funktionale Abhängigkeit widerlegen. Man kann versuchen, diese beiden Fälle durch folgende Heuristik zu unterscheiden: Es wird untersucht, welche Attributmengen *gleiche Einträge* bei mehreren Tupeln aufweisen. [...] Die Heuristik liefert bei größeren Relationen differenziertere Bewertungen für die Kandidaten für funktionale Abhängigkeiten. Je mehr gleiche Werte in den Relationen auftreten, desto wahrscheinlicher müssen funktionale Abhängigkeiten gelten [...]

H_F3 [...] Die Attribute einer Update-Operation, die zur Identifikation der zu ändernden Tupel verwendet wurden, bilden dabei die linke Seite der Abhängigkeit, die Attribute, die in der Operation geändert wurden, die rechte Seite. Diese abgeleitete funktionale Abhängigkeit muß in der Relation gegolten haben, auf die die Transaktion angewendet worden ist. Man kann deshalb vermuten, daß diese funktionale Abhängigkeit auch allgemeingültig ist sie muß jedoch ebenfalls validiert werden.

H_F4 Meist werden inhaltlich zusammenhängende Attribute auch im Zusammenhang eingegeben und stehen deshalb nebeneinander in der Relation. Man kann deshalb davon ausgehen, daß zwischen Attributen, die dicht nebeneinander in der Relation auftreten, eher funktionale Abhöngigkeiten auftreten als zwischen Attributen, die in der Relation weit auseinander liegen. Diese Heuristik kann nur im Zusammenhang mit anderen Heuristikregeln verwendet werden, da die Ergebnisse zu vage und zu wenig differenzierend sind.

H_F5 Für eine zu untersuchende Abhängigkeit wird überprüft, wie die Abhängigkeiten in der Nähe zu dieser Abhängigkeit aussehen. [...] Soll die Abhängigkeit zwischen der Attributmenge X und dem Attribut Y festgelstellt werden ($X \xrightarrow{?} Y$), so wird in dieser Prozedur untersucht, wie die Abhängigkeiten $X \xrightarrow{?} (Y-2)$ $X \xrightarrow{?} (Y-1)$ $X \xrightarrow{?} (Y+1)$ $X \xrightarrow{?} (Y+2)$ aussehen, wobei mit $(Y-1)$ das Attribut gemeint ist, das links neben dem Attribut Y in der Relation steht usw. Es müssen dabei triviale funktionale ABhängigkeiten ausgeschlossen werden d.h, ist die rechte Seite der zu überprüfenden Abhängigkeit in der linken Seite enthalten, so kann daraus keine Aussage getroffen werden. Sind die Abhängigkeiten in der Nähe überwiegend funktionale AB-hängigkeiten, so deutet das darauf, daß die untersuche Abhängigkeit ebenfalls funktional ist un umgekehrt.

Dabei hätten H_F1 und H_F2 besondere Aussagekraft. H_F3 und H_F5 seien aussagekräftig, während H_F4 nur im Zusammenhang mit anderen Heuristiken sinnvoll sei.

Schlüssel

Da Schlüssel spezielle FDs sind (Schlüssel bestimmen alle anderen Attribute einer Relation) werden diese in diesem Abschnitt gesondert betrachtet. Identifizierende Attributmengen und Schlüssel sind wie folgt definiert:

Definition: Identifizierende Attributmenge, Schlüssel nach [SSH13]
Eine Menge I von Attributen der Relation R heißt identifizierenden Attributmenge der Instanz r von R, falls:
$\forall t_1, t_2 \in r : t_1 \neq t_2 \rightarrow \exists A : t_1(A) \neq t_2(A)$, wobei $A \in I$
Eine identifizierende Attributmenge K heißt Schlüssel, falls:
$\nexists M : M \subset K$ mit M ist identifizierende Attributmenge

Schlüsselkandidaten sind identifizierende Attributmengen, die als Schlüssel für die Relation infrage kommen. In [Kle98] werden folgende Heuristiken für das Finden von Schlüsselkandidaten vorgestellt:

Zitat: Heuristiken zum Finden von Schlüsselkandidaten nach [Kle98]

H_S1 Oft werden in Datenbanken Schlüssel künstlich eingeführt. Wenn ein Attribut mit dem Typ Integer und einer sehr großen Länge vereinbart wurde, so könnte dieses Attribut ein Schlüssel sein.

H_S2 Attributnamen können Hinweise auf Schlüssel liefern. [...] Kommt in einem Attributnamen ein Teilstring -name-, -nummer-, -nr-, -identifikator-, -id-, -key-, -kennzahl-, -knz-, -code-, -bezeichnung-, -bez-, -#-, usw. vor, so deutet das darauf hin, daß dieses Attribut eines Schlüssels des Relationen-Schemas ist. [...]

H_s3 Sind in den Daten der Relationen bei einem Attribut laufende Nummern eingetragen, so deutet das stark darauf hin, daß dieses Attribut ein Schlüssel oder Teil eines Schlüssels ist. [...]

> H_S4 Attribute, die viele verschiedene Werte in einer Relation annehmen, sind eher Schlüssel oder Teil des Schlüssels als Attribute, die nur wenige verschiedene Werte in der Relation annehmen. [...]
>
> H_S5 Vermutungen über Schlüssel lassen sich auch aus der Menge der bekannten funktionalen und negierten Abhängigkeiten ableiten. Ein Attribut, das auf der linken Seite vieler funktionaler Abhängigkeiten steht [...], ist mit größerer Wahrscheinlichkeit Teil eines Schlüssels als ein Attribut, das auf der linken Seite vieler negierter funktionaler Abhängigkeiten steht. [...] Steht es auf der rechten Seite einer negierten funktionalen Abhängigkeit, [...] so deutet das wiederum darauf hin, daß es Schlüssel oder Teil eines Schlüssels der Relation ist. [...]
>
> H_S6 Aus bekannten Transaktionen lassen sich Schlüsselkandidaten ableiten. Die Attribute, die bei Updates nie oder nur sehr selten verändert werden, sind mit größerer Wahrscheinlichkeit Schlüssel oder Teil des Schlüssels als die Attribute, die oft geändert werden.
> Attribute, die in Transaktionen zur Identifikation von Tupeln einer Relation verwendet werden, [...] können Schlüssel des Relationen-Schemas sein.

Dabei hätten die Heuristiken H_S3, H_S5 und H_S6 besondere Aussagekraft. Die ersten beiden Heuristiken seien aussagekräftig, während die vierte Heuristik nur im Zusammenhang mit anderen Heuristiken sinnvoll sei.

2.2.2 Auf (Verbund-)Sichten

Das Finden von funktionalen Abhängigkeiten auf Sichten bezieht sich auf die Frage danach, welche FDs sich für Sichtrelation ableiten lassen, wenn die FDs der Basisrelationen der Sicht gegeben sind. Im Folgenden werden dabei nur Verbundsichten betrachtet.

Echte funktionale Abhängigkeiten

In [Klu80] wird gezeigt, dass das Problem der implizierten FDs für allgemeine Relationenalgebra nicht lösbar ist. Betrachtet man allerdings die Relationenalgebra ohne Mengendifferenz, so existiert ein Algorithmus zur Berechnung von funktionalen Abhängigkeiten auf Ausdrücken der Relationenalgebra. In [JAK82] wird vorgestellt, wie das Problem der implizierten funktionalen Abhängigkeiten gelöst werden kann. Hier werden funktionale Abhängigkeiten in der Skolemform dargestellt.

Beispiel: Studentendatenbank, abgeleitet von [JAK82]
Betrachten wir zur Veranschaulichung das Beispiel des universitären Umfeldes. Gegeben den Studenten und den Ansprechpartnern der jeweiligen Studienbüros, suchen wir für jeden Studenten die entsprechenden Kontaktperson. Dabei seien folgende Ausgangsrelationen gegeben:

Student((Matrikelnummer, Vorname, Nachname, Studiengang, Semester),
Primary Key{Vorname, Nachname},
Unique{Matrikelnummer})

Ansprechpartner((MitarbeiterID, Studiengang, Fakultät),
Primary Key{MitarbeiterID, Studiengang},
FD$\{Studiengang\} \rightarrow \{Fakultät\}$)

Die Sicht *View_StudentFakultät(Vorname, Nachname, Studiengang, Ansprechpartner, Fakultät)* wird durch folgende Anfrage aufgebaut:

```
SELECT Vorname, Nachname, Studiengang, Fakultät, MitarbeiterID AS
Ansprechpartner
FROM Student, Ansprechpartner
WHERE Student.Studiengang = Ansprechpartner.Studiengang
```

Die Skolemform der funktionalen Abhängigkeit $\{Vorname, Nachname\} \rightarrow \{Fakultät\}$, welche transitiv durch die FDs der Ausgangsrelationen entsteht ($\{Vorname, Nachname\} \rightarrow \{Studiengang\}$ und $\{Studiengang\} \rightarrow \{Fakultät\}$), lautet:
$\forall Vorname$, $Nachname$, $Studiengang_1$, $Studiengang_2$, $Ansprechpartner_1$, $Ansprechpartner_2$, $Fakultät_1$, $Fakultät_2$:

$$\begin{aligned} &\neg View\_StudentFakultät(Vorname, Nachname, Studiengang_1, Ansprechpartner_1, Fakultät_1) \\ \vee &\neg View\_StudentFakultät(Vorname, Nachname, Studiengang_2, Ansprechpartner_2, Fakultät_2) \\ \vee \ &Fakultät_1 = Fakultät_2 \end{aligned}$$

Wobei sich $View\_StudentFakultät(Vorname, Nachname, Studiengang, Ansprechpartner, Fakultät)$ auf die Existenz eines solchen Tupels in der Sicht bezieht. Zusätzlich lässt sich $View\_StudentFakultät(Vorname, Nachname, Studiengang, Ansprechpartner, Fakultät)$ auf $Student(\_, Vorname, Nachname, Studiengang, \_) \wedge Ansprechpartner(Ansprechpartner, Studiengang, Fakultät)$ reduzieren, da es sich bei $View\_StudentFakultät$ um eine einfache Verbundsicht handelt ("_" steht hierbei für einen beliebigen Wert).

Somit ist die Frage, ob eine funktionale Abhängigkeit auf der Sichtrelation gilt, unter Betrachtung der Ausgangsrelationen und der Sichtdefinition beantwortbar.

Schlüssel

Die Suche nach Schlüsseln von Sichtrelationen ist einfach beantwortet. Als Schlüssel der Sichtrelation eignet sich immer ein zusammengesetzter Schlüssel aus den Schlüsseln der Ausgangstabellen. Nur die Minimalität des Schlüssels ist hierbei nicht garantiert, z. B. falls alternative Schlüssel existieren.

Beispiel: Studentendatenbank
Betrachten wir wieder das Studentenbeispiel. Diesmal wollen wir die Noten eines Studenten mit dessen Informationen aus der Studenten-Relation zusammenführen, beispielsweise um die Berechnung der Durchschnittsnoten für jeden Studiengang durchführen zu können, ohne dass der Verbund der Relationen jedes mal neu durchgeführt werden muss.

Student((Matrikelnummer, Vorname, Nachname, Studiengang, Semester),
Primary Key{Vorname, Nachname},
Unique{Matrikelnummer})

Note((Matrikelnummer, Modulnummer, Note),
Primary Key{Matrikelnummer, Modulnummer})

View_StudentenBewertung(Matrikelnummer, Vorname, Nachname, Modulnummer, Note), welche durch folgende Anfrage aufgebaut wird:

```
SELECT Matrikelnummer, Vorname, Nachname, Modulnummer, Note
FROM Student, Note
WHERE Student.Matrikelnummer = Note.Matrikelnummer
```

Hier wäre $\{Vorname, Nachname, Matrikelnummer, Modulnummer\}$ eine identifizierende Attributmenge, $\{Matrikelnummer, Modulnummer\}$ sowie $\{Vorname, Nachname, Modulnummer\}$ wären aber ebenfalls identifizierende Attributmengen. Diese sind jeweils zusätzlich minimal und damit mögliche Schlüssel.

2.3 Finden von Quasi-Identifikatoren

Das Finden von Quasi-Identifikatoren bedient sich einiger Ansätze der vorausgegangenen Abschnitte. Im Nachfolgenden werden dabei nur Basisrelationen betrachtet. Folgende Verfahren lassen sich grundsätzlich, wie in den Abschnitten davor, auch auf Sichten anwenden, indem man sie als Basisrelation betrachtet. Dabei werden dann aber potenzielle Verbesserungen durch Informationen der Ausgangsrelationen vernachlässigt. Da das Finden dieser Verbesserungsmöglichkeiten Ziel dieser Arbeit ist, wird die Betrachtung von Verbundsichten explizit in den nachfolgenden Kapiteln behandelt.

2.3.1 Spaltenorientierte Verfahren

Spaltenorientierte Verfahren bauen die Menge der möglichen QI-Kandidaten als Gitternetz [GH14] auf. Dieses Gitternetz wird dann traversiert und für jeden einzelnen Knoten entschieden, ob es sich um einen Quasi-Identifikator handelt oder nicht. Hierbei kann Bottom-Up oder Top-Down als Ansatz gewählt werden.

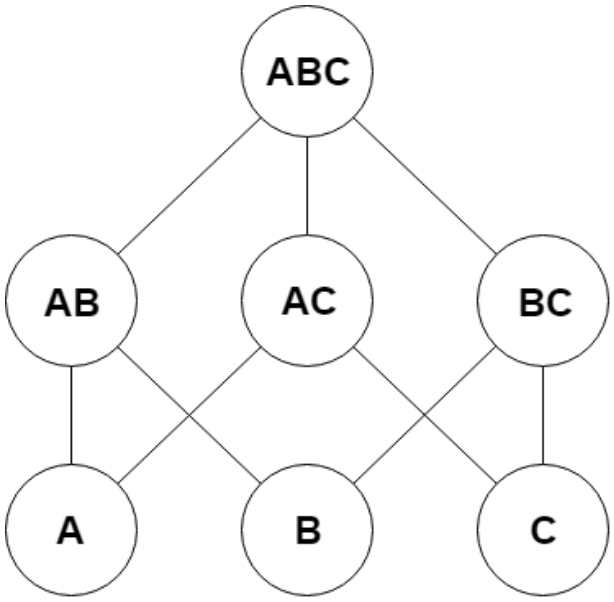


Abbildung 2.1: Gitternetz der Relation $R = \{A, B, C\}$

Bottom-Up hat den Vorteil, dass die Minimalität eines gefundenen QIs direkt gegeben ist. Wurde Knoten X im Bottom-Up-Verfahren als QI identifiziert, können alle Knoten Y mit $X \subset Y$ bei der Suche ignoriert werden, da sie nur eine Spezialisierung des QIs X sind. Beispielsweise muss die zweite Ebene des obigen Gitternetzes nicht mehr untersucht werden, falls A und B bereits als QIs und C als Nicht-QI bekannt sind. Grund dafür ist, dass es keine weiteren Attributkombinationen gibt, die weder A noch B enthalten. Nachteil dieses Ansatzes ist aber, dass QIs, die aus sehr vielen Attributen zusammengesetzt sind, erst spät entdeckt werden. Auch die Erkenntnis, dass die Relation keine Quasi-Identifikatoren enthält, wird erst am Ende des Algorithmus gewonnen.
Der Top-Down-Ansatz hat entsprechend genau gegenteilige Vor- und Nachteile. Ist ein Knoten X kein Quasi-Identifikator, so kann man in der Suche alle Knoten Y mit $Y \subset X$ ignorieren. Daher funktioniert der Top-Down-Ansatz besonders gut, wenn die Quasi-Identifikatoren aus sehr vielen Attributen zusammengesetzt sind bzw. gar keine QIs existieren. Jedoch erkennt man minimale Quasi-Identifikatoren erst nach Ende des Algorithmus.
Da sich die Ansätze in ihren Vor- und Nachteil jeweils gut ergänzen, lassen sich auch beide kombinieren [GH14]. Hierbei wird beim Traversieren des Suchraumes zwischen Top-Down und Bottom-Up gewechselt. Dies kann durch eine Wichtungsfunktion, die berechnet, welcher Ansatz im nächsten Schritt der beste ist, verbessert werden. Diese kann z. B. verwenden, dass Top-Down am Anfang langsamer sein wird als Bottom-Up. Dies ist darauf zurückzuführen, dass große Attributkombinationen mehr Speicherplatz benötigen, und damit eher Seiten aus dem Hauptspeicher verdrängen, die noch gebraucht werden, als kleine Attributkombinationen. Zudem ist der Aufwand für das Finden von Duplikaten einer Attributkombination proportional zur Anzahl der Attribute.
Da QIs in der Regel einige wenige Attributkombinationen sind, werden beim Bottom-Up-Verfahren zunächst viele Nicht-QIs entdeckt. Top-Down werden dahingegen zwar früh viele QIs entdeckt, deren Minimalität ist allerdings bis zunächst ungeklärt. Zudem ist hier die Berechnungsdauer durch die großen Attributkombinationen erhöht. Deshalb skalieren spaltenorientierte Verfahren im Allgemeinen schlecht mit zunehmender Anzahl an Attributen [PN17].

2.3.2 Zeilenorientierte Verfahren

Unique Column Combinations (UCCs) sind Quasi-Identifikatoren mit einem Grenzwert von 1. Das heißt, es existieren keine zwei Tupel mit den gleichen Werten bezüglich der Attribute der UCC. Zeilenorientierte Verfahren vergleichen einzelne Tupel der Relation und werden in [PN17] für Unique Column Combinations wie folgt beschrieben:

> **Zitat:** zeilenorienterte Verfahren nach [PN17]
> [Row-based] discovery strategies compare all records pair-wise and derive agree set[s] from these comparisons. An agree set is a negative observation, i.e., a set of attributes that have [the] same values in the two compared records and can, hence, not be a UCC; so agree sets correspond to non-UCCs in the attribute lattice. When all (or some) agree sets have been collected, there are efficient techniques to turn them into true UCCs.

Ein Agree-Set ist die Menge der Attributkombinationen, die für zwei verglichene Tupel den gleichen Wert besitzen. Da das paarweise Vergleichen aller Tupel auf großen Datenbeständen allerdings nicht in angemessener Zeit möglich ist, wird nur ein Ausschnitt des Datenbestandes betrachtet. Diese Ergebnisse sind wahrscheinlich nicht alle korrekt, weil durch das Sampling einige Agree-Sets nicht gefunden wurden. In [PN17] werden dabei folgende Eigenschaften der Zwischenergebnisse durch die Berechnung auf einem Sample r' der Originalrelation r herausgestellt:

- Vollständigkeit:
  Da alle Obermengen der UCCs im Ergebnis als korrekte UCCs angenommen werden, ist die Menge der $UCCs$ von r' vollständig: Sie impliziert die Menge aller $UCCs$ von r, d. h., für jedes X aus $UCC(r)$ existiert mindestens ein X' in $UCC(r')$ mit $X' \subset X$.

- Minimalität:
  Falls eine minimale UCC von r' wirklich gilt, dann muss die UCC auch minimal in Bezug auf das wirkliche Ergebnis sein. Deshalb kann das Sampling nicht zu nicht-minimalen oder unvollständigen Ergebnissen führen.

- Nähe:
  Falls eine minimale UCC von r' doch für r ungültig ist, dann ist diese dennoch nahe einer oder mehrerer Spezialisierungen, welche dann in r gelten.

2.3.3 Hybride Verfahren

In [PN17] wird eine hybride Strategie beschrieben, um UCCs zu bestimmen. Hierbei wird der zeilenorientierte Ansatz in der sogenannten Samplingphase und das spaltenorientierte Verfahren mit Bottom-Up-Ansatz in der Validierungsphase genutzt. In der Samplingphase werden viele Nicht-UCCs übersprungen. Dadurch kann der Nachteil des spaltenorientierten Bottom-Up-Verfahrens ausgeglichen werden, da viele Nicht-UCCs bereits als solche erkannt wurden. Die Validierungsphase produziert dann ein gültiges Ergebnis. Es wird immer wieder zwischen Sampling- und Validierungsphase gewechselt, sobald die aktuelle Phase ineffizient wird. Die Samplingphase wird verlassen, falls die Anzahl der neuen Agree-Sets pro Vergleich unter einen Grenzwert fällt, die Validierungsphase wird ineffizient, sobald die Anzahl der gültigen UCCs pro Nicht-UCC unter einen Grenzwert fallen. Mit jeder Iteration wird der Grenzwert gelockert. Dadurch wird sichergestellt, dass immer das momentan effizientere Verfahren genutzt wird. Dadurch skaliert das hybride Verfahren besser mit der Anzahl der Tupel und Attribute eines Datenbestandes.

Kapitel 3

Konzept

Um das Problem zu lösen, wie Quasi-Identifikatoren nach dem Verbund von Tabellen berechnet werden können, werden die bereits existierenden Algorithmen leicht modifiziert. Im Nachfolgenden wird der Ansatz kurz beschrieben und dann die Konzepte und Methoden erläutert, welche für die Berechnung genutzt werden. Hier werden dabei nur natürliche Verbunde betrachtet. Für andere Arten von Equi-Joins gelten die nachfolgenden Aussagen analog. Nur bei Outer-Joins gibt es einen Zusatz. Dadurch dass dort keine Tupel herausfallen können, müssen hier keine weiteren Einschränkungen bezüglich des Entfallens von Äquivalenzklassen gemacht werden, sodass die Nicht-QIs der Ausgangsrelationen immer auch auf der Verbundrelation gelten.

3.1 Beschreibung des Ansatzes

Ähnlich wie beim hybriden Verfahren zur Berechnung von UCCs (siehe 2.3.3) liegt der Problemlösung ein spaltenorientiertes Verfahren zu Grunde, welches durch Herausfiltern von bereits bekannten (Nicht-)QIs beschleunigt werden soll. Dazu wird dem Algorithmus die Menge der Attributkombinationen übergeben, welche bereits als (Nicht-)QIs bekannt sind, sodass diese bei der Berechnung einfach übersprungen werden können. Diese Menge wird im Folgenden als Ignore-Set bezeichnet.

3.2 Berechnung des Ignore-Sets

Auf die Attributkombinationen des Ignore-Sets lassen sich durch verschiedene Ansätze schließen. Im Folgenden werden diese Ansätze vorgestellt und gezeigt, welche Rückschlüsse sich daraus auf die QIs der Verbundrelation ziehen lassen und warum diese Schlüsse gelten.

3.2.1 Eigenschaften des Verbundattributes

Fremdschlüsselbeziehungen

Ist ein Attribut A Fremdschlüssel in Relation R_1 und referenziert in Tabelle R_2 das Attribut B, so impliziert ein Tupel aus R_1 mit $A = x$ die Existenz eines Tupels in R_2 mit $B = x$. Daraus folgt, dass bei einem Verbund von R_1 mit R_2 über den Fremdschlüssel A jedes Tupel aus R_1 mindestens einen Partner in R_2 findet. Deshalb sind mindestens alle Tupel aus R_1 in der Verbundrelation enthalten. Somit gilt mindestens für R_1, dass die Nicht-QIs und QIs der Ausgangsrelation auf der Verbundrelation gelten. Dies gilt auch für allgemeine Fremdschlüsselbeziehungen, die sich auf eine Menge von Attributen beziehen.

Verteilung der Attributwerte

Eine Schwierigkeit bei der Bestimmung von Quasi-Identifikatoren nach Verbundoperationen ist das Herausfallen von Tupeln. Fallen für einen QI während des Verbundes alle Tupel einer Äquivalenzklasse heraus, so kann die Anzahl der Tupel, welche durch den QI identifiziert werden, relativ zur Gesamtzahl der Tupel unter den Grenzwert für QIs fallen. So können Attributkombinationen durch das Herausfallen von Tupeln ihre QI-Eigenschaft verlieren. Analog können auch Nicht-QIs durch Herausfallen von Äquivalenzklassen zu QIs werden. Betrachten wir hierzu noch einmal das Studentenbeispiel mit den folgenden Relationen:

Beispiel: Studentenbeispiel

MatrNr.	Vorname	Nachname	Studiengang	Semester
1	Volker	Hagelstein	INF	3
2	Edmund	Hertel	WIN	3
3	Arthur	Ross	Lehramt für INF	5
4	Jakob	Schnell	ITTI	3
5	Natali	Geier	ET	5
6	Monika	Essig	BWL	3
7	Sven	Ferber	MA	5
8	Martin	Schüler	MA	7
9	Karin	Lang	CH	7
10	Erik	Kurz	MA	7

Tabelle 3.1: Student

Fach	Studiengang
1	INF
2	WIN
3	Lehramt für INF
4	ITTI
5	ET
6	MA
7	CH

Tabelle 3.2: Fach

Dabei identifiziert *Studiengang* in der Relation *Student* 70%, in *Fach* sogar 100%, der Tupel. Werden diese Relationen über den Studiengang Verbunden fällt die gesamte Äquivalenzklasse $Studiengang = BWL$ aus der Verbundrelation.

MatrNr.	Vorname	Nachname	Studiengang	Semester	Fach
1	Volker	Hagelstein	INF	3	1
2	Edmund	Hertel	WIN	3	2
3	Arthur	Ross	Lehramt für INF	5	3
4	Jakob	Schnell	ITTI	3	4
5	Natali	Geier	ET	5	5
7	Sven	Ferber	MA	5	6
8	Martin	Schüler	MA	7	6
9	Karin	Lang	CH	7	7
10	Erik	Kurz	MA	7	6

Tabelle 3.3: Student ⋈ Fach

Dadurch identifiziert *Studiengang* nun weniger Tupel ($6/9 = 66,66\%$). Wurde die Grenze für Quasi-Identifikatoren hierbei auf 70% vereinbart, fällt *Studiengang* nach dem Verbund aus der Menge der QIs heraus.

Ob Äquivalenzklassen herausfallen, lässt sich durch Betrachtung der Attributwerte berechnen:

Quellcodeausschnitt 3.1: Algorithmus - Zählen der Äquivalenzklassen

```
SELECT  base1_t.c-view_t.c as diff1, base2_t.c-view_t.c as diff2
FROM    (SELECT count(DISTINCT attribute)as c FROM base1) base1_t,
        (SELECT count(DISTINCT attribute)as c FROM base2) base2_t,
        (SELECT count(DISTINCT attribute)as c FROM view_table) view_t
```

Dabei sind *base1* und *base2* die Ausgangsrelationen des Verbundes, welcher die Sicht *view_table* bildet. Ist das Ergebnis ungleich 0, so hat sich die Anzahl der Äquivalenzklassen verändert. Dies lässt sich aus obiger Anfrage direkt für beide Basisrelationen entnehmen. Soll dies allerdings für eine Kombination mehrerer Attribute bestimmt werden, ist dies problematisch. Anfragen der Form

```
SELECT count(DISTINCT attribute1, attribute2,...)FROM t
```

sind nicht standardkonform, auch wenn sie von einigen Systemen unterstützt werden [MSD19]. Eine Möglichkeit wäre es jedoch, die Werte der Attribute zusammenzufügen, sodass nur auf einen zusammengesetzten Wert geprüft wird. Dies könnte aber Einbußen in der automatischen Optimierung des Datenbanksystem bedeuten. Eine Anfrage könnte dann, unter Ausnutzung der Konkatenation von Zeichenketten ("||" ist der binäre Operator für die Konkatenation), wie folgt aussehen:

```
SELECT count(DISTINCT attribute1||'#'||attribute2||'#'||...)FROM t
```

Das # steht exemplarisch für ein beliebiges Sonderzeichen, welches in keiner Spalte der Tabelle vorkommen darf, ohne vorher escapt zu werden. Ein solches Sonderzeichen zwischen den einzelnen Attributen sorgt hier dafür, dass bestimmte Kombinationen von Attributwerten nicht fehlinterpretiert werden. Sonst könnten, falls Groß- und Kleinschreibung nicht beachtet wird, folgende Einträge der Studentendatenbank als gleich erkannt werden (in Bezug auf die Attributkombination {Vorname, Nachname}):

MatrNr.	Vorname	Nachname	Studiengang	Semester
144	Johanna	Aron	INF	3
169	Johann	Aaron	WIN	5

Für Datenbanksysteme, die den ROW-Operator besitzen, wie z..B. PostgreSQL, gibt es eine weitere Variante diese Anfrage zu schreiben, ohne die Stringkonkatenation zu verwenden. Mittels des Tupelkonstruktors lassen sich ebenfalls alle als Parameter angegebenen Attribute als logische Einheit auswerten. Die Anfrage würde dann wie folgt aussehen:

```
SELECT count(DISTINCT ROW(attribute1,attribute2,...)) FROM t
```

Allgemein betrachtet ist der obige Ansatz aber nicht sehr effizient, da für jede Unteranfrage eine Duplikateliminierung benötigt wird. Das bedeutet, dass die Relation vorher intern sortiert oder gehasht wird. Zudem werden drei Unteranfragen ausgeführt, die einen Tablescan für die gesamte jeweilige Relation bedeuten.
Ob Äquivalenzklassen herausfallen, lässt sich aber auch durch Betrachtung des Verbundattributes alleine abschätzen. Findet sich jeder Attributwert für das Verbundattribut in beiden Tabellen wieder, können keine Tupel, und damit auch keine Äquivalenzklassen, herausfallen. Dieser Ansatz lässt sich in SQL wie folgt umsetzen (nur für eine Richtung):

```
SELECT attribute
FROM base_1
WHERE (a1, a2, ...) NOT IN (
    SELECT a1, a2, ...
    FROM base_2)
```

Ist das Ergebnis der Anfrage leer, so fallen für *base_1* keine Äquivalenzklassen heraus. Dies ist im Gegensatz zu obigen Ansatz (3.1) effizienter, da diese Anfrage nicht für jede Attributkombination ausgeführt werden muss, hier keine Duplikateliminierung nötig ist und die Verbundrelation noch nicht benötigt wird. Somit könnte diese Anfrage parallel zur Berechnung des Verbundes ausgeführt werden. Der Nachteil dieses Ansatzes ist aber, dass unbekannt bleibt, für welche Attributkombinationen Äquivalenzklassen herausfallen. Das bedeutet, dass es nicht möglich ist, einzelne Attributkombinationen, für die keine Äquivalenzklassen herausfallen, gesondert zu betrachten.

3.2.2 Verbundkardinalität

Die Kardinalität des Verbundes, gemeint ist wie oft jedes Tupel der Relation am Verbund teilnehmen kann, ist ausschlaggebend dafür, ob die Quasi-Identifikatoren der Basisrelationen auch auf der Verbundrelation gelten. Relevant ist, ob für diese Attributkombination neue Tupel generiert werden. Werden neue zusätzliche Tupel mit gleichen Werten für die QIs generiert, so identifizieren sie weniger Tupel, sodass die QI-Eigenschaft für diese Attributkombination verloren gehen kann. Folgende Aussagen gelten für diejenigen Attributkombinationen, bei welchen durch den Verbund keine Äquivalenzklassen entfallen:

1-zu-1-Verbund

Hier findet jedes Tupel der einen Tabelle maximal einen Partner mit dem gleichen Wert für das Verbundattribut in der anderen Tabelle und umgekehrt. Das bedeutet, dass keine neuen Tupel für die Attributkombinationen einer Tabelle generiert werden. Daher gelten die Quasi-Identifikatoren der Ausgangsrelationen auch auf der Verbundrelation und können dem Ignore-Set hinzugefügt werden.

1-zu-n-Verbund

Bei einem 1-zu-n-Verbund findet sich jedes Tupel der 1-Seite maximal einmal in der Verbundrelation. Das bedeutet, für diese Relation bleiben, wie beim 1-zu-1-Verbund, die QIs der Ausgangsrelation gültig. Ein Tupel der n-Seite kann dagegen beliebig oft in der Verbundrelation auftauchen. Daher können für die zusätzliche Tupel mit gleichen Werten für die QIs entstehen. Deshalb können die Quasi-Identifikatoren dieser Ausgangsrelation nicht ohne Weiteres als gültig auf der Verbundrelation angenommen werden.

n-zu-m-Verbund

Da hier ein Tupel aus jeder Seite beliebig oft in der Verbundrelation auftauchen kann, werden potenziell Tupel mit gleichen Werten für die QIs gebildet. Bei einem n-zu-m-Verbund können daher keine der Quasi-Identifikatoren aus den Basisrelationen als geltend angenommen werden.

3.3 Berechnung während des Verbundes

Die Ermittlung der Quasi-Identifikatoren einer Verbundrelation während des Verbundes ist eigentlich nicht Gegenstand dieser Arbeit. Allerdings lassen sich durch die Berechnung während des Verbundes die Attributwerte parallel zur Verbundberechnung betrachten, sodass einige Aussagen über die QIs der Verbundrelation getroffen werden können, die bei einer reinen Betrachtung der Ausgangsrelationen und der Verbundrelation nicht möglich wären. Daher werden im Nachfolgenden nur einige Ideen beschrieben, wie die Berechnung während des Verbundes ausgenutzt werden könnte.

3.3.1 Neuberechnung der Quasi-Identifikatoren

Der einfachste Ansatz zur Berechnung von Quasi-Identifikatoren während des Verbundes ist es, die QI-Eigenschaft für alle Attributkombinationen zu überwachen. Da für jegliche Berechnung während des Verbundes eine Tabellenfunktion implementiert werden muss, kann diese auch gleich während des Verbundes mitzählen, welche Attributkombination wie viele Tupel in der entstehenden Tabelle identifiziert. Dafür könnte man zunächst eine temporäre Tabelle generieren, welche als Map fungiert (gemeint ist der Typ Map, welcher aus Programmiersprachen wie Java bekannt ist). Diese sollte die Attributkombination, den Wert für diese Kombination und die Anzahl der Tupel mit diesem Wert enthalten. Dann wird für jedes Verbundpaar und jede Teilmenge der Attributmenge der entsprechende Wert der Attributkombination dort eingetragen.
Damit erspart man es sich, die Verbundrelation erst aufzubauen und dann Anfragen an sie zu stellen. Andererseits müssen dann für jedes Verbundpaar und jede Teilmenge der Attributmenge mit dynamischem SQL Werte in diese temporäre Tabelle eingetragen werden.

3.3.2 Berechnung mittels Verbundkardinalität

Eine weitere Möglichkeit wäre es, die oben erläuterten Ansätze aus Abschnitt 3.2 mit der Möglichkeit den Verbund zu überwachen, zu verbinden. Statt die QIs neu zu berechnen, könnte man während des Verbundes bereits überprüfen, um welche Form eines Verbundes es sich handelt und ob Tupel herausfallen. Weiter lässt sich sogar genau bestimmen, welche Tupel, und damit auch welche Äquivalenzklassen, die Gültigkeit der (Nicht-)QIs der Ausgangsrelationen beeinflussen. Da durch die Überwachung des Verbundes bekannt ist, wie viele Tupel herausfallen und welche Äquivalenzklassen dadurch entfallen, können dadurch betroffene Attributkombinationen gesondert betrachtet werden.

Kapitel 4

Implementierung

In diesem Kapitel werden alle Implementierungen vorgestellt, die für die Umsetzung des Konzeptes benötigt werden. Dazu werden Programmausschnitte erklärt und auf mögliche Schwierigkeiten bei der Implementation eingegangen.

4.1 Allgemeines zur Implementierung

Um den Algorithmus möglichst effizient zu halten, wurde dieser direkt als Stored Procedure in SQL mit Programmiersprachenerweiterung implementiert. Dadurch wird der Kommunikationsaufwand reduziert. Dieser würde anfallen, falls der Algorithmus in einer Programmiersprache implementiert wird und das Programm durch einen entsprechenden Datenbanktreiber, wie beispielsweise JDBC, mit der Datenbank kommuniziert. Zusätzlich kann so direkt auch datenbankinterne Statistiken und Funktionen zugegriffen werden. Der nachfolgende Quellcode wurde für PostgreSQL (ab Version 9.4) entwickelt.

4.2 Berechnung von Quasi-Identifikatoren einer Relation

Da das vorgestellte Konzept auf einem Bottom-Up Algorithmus zur Berechnung von Quasi-Identifikatoren basiert, wird zuerst diese Implementation benötigt. PostgreSQL untersützt Arrays als Datentyp, welche für die Implementierung des Bottom-Up-Ansatzes genutzt werden. Für Datenbankmanagementsysteme, die noch keine Arrays unterstützen, können temporäre Tabellen oder Cursors verwendet werden. Eingabevariablen für die Funktion sind der Tabellenname, *tname*, sowie der Grenzwert für Quasi-Identifikatoren, *threshold*. Weiter werden folgende Variablen benötigt:

Name	**Typ**	**Beschreibung**
outArr	varchar[][]	Liste der erkannten QIs und Rückgabewert
attributeList	varchar[]	Liste der Attribute der Tabelle
lowerSet	varchar[][]	Menge der aktuell zu überprüfenden Attributkombinationen
copySet	varchar[][]	Wird gebraucht, um auf einer Kopie des lowerSet zu arbeiten
tCount	double precision	Anzahl der Tupel in der Relation
eCount	double precision	Anzahl der Äquivalenzklasseneiner Attributkombination
maxQISize	integer	Maximallänge eines QIs
containsQI	boolean	wird gebraucht, um auf Enthaltensein von QIs zu prüfen
containedByLS	boolean	wird für die Erweiterung des lowerSets benötigt
iterSet	varchar[]	wird gebraucht, um Werte zusammenzusetzen
attr	varchar	Laufvariable
subset, lSet, qi	varchar[]	Laufvariablen

Der Bottom-Up-Algorithmus arbeitet in drei Schritten. Der erste Schritt ist dabei eine Initialisierungsphase. Aus dem *information_schema* werden die Attribute der Tabelle ausgelesen, die Variablen initialisiert und das erste *lowerSet* erstellt. Statt *tCount* selbst zu berechnen, können auch datenbankinterne Statistiken genutzt werden. Die Abfrage dieser ist besonders für große Tabellen viel schneller als die Berechnung aus dem folgenden Quellcodeausschnitt. Allerdings ist diese Statistik ungenau. Hier muss abgewogen werden, ob die Verbesserung der Laufzeit Ungenauigkeiten rechtfertigt.

```
1  EXECUTE 'SELECT ARRAY(
2              SELECT column_name
3              FROM information_schema.columns
4              WHERE table_name = $1)'
5  INTO attributeList USING tname;
6  maxQISize := array_length(attributeList,1);
7  FOREACH attr in ARRAY attributeList LOOP
8    lowerSet := lowerSet||ARRAY[[attr]];
9  END LOOP;
10 EXECUTE 'SELECT COUNT(*) FROM '||tname INTO tCount;
```

Quellcodeausschnitt 4.1: Initialisierungsphase

Der zweite und dritte Schritt werden hintereinander in einer Schleife ausgeführt, solange das *lowerSet* nicht leer ist. Im zweiten Schritt werden die Elemente des *lowerSet*s darauf überprüft, ob sie der QI-Eigenschaft entsprechend des verwendeten Grenzwertes genügen. Dabei wird für jede Attributkombination die Anzahl der verschiedenen Werte berechnet. Genügt eine Attributkombination der QI-Eigenschaft, wird sie bis zu einer Länge von *maxQISize* mit Nullwerten aufgefüllt und dann *outArr* hinzugefügt. Das Padding mit Nullwerten ist nötig, da in mehrdimensionalen Arrays, wie der Menge von QIs, Arrays einer Dimension die gleiche Länge besitzen müssen. Aufgrund der Umsetzung von Arrays in PostgreSQL sind noch einige weitere Workarounds notwendig:

Beispiel: Workarounds für Probleme mit Arrays in PostgreSQL

- In mehrdimensionalen Arrays müssen die Elemente (ebenfalls Arrays) einer Dimension die gleiche Länge besitzen. Eine Möglichkeit allgemeine Mengen von Mengen darzustellen ist beispielsweise *'{{a},{c,d}}'* zu *'{{a,NULL},{c,d}}'* aufzufüllen.
- Die binären Operatoren @> und <@ (A @> B bedeutet "A enthält B") führt vor der Auswertung eine Dimensionsglättung aus. Das heißt, *'{{1,2},{3,4}}'@>'{1,4}'* = *True*.
  Deshalb muss über den mehrdimensionalen Array iteriert und elementweise auf Gleichheit geprüft werden.
- Arrays sind keine Mengen. *'{1,2}'='{2,1}'* muss deshalb durch *'{1,2}'@> '{2,1}'∧'{1,2}'<@'{2,1}'* abgebildet werden.

```
1  copySet := lowerSet;
2  FOREACH subset SLICE 1 in ARRAY copySet LOOP
3    EXECUTE 'SELECT COUNT(*) FROM(SELECT DISTINCT "'||
     array_to_string(subset,'","')||'"FROM '||tname||') e' INTO eCount;
4    IF (eCount/tCount)>=threshold THEN
5      outArr := outArr || array[(subset||array_fill(NULL::varchar,
       ARRAY[maxQISize-array_length(subset,1)]))];
6    END IF;
7  END LOOP;
```

Quellcodeausschnitt 4.2: Überprüfung des lowerSets

Im letzten Schritt wird das *lowerSet* für die nächste Iteration erweitert. Dazu wird jede Attributkombination mit jedem Attribut erweitert. Alle neuen, um eins größeren Attributkombinationen, welche keinen Quasi-Identifikator enthalten werden zum *lowerSet* der nächsten Iteration hinzugefügt.

```
1  lowerSet := ARRAY[]::varchar[][];
2  FOREACH subset SLICE 1 in ARRAY copySet LOOP
3    FOREACH attr in ARRAY attributeList LOOP
4      containedByLS := False;
5      IF NOT(subset @> ARRAY[attr]) THEN
6        iterSet := subset || attr;
7        IF NOT(cardinality(lowerSet) = 0 )THEN
8          FOREACH lSet SLICE 1 in ARRAY lowerSet LOOP
9            IF(lSet @> ARRAY[iterSet] AND lSet <@ ARRAY[iterSet]) THEN
10             containedByLS := true;
11           END IF;
12         END LOOP;
13       END IF;
14       IF NOT(containedByLS) THEN
15         IF cardinality(outArr) = 0 THEN
16           lowerSet:= lowerSet || array[iterSet];
17         ELSE
18           containsQI := False;
19           FOREACH qi SLICE 1 in ARRAY outArr LOOP
20             IF (iterSet @> array_remove(qi,NULL)) THEN
21               containsQI := true;
22               EXIT;
23             END IF;
24           END LOOP;
25           IF NOT(containsQI) THEN
26             lowerSet := lowerSet || array[iterSet];
27           END IF;
28         END IF;
29         iterSet := ARRAY[]::varchar[];
30       END IF;
31     END IF;
32   END LOOP;
33 END LOOP;
34 copySet := ARRAY[]::varchar[][];
```

Quellcodeausschnitt 4.3: Erweiterung des lowerSets

4.3 Berechnung von Quasi-Identifikatoren einer Verbundrelation

Im nachfolgenden Abschnitt werden die Implementationen für den im Konzept (siehe Kapitel 3) beschriebenen Ansatz vorgestellt. Da es, wie im Konzept vorgestellt, drei Fälle zu unterscheiden gibt, werden diese durch jeweils drei Stored Procedures implementiert. Hier werden alle Variablen des Algorithmus zur Berechnung der QIs einer Relation benötigt (siehe Abschnitt 4.2). Weiter sind das Ignore-Set, ein Array von Arrays des Typs varchar, und die boolsche Variable *skipCount* erforderlich. Da sich am Schritt, in welchem das *lowerSet* erweitert wird, nichts ändert, wird dieser im Nachfolgenden ausgelassen.

4.3.1 n-zu-m-Verbund

Die folgende Stored Procedure wird im allgemeinsten Fall, dem n-zu-m-Verbund, aufgerufen. Als Eingabeparameter werden die Namen beider Ausgangsrelationen sowie der Name des Verbundattributes in den jeweiligen Tabellen und der Grenzwert für QIs benötigt.

```
qi_detect_NtoM (tname1 varchar, tname2 varchar,
        joinAttr1 varchar, joinAttr2 varchar,
        threshold double precision)
returns varchar[][]
```

Quellcodeausschnitt 4.4: Signatur der Stored Procedure (n-zu-m-Verbund)

Für den Fall, dass Attribute mit gleichen Namen in beiden Relationen existieren, werden die Tabellennamen den entsprechenden Attributen angefügt. Dadurch wird allerdings das Verbundattribut doppelt berechnet, sofern man es nicht explizit aus dem *lowerSet* entfernt. Alle vorher nur auf Attribute eine Tabelle beschränkten Anfragen, müssen jetzt auf den Verbund der Tabellen über das angegebene Verbundattribut angewendet werden. Da dieser Verbund sonst immer wieder berechnet werden müsste, bietet sich hier die Nutzung einer temporären Tabelle an. Das bedeutet allerdings einen kleinen Mehraufwand, da für die Anfrage die Namen der Ausgangsrelationen durch den der temporären Tabelle ersetzt werden müssen (siehe Quellcodeausschnitt 4.5, Zeile 9).

```
1  EXECUTE 'SELECT ARRAY(
2      SELECT table_name||''.''||column_name
3      FROM Information_schema.columns
4      WHERE table_name = $1
5      OR table_name = $2)'
6  INTO attributeList USING tname1, tname2;
7  attributeList := array_remove(attributeList,
   (tname2||'.'||joinAttr2)::varchar);
8  tempTableName := 'qi';
9  EXECUTE 'CREATE TEMP TABLE '||tempTableName||' ON COMMIT DROP AS
10     SELECT '|| array_to_string(attributeList,',')||'
11     FROM '||tname1||','||tname2||'
12     WHERE '|| tname1||'.'||joinAttr1||'='||tname2||'.'||joinAttr2
13 USING tempTableName;
14 attributeList := string_to_array(replace(replace(
15                  array_to_string(attributeList,','),tname1||'.',
16                  tempTableName||'.'),
17                  tname2||'.',tempTableName||'.'),',');
18 maxQISize := array_length(attributeList,1);
19 FOREACH attr in ARRAY attributeList LOOP
20   lowerSet := lowerSet||ARRAY[[attr]];
21 END LOOP;
22 EXECUTE 'SELECT COUNT(*) FROM '||tempTableName INTO tCount;
```

Quellcodeausschnitt 4.5: Initialisierungsphase (n-zu-m-Verbund)

Da in diesem Fall keine der bereits bekannten QIs und Nicht-QIs als gültig auf der Verbundrelation angenommen werden können, ähnelt der 2. Schritt, in welchem das *lowerSet* überprüft wird, stark dem des ursprünglichen Algorithmus für eine Relation:

```
1  copySet := lowerSet;
2  FOREACH subset SLICE 1 in ARRAY copySet LOOP
3    EXECUTE 'SELECT COUNT(*) FROM(
4               SELECT DISTINCT ' ||array_to_string(subset,',')||'
5               FROM '||tempTableName||') e'
6    INTO eCount;
7    IF (eCount/tCount)>=threshold THEN
8      outArr := outArr || array[(subset||array_fill(NULL::varchar,
       ARRAY[maxQISize-array_length(subset,1)]))];
9    END IF;
10 END LOOP;
```

Quellcodeausschnitt 4.6: Überprüfung des lowerSets (n-zu-m-Verbund)

Bei einem n-zu-m-Verbund kann keinerlei zusätzliches Wissen um die Eigenschaften der Ausgangsrelationen genutzt werden. Entsprechend ist der Algorithmus so implementiert, dass man ihn auf beliebige Tabellen Anwenden kann, auch wenn sie in Wirklichkeit in einer 1-zu-1- oder 1-zu-n-Beziehung stehen. Deshalb wird diese Stored Procedure auch für den Fall genutzt, dass gar nichts über die Ausgangsrelationen bekannt ist.
Diese Implementation, sowie die für die anderen Verbundkardinalitäten, sind für natürliche Verbunde (Equi-Joins) gedacht. Für andere Verbundarten, wie z. B. Outer-Joins, müssten allerdings nur die SQL-Statements angepasst werden, welche die temporäre Tabelle erstellen. Weiter kann bei Outer-Joins noch die Eigenschaft ausgenutzt werden, dass bei diesen Verbunden keine Tupel herausfallen können.

4.3.2 1-zu-n-Verbund

Diese Stored Procedure ist für den Fall eines 1-zu-n-Verbundes nutzbar, sofern bei diesem keine Äquivalenzklassen für Attributkombinationen herausfallen. Sie überprüft nicht selbst, ob wirklich ein 1-zu-n-Verbund vorliegt, sondern führt lediglich die Berechnungen der Quasi-Identifikatoren durch. Dabei werden die im Konzept unter 3.2.2 vorgestellten Ansätze berücksichtigt. Die Möglichkeit Attributkombinationen gesondert zu betrachten, für die Äquivalenzklassen herausfallen, wurde hier bewusst nicht implementiert. Grund dafür ist, dass die Erkennung eben dieser Attributkombinationen ungefähr so komplex ist, wie erneut zu bestimmen, ob es sich bei dieser Attributkombination um einen QI handelt:
Betrachten wir hierzu zuerst die Anfrage in Quellcodeausschnitt 3.1. Sie ist nicht konform mit dem SQL-Standard, aber da äquivalente Umsetzungen in SQL noch weitere Unteranfragen oder zusätzliche Berechnungen benötigen, welche sich negativ auf die Laufzeit auswirken, soll sie uns dennoch als Ausgangslage dienen. In dieser Anfrage werden jeweils die unterschiedlichen Werte für diese Attributkombination auf beiden Ausgangsrelationen und auf der Verbundrelation gezählt und anschließend miteinander verrechnet. Die letzte Unterabfrage, welche dies für die Verbundrelation tut, ist aber bereits der aufwendigste Teil der Berechnung dafür, ob diese Attributkombination ein QI ist. Lediglich das Teilen durch die Gesamtzahl der Tupel in der Verbundrelation ist noch nötig. Dafür wird ein Tablescan ($O(n)$) durchgeführt. Dafür fallen bei der Anfrage in Quellcodeausschnitt 3.1 noch zwei Unteranfragen ins Gewicht, welche ebenfalls Duplikateliminierung (wegen Sortierung bzw. Hashen in $O(nlog(n))$ bzw. $O(n)$) benötigen.

```
qi_detect_1toN (tname1 varchar, tname2 varchar,
        joinAttr1 varchar, joinAttr2 varchar,
        qi1 varchar[][], non_qi1 varchar[][],
        threshold double precision)
returns varchar[][]
```

Quellcodeausschnitt 4.7: Signatur der Stored Procedure (1-zu-n-Verbund)

Da bei einem 1-zu-n-Verbund ohne Herausfallen von Äquivalenzklassen die QIs der 1-Seite gültig auf der Verbundrelation sind, können die QIs direkt der Menge der QIs, dem *outArr*, hinzugefügt werden, sobald sie für die neue *maxQISize* normalisiert wurden. Abgesehen davon funktioniert die Initialisierung analog zur Initialisierungsphase des n-zu-m-Verbundes.

```
1  EXECUTE 'SELECT ARRAY(
2        SELECT table_name||''.''||column_name
3        FROM Information_schema.columns
4        WHERE table_name = $1
5        OR table_name = $2)'
6  INTO attributeList USING tname1, tname2;
7  attributeList := array_remove(attributeList,
   (tname2||'.'||joinAttr2)::varchar);
8  tempTableName := 'qi';
9  EXECUTE 'CREATE TEMP TABLE '||tempTableName||' ON COMMIT DROP AS
10     SELECT '|| array_to_string(attributeList,',')||'
11     FROM '||tname1||','||tname2||'
12     WHERE '|| tname1||'.'||joinAttr1||'='||tname2||'.'||joinAttr2
13   USING tempTableName;
14 attributeList := string_to_array(replace(replace(
15                 array_to_string(attributeList,','), tname1||'.',
16                 tempTableName||'.'),
17                 tname2||'.', tempTableName||'.'), ',');
18 maxQISize := array_length(attributeList,1);
19 IF NOT(cardinality(qi1)=0) THEN
20   FOREACH subset SLICE 1 IN ARRAY qi1 LOOP
21     outArr := outArr||ARRAY[(subset||array_fill(NULL::varchar,
       ARRAY[maxQISize-array_length(subset,1)]))];
22   END LOOP;
23 END IF;
24 ignoreSet := outArr;
25 IF NOT(cardinality(non_qi1)=0) THEN
26   FOREACH subset SLICE 1 IN ARRAY non_qi1 LOOP
27     ignoreSet = ignoreSet || ARRAY[(subset||array_fill(NULL::varchar,
       ARRAY[maxQISize-array_length(subset,1)]))];
28   END LOOP;
29 END IF;
30 FOREACH attr in ARRAY attributeList LOOP
31   lowerSet := lowerSet||ARRAY[[attr]];
32 END LOOP;
33 EXECUTE 'SELECT COUNT(*) FROM '||tempTableName INTO tCount;
```

Quellcodeausschnitt 4.8: Initialisierungsphase (1-zu-n-Verbund)

Während der Berechnung können hier Attributkombinationen, die bereits als Nicht-QI auf der Verbundrelation bekannt sind, einfach übersprungen werden. Der Algorithmus ist insofern flexibel, dass auch bei einem Verbund, bei welchem Tupel, aber keine Äquivalenzklassen herausfallen, diese Stored Procedure genutzt werden kann. Da dabei aber Nicht-QIs zu QIs werden könnten, weil sich die Gesamtzahl der Tupel verringern kann, muss *non_ qi* als leerer Array mit dem richtigen Typen übergeben werden.
Die Abfrage darauf, ob *cardinality(ignoreSet)=0* ist, ist ein weiteres Workaround (siehe Quellcodeausschnitt 4.9, Zeile 4). Die Dimensionalität eines Arrays wird, anders als der Typ, nicht bei der Initialisierung festgelegt. *Cardinality(anyarray)* gibt die Gesamtzahl der Elemente zurück und ist nötig, um zu verhindern, dass versucht wird über einen Array mit 0 Dimensionen (leerer Array) zu iterieren, was sonst zu Fehlern führen würde.

```
1  copySet := lowerSet;
2  FOREACH subset SLICE 1 IN ARRAY copySet LOOP
3    skipCount := FALSE;
4    IF(NOT(cardinality(ignoreSet)=0)) THEN
5      FOREACH subset2 SLICE 1 IN ARRAY ignoreSet LOOP
6        IF(subset @> array_remove(subset2,NULL) AND subset <@
         array_remove(subset2,NULL)) THEN
7          skipCount := TRUE;
8          EXIT;
9        END IF;
10     END LOOP;
11   END IF;
12   IF(NOT(skipCount)) THEN
13     EXECUTE 'SELECT COUNT(*) FROM(
14           SELECT DISTINCT '||array_to_string(subset,',')||
15           ' FROM '||tempTableName||' ) e'
16       INTO eCount;
17     IF (eCount/tCount)>=threshold THEN
18       outArr := outArr || ARRAY[(subset||array_fill(NULL::varchar,
         ARRAY[maxQISize-array_length(subset,1)]))];
19     END IF;
20   END IF;
21 END LOOP;
```

Quellcodeausschnitt 4.9: Überprüfung des lowerSets (1-zu-n-Verbund)

4.3.3 1-zu-1-Verbund

Beim 1-zu-1-Verbund ohne herausfallende Äquivalenzklassen sind die QIs beider Relationen auf der Verbundrelation gültig. Sollte bekannt sein, dass keine Tupel einer oder sogar beider Seiten herausfallen, lassen sich auch die Nicht-QIs der entsprechenden Tabelle verwenden. Folglich muss die Signatur für die Stored Procedure wie folgt aussehen:

```
qi_detect_1to1 (tname1 varchar, tname2 varchar,
        joinAttr1 varchar, joinAttr2 varchar,
        qi1 varchar[][], qi2 varchar[][],
        non_qi1 varchar[][], non_qi2 varchar[][],
        threshold double precision)
returns varchar[][]
```

Quellcodeausschnitt 4.10: Signatur der Stored Procedure (1-zu-1-Verbund)

Hier gelten die QIs beider Ausgangsrelationen, welche deshalb gleich der Menge der QIs hinzugefügt werden. Auch dieser Algorithmus ist flexibel in der Nutzung der Nicht-QIs. Nehmen Tupel einer Ausgangsrelation nicht am Verbund teil, können deren Nicht-QIs einfach ignoriert werden (nicht als Parameter übergeben werden). Auch hier müssen die bereits bekannten QIs und Nicht-QIs für die neue *maxQISize* normalisiert werden. Dies geschieht wie folgt:

```
1  IF NOT(cardinality(qi1)=0) THEN
2    FOREACH subset SLICE 1 IN ARRAY qi1 LOOP
3      normalized_qi :=
       normalized_qi||ARRAY[(subset||array_fill(NULL::varchar,
       ARRAY[maxQISize-array_length(subset,1)]))];
4    END LOOP;
5  END IF;
6  IF NOT(cardinality(qi2)=0) THEN
7    FOREACH subset SLICE 1 IN ARRAY qi2 LOOP
8      normalized_qi :=
       normalized_qi||ARRAY[(subset||array_fill(NULL::varchar,
       ARRAY[maxQISize-array_length(subset,1)]))];
9    END LOOP;
10 END IF;
11 outArr := normalized_qi;
12 ignoreSet := outArr;
13 IF NOT(cardinality(non_qi1)=0) THEN
14   FOREACH subset SLICE 1 IN ARRAY non_qi1 LOOP
15     normalized_non_qi = normalized_non_qi ||
       ARRAY[(subset||array_fill(NULL::varchar,
       ARRAY[maxQISize-array_length(subset,1)]))];
16   END LOOP;
17 END IF;
18 IF NOT(cardinality(non_qi2)=0) THEN
19   FOREACH subset SLICE 1 IN ARRAY non_qi2 LOOP
20     normalized_non_qi = normalized_non_qi ||
       ARRAY[(subset||array_fill(NULL::varchar,
       ARRAY[maxQISize-array_length(subset,1)]))];
21   END LOOP;
22 END IF;
23 ignoreSet := ignoreSet||normalized_non_qi;
```

Quellcodeausschnitt 4.11: Initialisierungsphase (1-zu-1-Verbund)

Der letzte Schritt, die Berechnung der QIs im aktuellen *lowerSet*, ist identisch zu der des 1-zu-n-Verbundes (siehe Quellcodeausschnitt 4.9). Der Grund dafür ist, dass die Stored Procedures aufeinander aufbauende Spezialisierungen sind.

$$\text{n-zu-m-Verbund} \succ \text{1-zu-n-Verbund} \succ \text{1-zu-1-Verbund}$$

Daher unterscheiden sich der 1-zu-1- und der 1-zu-n-Fall nur in ihrer Signatur und der Initialisierungsphase. Genauer liegt der Unterschied lediglich darin, welche QIs und Nicht-QIs als gültig auf der Verbundrelation betrachtet werden.

4.3.4 Auswahl der Stored Procedure

Für die Auswahl der Stored Procedure gibt es eine Reihe vorläufiger Anfragen, die genutzt werden können, um zu bestimmen, welche Verbundkardinalität vorliegt und ob Äquivalenzklassen bzw. Tupel herausfallen. Die Wichtigsten davon wurden bereits im Konzept vorgestellt. Hier wird nur eine Auswahl von Möglichkeiten präsentiert, welche als effizient genug beachtet wurden, sodass tatsächlich ein Gewinn an Performance erkennbar wäre. Zu viele Anfragen zur Einteilung des aktuellen Verbundes in die verschiedenen möglichen Fälle kann allerdings auf zu einem Verlust an Performance führen. Dies gilt insbesondere dann, wenn die Anzahl an QIs und Nicht-QIs, welche auf der Verbundrelation gelten, sehr klein ist. In diesem Fall wäre die Ersparnis durch das Überspringen dieser wenigen Attributkombinationen bei der Überprüfung des *lowerSets* eher gering. Dadurch könnte die Betrachtung des Verbundes als n-zu-m-Verbund schneller sein, als diesen vorher durch andere Anfragen richtig einzuordnen. Die verschiedenen Fälle, die sich ergeben sind die Folgenden:

- 1-zu-1-Verbund
- 1-zu-1-Verbund ohne herausfallende Äquivalenzklassen (nur eine Ausgangsrelationen)
- 1-zu-1-Verbund ohne herausfallende Äquivalenzklassen und Tupel (nur eine Ausgangsrelationen)
- 1-zu-1-Verbund ohne herausfallende Äquivalenzklassen (beide Ausgangsrelationen)
- 1-zu-1-Verbund ohne herausfallende Äquivalenzklassen und Tupel (beide Ausgangsrelationen)
- 1-zu-n-Verbund
- 1-zu-n-Verbund ohne herausfallende Äquivalenzklassen
- 1-zu-n-Verbund ohne herausfallende Äquivalenzklassen und Tupel
- n-zu-m-Verbund

Wie im Konzept (siehe Unterabschnitt 3.2.1) gezeigt ist es bei Verbunden, bei welchen Äquivalenzklassen herausfallen können, nicht ohne Weiteres möglich, Aussagen über die Gültigkeit der QIs und Nicht-QIs der Ausgangsrelationen auf der Verbundrelation zu tätigen. Dies ginge nur, indem man während des Verbundes erfasst, ob Äquivalenzklassen herausfallen. Deshalb können, ungeachtet der Verbundkardinalität, alle Verbunde, bei denen Äquivalenzklassen herausfallen können, mit der Stored Procedure für den n-zu-m-Verbund behandelt werden. Um herauszufinden, ob Äquivalenzklassen herausfallen, werden in dieser Implementation folgende Ansätze verwendet:

```
SELECT ccu.table_name AS referenced_table_name
FROM  information_schema.table_constraints AS tc
  JOIN information_schema.key_column_usage AS kcu
    ON tc.constraint_name = kcu.constraint_name
    AND tc.table_schema = kcu.table_schema
  JOIN information_schema.constraint_column_usage AS ccu
    ON ccu.constraint_name = tc.constraint_name
    AND ccu.table_schema = tc.table_schema
WHERE tc.constraint_type = 'FOREIGN KEY'
    AND (tc.table_name=tname1 AND ccu.table_name=tname2
        AND kcu.column_name=joinAttr1 AND  ccu.column_name=joinAttr2
      OR tc.table_name=tname2 AND ccu.table_name=tname1
        AND kcu.column_name=joinAttr2 AND  ccu.column_name=joinAttr1)
```

Quellcodeausschnitt 4.12: Erkennung von Fremdschlüsselbeziehungen

Die meisten Verbunde laufen über Schlüssel und Fremdschlüssel. Deshalb sollten diese möglichst früh entdeckt werden. Ist das Verbundattribut einer Relation ein Fremdschlüssel, der das Verbundattribut der anderen Relation referenziert, so fallen für die referenzierende Seite keine Äquivalenzklassen und Tupel heraus. Diese Anfrage ist ausreichend, um zu bestimmen, ob eine Fremdschlüsselbeziehung zwischen den beiden Relationen vorliegt. Nur sollte bei mehreren vorliegenden Schemata darauf geachtet werden, dass auch das Schema der zu untersuchenden Tabellen benutzt wird. Es kann entweder kein Ergebnis ausgegeben werden oder einer der beiden Tabellennamen wird zurückgegeben. Im ersten Fall bedeutet das, dass es keine Fremdschlüsselbeziehung zwischen den beiden Tabellen gibt. Wird ein Tabellenname zurückgegeben, handelt es sich um den Namen der 1-Seite eines 1-zu-n- oder 1-zu-1-Verbundes. Das lässt sich dadurch begründen, dass jedes Tupel der referenzierenden Tabelle genau einmal am Verbund teilnimmt. Ein Tupel dieser Relation muss mindestens einmal am Verbund teilnehmen, da die Fremdschlüsselbeziehung die Existenz eines Tupels mit gleichem Wert für das referenzierte Attribut in der referenzierten Tabelle voraussetzt. Weiter kann es nicht mehr als einmal am Verbund teilnehmen, da ein Fremdschlüssel nur einen Primärschlüssel oder ein als *UNIQUE* markiertes Attribut referenzieren kann, und diese nicht mehr als einmal in ihrer Tabelle auftauchen dürfen. Somit ist die Verbundkardinalität bereits eingeschränkt. Außerdem bedeutet dies, dass das Herausfallen von Tupeln von einer Seite des Verbundes bereits unmöglich ist.
Selbst falls keine solche Fremdschlüsselbeziehung zwischen den Ausgangstabellen vorliegt, kann trotzdem noch effizient geprüft werden, ob keine Äquivalenzklassen aus dem Verbund herausfallen. Wie bereits im Konzept erklärt ist der nach Attributkombinationen aufgeschlüsselte Ansatz, um herausfallende Äquivalenzklassen zu entdecken, zu aufwendig, um ihn an dieser Stelle einzusetzen. Stattdessen wird der aus dem Konzept bekannte Ansatz verwendet, die Werte der Verbundattribute zu betrachten:

```
SELECT count(*)
FROM tname1
WHERE joinAttr1 NOT IN (
  SELECT joinAttr2
  FROM tname2)
```

Quellcodeausschnitt 4.13: Erkennung von herausfallenden Äquivalenzklassen

Diese Anfrage muss in beide Richtungen durchgeführt werden (tname1 und tname2 vertauschen), um zu wissen, für welche Relationen Tupel herausfallen. Die Ausnahme dabei bildet der Fall, bei welchem zuvor bereits eine Fremdschlüsselbeziehung erkannt wurde. Falls dem so ist, muss dies nur für die (potenzielle) n-Seite des Verbundes durchgeführt werden. Fallen für diese Seite nämlich Tupel heraus, so kann, ungeachtet davon, ob es sich um einen 1-zu-1- oder einen 1-zu-n-Verbund handelt, der 1-zu-n-Algorithmus zur Berechnung der QIs benutzt werden, da für diese Seite des Verbundes keine QIs oder Nicht-QIs als gültig angenommen werden können.
Ist das Ergebnis der Anfrage in beide Richtungen ungleich 0, so kann einfach die Stored Procedure für den n-zu-m-Fall benutzt werden. Für alle Fälle, bei denen die Auswahl der Stored Procedure bisher noch nicht geklärt ist, muss anschließend noch überprüft werden, um welche Art von Verbund es sich handelt:

```
1 FOR dist_val in EXECUTE 'SELECT DISTINCT '||joinAttr1||' from '||tname1
  LOOP
2   EXECUTE 'SELECT count(*) from '||tname2||' WHERE '||joinAttr2||'=$1'
    INTO cardinalityCounter USING dist_val
3   IF cardinalityCounter>1 THEN
4     tname1IsN := 'true';
5     EXIT;
6   END IF;
7 END LOOP;
```

Quellcodeausschnitt 4.14: Erkennung der Verbundkardinalität

Findet sich für ein Tupel aus der ersten Ausgangsrelation mehr als ein Verbundpartner in der anderen Tabelle, kann es sich nicht mehr um einen 1-zu-1-Verbund handeln. Dann sind nur noch ein 1-zu-n- oder ein n-zu-m-Verbund möglich. Existiert so ein Tupel allerdings nicht, kann es sich sowohl um einen 1-zu-1-Verbund als auch um einen 1-zu-n-Verbund handeln. Führt man dies mit vertauschter Reihenfolge der Tabellen noch einmal durch, ist der Verbund gefunden. In manchen Teilen des Gesamtalgorithmus kann es sein, dass eine Verbundkardinalität schon vorher ausgeschlossen wurde. Dadurch ist es nicht immer nötig diese Überprüfung in beide Richtungen durchzuführen. Die hier gezeigte Implementation wird genutzt, damit bereits beim ersten Attributwert, welcher in der anderen Relation mehr als einmal auftritt, abgebrochen und unnötige Berechnungen übersprungen werden können.
Wurde auf Fremdschlüsselbeziehungen, herausfallende Tupel bzw. Äquivalenzklassen und auf die Art des Verbundes geprüft, bleibt nur noch das Ausführen der richtigen Stored Procedure mit den entsprechenden Parametern. Da hierfür die (Nicht-)QIs der Ausgangsrelationen als bekannt vorausgesetzt sind, nehmen wir hier an, dass diese in einer designierten Tabelle abgelegt sind.
Mit Ausnahme von Tabellenkommentaren sind alle weiteren Informationen über Relationen in den Systemtabellen eines Datenbanksystems abgelegt. Eigene Statistiken, welche man für die Datenbank überwachen möchte, muss man selbst verwalten. Deshalb schlagen wir vor, einen Trigger zu nutzen, welcher die QIs einer Tabelle nach Quellcodeausschnitt 4.2 berechnet und in einer für QIs designierten Tabelle ablegt. Da allerdings ein Trigger immer zu genau einer Tabelle gehört, muss dieser für jede Tabelle explizit erstellt werden. Die designierte QI-Tabelle könnte beispielsweise wie folgt aussehen:

```
1 CREATE TABLE qiinformation(
2   t_name varchar(63),
3   qi varchar[][],
4   non_qi varchar[][],
5
6   PRIMARY KEY(t_name)
7 );
```

Quellcodeausschnitt 4.15: Aufbau der QI-Tabelle

Weiter müsste der Trigger mit jeder neuen Berechnung der QIs das alte Tupel löschen und durch ein Neues ersetzen. Die größte Problematik ist aber, wann QIs neu berechnet werden sollen. Werden Quasi-Identifikatoren zu häufig neu berechnet, kann sich das stark negativ auf die Performance der Datenbank auswirken. Andererseits bedeutet das seltenere Updaten der QIs Ungenauigkeit, die wiederum den Schutz der personenbezogenen Daten der Datenbank gefährdet.

.

Kapitel 5

Evaluation

In diesem Kapitel werden die Algorithmen, welche im vorherigen Teil der Arbeit vorgestellt wurden, anhand einer Testdatenbank ausgewertet. Dabei wird vor allem die letzte Implementation, welche die optimale Stored Procedure zur Berechnung der QIs findet, mit n-zu-m-Variante für Verbunde verglichen. Grund dafür ist, dass letztere der Anwendung des ursprünglichen QI-Algorithmus auf eine Verbundrelation entspricht.

5.1 Testumgebung

Das Test-Setup ist eine lokale PostgreSQL-Installation der Version 11, auf einem Laptop mit 8 GB Arbeitsspeicher, einer 4-Kern CPU mit jeweils 2,2 GHz ohne Hyperthreading. Als Plattenspeicher dient eine SSD und das verwendete Betriebssystem ist die 64-Bit-Version von Windows 10. Weiter wurden am Testrechner keine Einstellungen vorgenommen, die Einfluss auf die Performance haben. Zusätzlich wurden folgende Einstellungen für PostgreSQL getroffen:

Parameter	**Wert**
max_connections	100
shared_buffers	2 GB
temp_buffers	80 MB
work_mem	1024 MB
maintenance_work_mem	24 MB
max_stack_depth	2 MB
max_files_per_process	1000

Tabelle 5.1: Parameter postgresql.conf

Die Datenbank, die als Testszenarium verwendet wurde, ist die TPC-H-Datenbank ([TPC19]), welche für Benchmarks im Bereich der Entscheidungsunterstützungssysteme genutzt wird. Die Datenbank wurde mit Beispieldaten befüllt, welche durch DBGEN ([DBG19]) mit dem Skalierungsfaktor 1 generiert wurden. Die Gesamtgröße der so generierten Daten beträgt ca. 1,1 GB. Da in dieser Datenbank Tabellen ausschließlich über Fremdschlüsselbeziehungen verbunden sind, werden in einigen Testfällen Verbunde gebildet, welche im Kontext einer echten Anwendung untypisch wären. Diese sind aber nötig, um die drei verschiedenen Verbundkardinalitäten darstellen zu können.

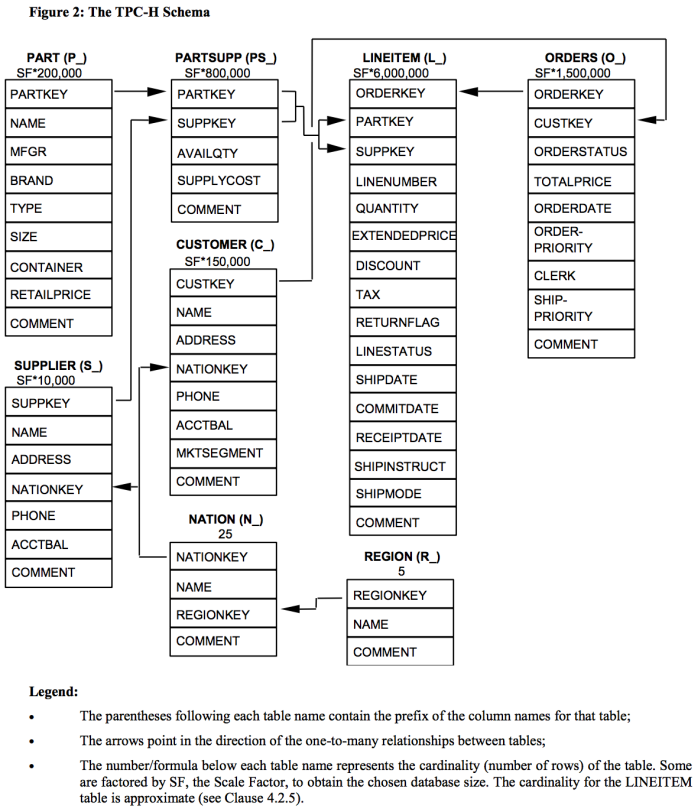


Abbildung 5.1: Schema der TPC-H-Benchmark-Datenbank
Quelle: [THD19]

5.2 Laufzeiten

Relation	Anzahl Tupel	Anzahl Attribute	∅Ausführungszeit(hh:mm:ss.SSS)
LINEITEM	6.000.000	16	*
ORDERS	1.500.000	9	00:01:01.198
PARTSUPP	800.000	5	00:00:11.632
PART	200.000	9	00:00:10.676
CUSTOMER	150.000	8	00:00:01.665
SUPPLIER	10.000	7	00:00:00:101
NATION	25	4	00:00:00.016
REGION	5	3	00:00:00.015

Tabelle 5.2: Ausführungszeit $qi\_detect(Relation, 0.70)$
* - Laufzeit unbekannt, da nach über einer Stunde abgebrochen

Aus der Tabelle 5.2 ist ablesbar, dass *LINEITEM* für diese Auswertung ungeeignet ist. Das liegt an der hohen Anzahl von Attributen und Tupeln, gepaart mit den vielen Fremdschlüsseln, welche sehr unwahrscheinlich QIs sind. Daher finden sich die meisten QIs erst in höheren Ebenen des Attributgitters. Da der implementierte Algorithmus ein Bottom-Up-Ansatz ist und das Gitter nicht bidirektional traversiert, werden diese QIs erst sehr spät gefunden. Deshalb wird *LINEITEM* direkt als Tabelle für Tests

ausgeschlossen. *ORDERS* ist allerdings für die Auswertung von Verbundtabellen ebenfalls problematisch, da einige weitere Attribute bereits ausreichen, um die Laufzeit sehr stark zu erhöhen. Als Beispiel hierfür betrachten wir *qi_ detect_ join('customer','orders','c_ custkey','o_ custkey',0.70).* Aufgrund sehr langer Ausführungszeit wurde die Laufzeit dieses Funktionsaufrufs nicht gemittelt, sodass nur eine einzelne Messung betrachtet wird. Die Ausführungszeit hierfür beträgt mit den oben genannten Einstellungen mehr als 5 Stunden, wobei hier die Nicht-QIs der Ausgangsrelationen nicht in die Berechnung eingeflossen sind. Dies Laufzeitersparnis, welche durch die Betrachtung der Nicht-QIs erreicht werden würde, ist aber angesichts der Laufzeiten des QI-Algorithmus für eine Tabelle eher gering. Dies veranschaulicht das Problem, welches das exponentielle Wachstum des Suchraums darstellt.

Attributel	**Anzahl**
1	1
2	24
3	26
4	64
5	54
6	3

Tabelle 5.3: Quasi-Identifikatoren von *lineitem* nach Attributanzahl; Treshold = 0.70 [1]

5.2.1 Allgemeine Einflussfaktoren

Für die Laufzeit des QI-Algorithmus gibt es drei bedeutende Faktoren:

1. Anzahl der Tupel:
   Für die Berechnung des Identifikationswertes einer Attributkombination müssen Anfragen folgender Form für jede Attributkombination, die noch ein QI-Kandidat ist, durchgeführt werden:

   ```
   SELECT COUNT(*) FROM (SELECT DISTINCT attributes FROM tbl)
   ```

   Die Laufzeit dieser Anfrage ist hauptsächlich von der Anzahl der Tupel in *tbl* abhängig, die Anzahl der Attribute in der Attributkombination spielt ebenfalls eine Rolle (siehe [GH14]), wird hier aber vorerst vernachlässigt. Da in dieser Anfrage ein *DISTINCT* vorkommt, muss die Relation zunächst nach den angegebenen Attributen sortiert bzw. eine Hash-Tabelle für diese Attribute aufgebaut werden. Der Aufwand dafür liegt bekanntlich in $O(n * log(n))$ bzw. $O(n)$ (n ist hier die Anzahl der Tupel). Anschließend müssen diese noch mit *COUNT* gezählt werden. Der Aufwand hierfür liegt ebenfalls in $O(n)$ und muss noch zum Aufwand des *DISTINCT* hinzuaddiert werden.

2. Anzahl der Attribute:
   Die obige Anfrage muss im schlechtesten Fall für jede Attributkombination durchgeführt werden. Das bedeutet einen Aufwand von $(2^m - 1) *$ Aufwand der obigen Anfrage. Hierbei ist m die Anzahl der Attribute. Im besten Fall muss die obige Anfrage nur m-mal ausgeführt werden. In diesem Fall wären alle Attribute Quasi-Identifikatoren, sodass die Berechnung hier bereits abgebrochen werden kann.

3. Identifikationswert der Attribute:
   Dieser Faktor bestimmt, wie viele Attributkombinationen wirklich überprüft werden. Je höher der Identifikationswert ($\frac{\#\text{distinct values}}{\#\text{tuples}}$) der einzelnen Attribute, desto wahrscheinlicher ist es, dass viele

[1] Hannes Grunert, persönliche Mitteilung vom 12.03.2019. Wurde mit einem bidirektional arbeitenden Algorithmus ermittelt, da der hier vorgestellte Algorithmus auf Grund zu langer Laufzeit abgebrochen werden musste.

QIs in den unteren Ebenen des Attributgitters gefunden werden. Da die vorgestellte Implementation das Gitter nicht bidirektional traversiert, dauert die Berechnung der QIs länger, je mehr Attribute gebraucht werden, um den mit *threshold* angegebenen Identifikationswert zu erreichen. Dieses Maß ist allerdings nicht quantifizierbar, da es allein von den Daten in der Relation abhängt. Lediglich die Eigenschaften der Attribute können hier weitere Hinweise geben. So haben Schlüssel und als *UNIQUE* markierte Spalten offensichtlich einen Identifikationswert von 1. Im Gegensatz dazu haben Fremdschlüssel und Attribute mit eingeschränktem Wertebereich im Durchschnitt einen geringeren Identifikationswert, da die Dopplung von Attributwerten hier wahrscheinlicher ist. Dadurch entsteht eine Abhängigkeit der Laufzeit vom geforderten Grenzwert für QIs.

Beim Erweitern des *lowerSet*s entsteht durch die Array-Implementation ein zusätzlicher Aufwand. Da durch die Arrays nur Mengen simuliert werden, müssen doppelt generierte Attributkombinationen vor dem Einfügen gefunden werden.

Beispiel: doppelt generierte Attributkombinationen
Betrachten wir die Beispielrelation R: R={A,B,C}. So wäre das erste *lowerSet* offensichtlich **{{A},{B},{C}}**. Bei der Überprüfung der Elemente des *lowerSet*s stellte sich heraus, dass **C** ein Quasi-Identifikator ist. Demnach fallen alle weiteren Attributkombinationen, die **C** enthalten aus den neu generierten *lowerSet*s heraus. Für **A** und **B** muss allerdings noch das *lowerSet* erweitert werden:

$$\mathbf{\{A\}} \xrightarrow{\textbf{Erweiterung mit B}} \mathbf{\{A,B\}}$$
$$\mathbf{\{B\}} \xrightarrow{\textbf{Erweiterung mit A}} \mathbf{\{B,A\}}$$

Da diese erweiterten Mengen gleich sind, soll nur eine von beiden in das neue *lowerSet*.

Dafür muss das gesamte *lowerSet*, welches gerade schrittweise aufgebaut wird, durchsucht werden. Nachfolgende Messungen zeigen jedoch, dass dieser Aufwand nur einen sehr geringen Anteil der Gesamtlaufzeit ausmacht:

Relation	**Anteil** $t_{\textbf{Berechnung QI}}$	**Anteil** $t_{\textbf{Erweiterung lowerSet}}$
ORDERS	99,51%	00,02%
PARTSUPP	91,37%	00,12%
PART	95,53%	00,12%
CUSTOMER	98,19%	00,19%

Tabelle 5.4: Durchschnittlicher prozentualer Anteil der Teilschritte $qi\_detect(Relation, 0.70)$

5.2.2 In Abhängigkeit von Parametern

Im Nachfolgenden werden die Laufzeiten in Abhängigkeit verschiedener Einflussfaktoren gezeigt. Die Werte wurden aus jeweils zehn Messungen gemittelt.

Grenzwert für Quasi-Identifikatoren

Wir haben bereits herausgestellt, dass der Grenzwert für einen Quasi-Identifikator Einfluss auf die Laufzeit des Algorithmus nehmen muss. Je geringer der Grenzwert, desto weniger wahrscheinlich sind QIs, die viele Attribute enthalten. Folglich entfallen bei dem hier evaluierten Bottom-Up-Ansatz früher Berechnungspfade, sodass der Algorithmus früher terminiert. Nachfolgende Laufzeitmessung für *qi_ detect('orders',threshold)* spiegelt dies wider. Dabei wurden für *threshold* Werte zwischen 0.0 und

1.0 (jeweils in 0.1er Schritten, zwischen 0.90 und 1.0 in 0.01er Schritten) getestet. Zu beachten ist aber, dass die QIs weiterhin von den Daten selbst abhängig sind, sodass für manche nah beieinanderliegende Werte für *threshold* die QIs der Relation gleich bleiben. Dadurch ist auch die entsprechende Laufzeit sehr ähnlich.

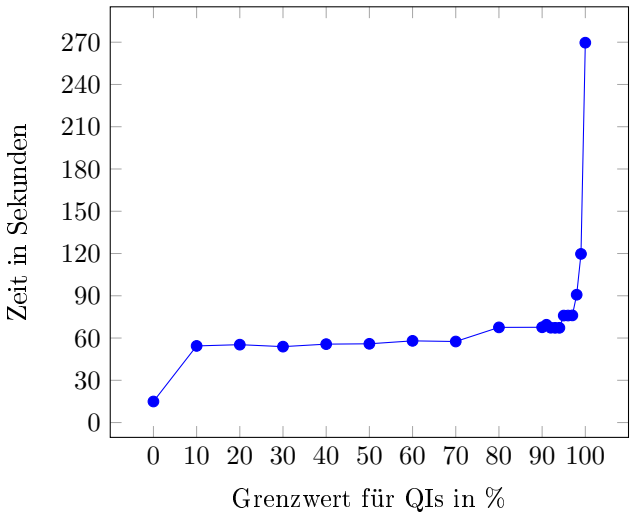


Abbildung 5.2: Laufzeitmessung: *qi_ detect('orders',threshold)* in Abhängigkeit von *threshold*

Tupelzahl

Die Tupelzahl hat Einfluss auf die Anfrage, welche eine Attributkombination auf die QI-Eigenschaft prüft. Wie in Tabelle 5.4 gezeigt, macht dies den Hauptaufwand der Berechnung aus. Da die Implementation, welche in dieser Arbeit vorgestellt wurde, sich nur auf die Gesamtheit einer Tabelle bezieht, werden wir im Folgenden nicht die Laufzeit der Funktion evaluieren, sondern die Anfrage für die Laufzeitmessung anpassen. Dazu wird immer nur ein Ausschnitt *ORDERS*-Tabelle betrachtet. Um zu vermeiden, dass der Aufwand für das Erstellen und Löschen der vielen Tabellen für diese Ausschnitte in die Laufzeitmessung einfließt, wird dieses Slicing in die Anfrage übertragen. Dazu wird eine Unteranfrage mit einem Limit eingeführt, welches die Tupelzahl beschränkt. Die Anfrage hat danach die Form:

```
SELECT COUNT(*)
FROM(
    SELECT DISTINCT attr1, attr2, ...
    FROM(
        SELECT *
        FROM orders
        LIMIT x
    )a
)b
```

Quellcodeausschnitt 5.1: Bestimmung der unterschiedlichen Werte einer Attributkombination (für Laufzeitmessung angepasst)

Dabei ist x die entsprechende Zahl von Tupeln, die in diese Anfrage einfließen sollen. Die Attribute, für welche eine Duplikateliminierung durchgeführt werden soll, wurden auf *o_custkey*, *o_orderstatus*, *o_orderpriority* und *o_shippriority* festgelegt. Diese Attributkombination wurde gewählt, weil die Laufzeit der Anfrage bei dieser Anzahl an Attributen nicht mehr im Millisekundenbereich liegt und diese nicht

den Schlüssel der Tabelle enthält. Hier wird deutlich sichtbar, dass die Laufzeit linear von der Anzahl der Tupel abhängt, weil das Datenbanksystem Hashing für die Duplikateliminierung wählt.

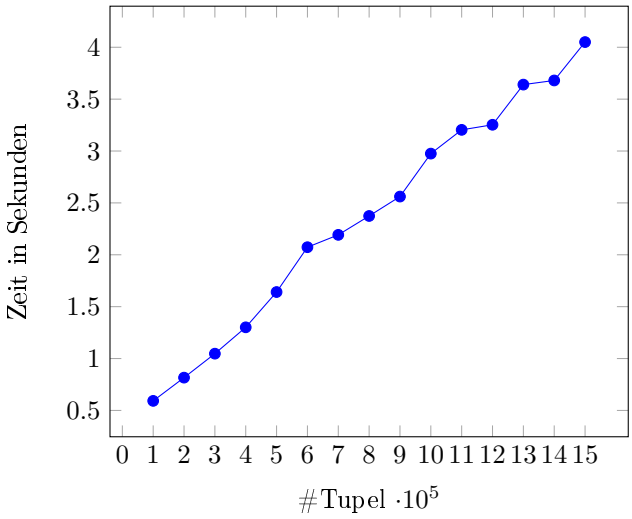


Abbildung 5.3: Laufzeitmessung: Anfrage 5.1 in Abhängigkeit von der Tupelzahl

Attributzahl

Die Anzahl der Attribute beeinflusst die Laufzeit für die Überprüfung der QI-Eigenschaft einer Attributkombination und die Anzahl der zu untersuchenden Attributkombinationen. Um zu verhindern, dass das Ergebnis davon beeinflusst wird, welche Spalten für die Laufzeitmessung ausgewählt werden, wurde für jede mögliche Kombination der Spalten der Ausgangstabelle in eine separate Tabelle erstellt. Dies bedeutet allerdings, dass $2^{\text{Anzahl Attribute}} - 1$ Tabellen benötigt werden. Im Folgenden wird statt der *ORDERS*-Tabelle die *PARTSUPP*-Tabelle betrachtet, damit nicht über 500 Tabellen mit jeweils 1500000 Tupeln, sondern nur 32 Tabellen mit jeweils 800000 Tupeln erstellt werden müssen. Nachfolgende Laufzeiten wurden über alle Tabellen mit gleicher Anttributzahl gemittelt:

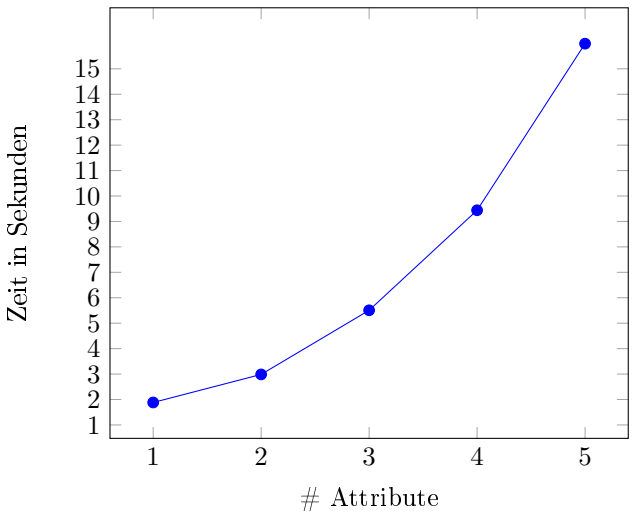


Abbildung 5.4: Laufzeitmessung: *qi_detect('orders',0.70)* in Abhängigkeit von der Attributzahl

Verbundkardinalität

Die Verbundkardinatlität hat einen Einfluss darauf, welche (Nicht-)QIs der Ausgangsrelationen für gültig auf der Verbundrelation angenommen werden dürfen. Da diese bereits bekannten Attributkombinationen nicht noch einmal untersucht werden müsse, wirkt sich die Art des Verbundes auch auf die Laufzeit aus. Um die Bedingungen für die Laufzeitmessungen von *qi_detect_join* möglichst fair für die verschiedenen Kardinalitäten zu halten, wurden folgende Rahmenbedingungen festgelegt:

- Die Anzahl der Spalten und Tupel ist für jede Verbundrelation gleich.
- Die Größe der zu untersuchenden Tabellen lässt die Laufzeit den Minutenbereich nicht überschreiten.

Dafür wurden entsprechend der Fallunterscheidungen in *qi_detect_join* verschiedene Variationen der *PART*-Tabelle erstellt. Folgende Tabellen werden im nachfolgend genutzt:

- **MODPART**
  Diese Tabelle entspricht der Projektion der *PART*-Tabelle auf $\{PARTKEY, BRAND, SIZE, CONTAINER, RE$
- **CPART**
  Diese Tabelle ist gleich *MODPART*, wobei das Prefix der Attributnamen geändert wurde, damit Tabellen, die miteinander verbunden werden, keine gleichnamigen Attribute besitzen.
- **DPART**
  Diese Tabelle entspricht der *MODPART*-Tabelle mit einigen zusätzlichen Tupeln. Auch diese Tabelle hat ein eigenes Prefix für Attributnamen.
- **FCPART und FDPART**
  Diese Tabellen entsprechen jeweils *CPART* und *DPART*. Zusätzlich wurde ihnen ein Fremdschlüssel auf dem jeweiligen *PARTKEY*-Attribut hinzugefügt, welches *PARTKEY* in *MODPART* referenziert.
- **NPART und MPART**
  Jede dieser Tabellen entspricht einem Ausschnitten der *CPART*- bzw. *DPART*-Tabelle (200 bzw. 1000 Tupel). Zusätzlich sind in beiden Tabellen alle *PARTKEY*s gleich 1.

Diese Tabellen lassen sich so verbinden, dass die Anforderung der gleichen Tupel- und Attributanzahl für alle Verbunde eingehalten wird. Dazu werden folgende Verbunde gebildet:

	mit Fremdschlüssel	**ohne Fremdschlüssel**
1-zu-1	$MODPART \bowtie FCPART$	$MODPART \bowtie CPART$
1-zu-n	$MODPART \bowtie FDPART$	$MODPART \bowtie DPART$
n-zu-m	*unmöglich*	$NPART \bowtie MPART$

Tabelle 5.5: Verbunde für Laufzeitmessungen: *qi_detect_join*

Diese Verbunde sorgen zwar für die Einhaltung der ersten Rahmenbedingung, aber nicht für eine annehmbare Laufzeit. Die sehr hohe Laufzeit der *qi_join_detect*-Funktion liegt an der Erkennung der Verbundkardinalität (siehe Quellcodeausschnitt 4.14). Diese Erkennung beinhaltet eine dynamisch generierte Anfrage in einer Schleife, was besonders bei sehr vielen unterschiedlichen Werten für das Verbundattribut sehr lange dauert. Dies wird z. B. bei Verbunden über Primärschlüssel, wie auch in den hier vorgeschlagenen Testverbunden, problematisch. Deshalb schlagen wir vor, diese Erkennung nicht zu nutzen, sofern die Ausgangsrelationen nicht sehr viele Attribute besitzen. Dadurch tauscht man bei Tabellen mit wenigen Attributen eine große Laufzeitersparnis gegen die Möglichkeiten ein, einige der auftretenden Fälle

zu unterscheiden. Ohne diese Anfrage sind 1-zu-1-Verbunde über einen Fremdschlüssel nicht mehr von 1-zu-n-Verbunden über Fremdschlüssel nicht mehr unterscheidbar. Ebenfalls wird dann bei einem Verbund ohne Fremdschlüssel immer von einem n-zu-m-Verbund ausgegangen. Weiter besteht auch die Möglichkeit die Funktionen für die einzelnen Verbundkardinalitäten direkt aufzurufen, sofern man sich der Art des Verbundes sicher ist. Dies kann aber beispielsweise aus dem Anwendungskontext bekannt sein. Im Folgenden wird die optimierte Variante, welche die Verbundkardinalität nicht genauer untersucht, betrachtet:

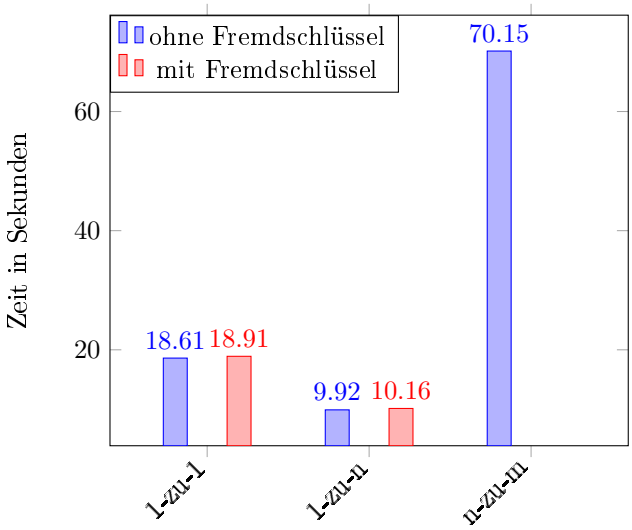


Abbildung 5.5: Laufzeitmessung: *qi_detect_join_opt* in Abhängigkeit der Verbundkardinalität

Aus den Laufzeitmessungen geht hervor, dass die Abweichungen zwischen der Berechnung mit Fremdschlüssel und ohne Fremdschlüssel sehr klein sind. Um Unterschiede durch die Daten auf der Verbundrelation auszuschließen, betrachten wir abschließend die Funktionen für die einzelnen Verbundkardinalitäten. Dazu wird jede der drei Funktionen mit dem gleichen Verbund untersucht. Dabei wird der Verbund von *MODPART* mit *CPART* über ihre Schlüssel mit einem Grenzwert von 0.7 verwendet:

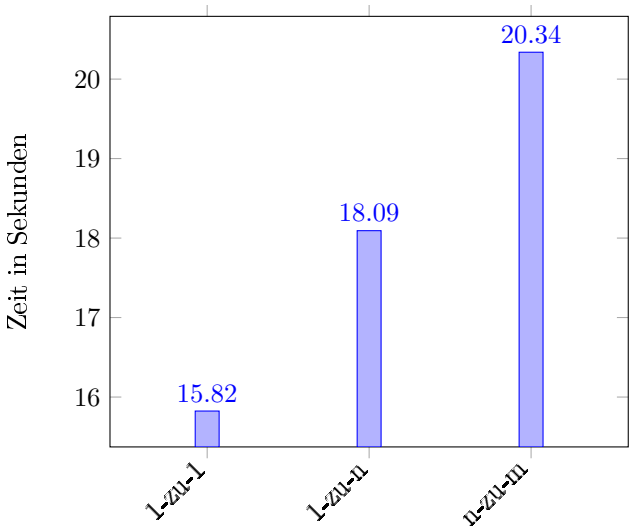


Abbildung 5.6: Laufzeitmessung: Funktionen nach Verbundkardinalität

Sichtbar ist, dass eine deutliche Einsparung der Laufzeit durch das Verwenden der richtigen Funktion mir den (Nicht-)QIs der Ausgangsrelationen erreicht wird.

5.3 Vor- und Nachteile

In diesem Abschnitt werden die Vor- und Nachteile der Implementation als Stored Procedure diskutiert. Betrachten wir die Implementation zunächst aus Sicht des Datenschutzes. Aus dieser Perspektive ist die Implementation als SQL-Funktion sehr vorteilhaft. Die Verarbeitung findet nahe an den Daten statt, sodass diese gar nicht erst über Netzwerke verschickt werden müssen, um von einem Client verarbeitet zu werden. Damit können die Daten nicht bei der Übertragung von Server zu Client abgefangen werden. Auch gibt es clientseitig nicht die Möglichkeit, auf die Daten durch das Auslesen des Speichers des Client-Rechners zuzugreifen. Weiter werden keine neuen Daten generiert, die nicht ohne Weiteres veröffentlicht werden dürfen. Auch im Hinblick auf Performance hat die Implementation als Stored Procedure auf der Datenbank Vorteile. Beispielsweise müssen Ergebnisse für Berechnungen, wie Anfrageergebnisse von Teilschritten, nicht immer zwischen Client und Datenbank kommuniziert werden. Dadurch wird ein erhöhter Kommunikationsaufwand vermieden. Allerdings ist die Implementation mit PostgreSQLs aktuell gegebenen Möglichkeiten schwierig. Das Ersetzen eines Datentyps für Mengen durch Arrays führt zu erhöhtem Aufwand. Die Verwendung von temporären Tabellen als Ersatz für einen Mengentyp wurde nicht umgesetzt, um den erhöhten Aufwand des häufigen Auf- und Abbaus von Tabellen zu vermeiden. Weiter erlaubt das Berechnen der Quasi-Identifikatoren auf der Datenbank auch, diese dort zu halten, ohne dass man sie dafür erst vom Client an den Datenbankserver schicken muss. Durch Trigger können die Quasi-Identifikatoren automatisch aktualisiert werden, indem man die entsprechenden Stored Procedures aufruft. Dies hat Vorteile aus Sicht des Datenschutzes. Durch das Speichern der QIs in der Datenbank können zusammen mit der feingranularen Zugriffskontrolle, die SQL für Datenbaken ermöglicht, Zugriffsrechte datenschutzfreundlich angepasst werden. So können z. B. Zugriffsrechte so verteilt werden, dass nicht jeder Datenbanknutzer Anfragen stellen darf, die sensible Informationen und Quasi-Identifikatoren zusammen anzeigen.

Kapitel 6

Zusammenfassung und Ausblick

In diesem letzten Kapitel werden die Struktur und Ergebnisse der Arbeit noch einmal zusammengefasst. Weiter wird ein Ausblick darauf gegeben, welche Fragen bzw. Problemstellungen in dieser Arbeit nicht betrachtet wurden und deshalb noch offen sind.

6.1 Zusammenfassung

Eingangs haben wir mit der Einleitung (1) die Thematik der Quasi-Identifikatoren im Datenschutz eingeführt. Weiter wurde dort die wissenschaftliche Relevanz des Themengebietes an einem Beispiel dargelegt und gezeigt, warum die spezielle Problemstellung der Berechnung von Quasi-Identifikatoren nach Verbundoperationen durch die Explosion des Suchraums problematisch werden kann.
In Kapitel 2 wurden zunächst einige Grundlagen zum Datenschutz eingeführt, die noch einmal die Relevanz des Forschungsgebietes unterstrichen. Anschließend wurde vom Finden von funktionalen Abhängigkeiten und Schlüsseln zum Finden von Quasi-Identifikatoren und Unique-Column-Combinations, welche den Quasi-Identifikatoren stark ähneln, übergeleitet.
Im 3. Kapitel wurde vorgestellt, wie die Problemstellung dieser Arbeit gelöst werden kann. Dazu wurde ein Konzept entwickelt, mit dessen Hilfe die Berechnung von Quasi-Identifikatoren auf Verbundrelationen beschleunigt werden soll. Hierbei wurden Ansätze vorgestellt, die es erlauben, durch Bestimmung von Verbundkardinalitäten und Finden von Fremdschlüsselbeziehungen Berechnungen zu sparen. Weiter wurden Ansätze skizziert, wie es möglich wäre die Berechnungen parallel zum Aufbau der Verbundrelation durchzuführen.
Im darauffolgenden Kapitel (4) wurden die Ansätze aus Kapitel 3 in SQL implementiert. Dabei wurde auch Besonderheiten von SQL eingegangen, die die Implementierung erleichtern bzw. erschweren. Wir haben sowohl die Implementation für die Berechnung der Quasi-Identifikatoren einer einzelnen Tabelle vorgestellt, als auch gezeigt, wie bei der Berechnung auf einer Verbundrelation vorgegangen werden sollte, um möglichst effizient verschiedene Fälle mit verschiedenen Berechnungen zu unterscheiden.
Abschließend wurde im 5. Kapitel ausgewertet, wie effizient die implementierten Algorithmen sind. Hierzu wurde eine Testumgebung mit einer Beispieldatenbank geschaffen.

6.2 Ausblick

In dieser Arbeit wurden Ansätze für die Lösung des Problems, Quasi-Identifikatoren auf Verbundrelationen zu berechnen, vorgestellt. Dabei wurden allerdings einige Ansätze, welche eventuell zur Verbesserung der vorgestellten Algorithmen beitragen könnten, nicht im Detail betrachtet.

6.2.1 Berechnung während der Verbundoperation

Die Berechnung während der Verbundoperation wurde im Konzept skizziert, aber nicht detailliert auf ihre Anwendbarkeit überprüft. Unklar ist weiter, wie effizient die Berechnung parallel zur Berechnung des Verbundes ist, da hierfür der Verbund selbst neu implementiert werden müsste. Hierbei wäre die Optimierung durch das Datenbanksystem wiederum eingeschränkt, sodass eventuell ein weiterer Kompromiss zwischen Optimierbarkeit der Implementation und Menge der einbeziehbaren Berechnungsmöglichkeiten eingehen muss. So wäre es z. B. möglich, während des Verbundes abzuschätzen, wie sich Attributkombinationen durch den Verbund in ihrem Identifikationswert ($\frac{\#\text{distinct values}}{\#\text{tuples}}$) verändern. Dafür kann das System dann nicht ohne Weiteres die Verbundoperation optimieren und bestimmen, ob sich ein Nested-Loop, Hash- oder Merge-Join am besten eignet. Voraussichtlich wäre es hier der Implementation überlassen, die gängigsten Verbundimplementationen für die Berechnung von QIs zu adaptieren, und selbst die effizienteste Verbundoperation auszuwählen. Offen ist demnach, wie sich Quasi-Identifikatoren effizient während des Verbundes berechnen lassen.

6.2.2 Optimierung der Algorithmen

Die vorgestellte Implementation nutzt Arrays in SQL. Wie an den gezeigten Quelltextausschnitten, den beschriebenen Problemen und Workarounds zu erkennen, ist PostgreSQL derzeitig nicht für solche Anwendungsfälle ausgelegt. Die datennahe Implementation der Berechnungen hat ihre Vorzüge, wie in der Evaluation (siehe Abschnitt 5.3) vorgestellt. Das Simulieren einer Menge durch Arrays in PostgreSQL ist umständlich, aber nötig, da es keinen Datentyp *Set*<*T*> gibt. Mengen werden nur durch Tabellen unterstützt. Um den Aufwand für das Erstellen und Nutzen mehrerer Tabellen zu umgehen, könnte man eine Tabelle mit einer *SetID* und dem *Element* aufbauen, sodass alle Mengen in einer Tabelle liegen. Allerdings würde dann eine *contains*-Funktion für Anfrageresultate benötigt werden. Andere Datenbanksysteme, z. B. DB2 ([IBM19]) von IBM und Oracle ([OD119]) bieten mit ihren Collection Types Implementationen für Mengen und Multimengen. Wie gut sich die Implementationen dieser kommerziellen Systeme eignet, wurde aber nicht weiter untersucht. Daher bleibt die Frage offen, ob es effizientere Wege gibt, diese Algorithmen auf der Datenbank selbst umzusetzen.
Die in dieser Arbeit vorgestellten Implementationen verfolgt eine Bottom-Up-Strategie zur Berechnung von Quasi-Identifikatoren. Die Laufzeit dieser könnte durch Erweiterungen verbessert werden. So könnte z. B. das Einschieben eines Top-Down-Berechnungsschrittes zwischen den Bottom-Up-Phasen (ähnlich wie in [GH14]) eine große Verbesserung der Laufzeit bringen. Grund dafür ist, dass gerade in Tabellen, die hauptsächlich QIs mit vielen Attributen besitzen, schneller gefunden werden können.
Weiter arbeitet die vorgestellte Implementation für die Berechnung auf Verbundrelationen ausschließlich mit Vergleichen zwischen der aktuell zu überprüfenden Attributkombination und den bereits bekannten (Nicht-)Quasi-Identifikatoren. Ein Test auf Überdeckung wurde nicht implementiert. Für QIs selbst ist dies nicht relevant, da die Attributkombinationen, welche einen QI enthalten, bei der Erweiterung des *lowerSets* in jedem Fall aussortiert werden. Für die Nicht-QIs könnte dies aber ein interessanter Ansatz sein, da so nur maximale Nicht-QIs in der Tabelle, welche die Informationen über (Nicht-)QIs hält, gespeichert werden müssen. Somit muss in *qi_detect_join* über weniger (Nicht-)QIs iteriert werden, was besonders für Tabellen mit vielen Attributen eine Laufzeitersparnis bedeuten könnte.
Eine weitere offene Fragestellung ist, ob sich die Implementation noch flexibler gestalten lässt. Die in dieser Arbeit vorgestellten Algorithmen beschränkten sich jeweils auf zwei Tabellen. Dies auf eine feste Anzahl von Tabellen zu erweitern ist trivial, eine flexible Anzahl an Tabellen wäre aber für den Einsatz in der Praxis sehr sinnvoll. Auch wurde sich in dieser Arbeit auf natürliche Verbunde und Equi-Joins beschränkt. Eine interessante Fragestellung wäre demnach, wie sich eine beliebige Verbundanfrage so aufschlüsseln ließ, dass die Quasi-Identifikatoren für das entstehende Ergebnis berechnen lassen. Dies wäre für den realen Einsatz solcher Funktionen wichtig, da hier auch Selektionsbedingungen und die Verknüpfung mehrerer Tabellen durch verschiedenartige Verbunde relevant sind.

Literaturverzeichnis

[ABA18] *Aufgabenstellung: Berechnung von Quasi-Identifikatoren nach Verbundoperationen.* https://dbis.informatik.uni-rostock.de/studium/studentische-arbeiten/in-bearbeitung/berechnung-von-quasi-identifikatoren-nach-verbundoperationen-bis-032019/, 2018. Aufgerufen: 27.02.2019, 17:23.

[BDS17] *Bundesdatenschutzgesetz (BDSG) §71 Datenschutz durch Technikgestaltung und datenschutzfreundliche Voreinstellungen.* https://www.gesetze-im-internet.de/bdsg_2018/__71.html, 2017. Aufgerufen: 27.11.2018, 14:33.

[Dal86] Dalenius, Tore: *Finding a Needle In a Haystack.* 1986.

[DBG19] *electum/tpch-dbgen - GitHub.* https://github.com/electrum/tpch-dbgen, 2019. Aufgerufen: 01.03.2019, 13:36.

[GH14] Grunert, Hannes and Andreas Heuer: *Big Data und der Fluch der Dimensionalität: Die effiziente Suche nach Quasi-Identifikatoren in hochdimensionalen Daten.* In Klan, Friederike, Günther Specht, and Hans Gamper (editors): *Proceedings of the 26th GI-Workshop Grundlagen von Datenbanken, Bozen-Bolzano, Italy, October 21st to 24th, 2014.*, volume 1313 of *CEUR Workshop Proceedings*, pages 29–34. CEUR-WS.org, 2014.

[GH16] Grunert, Hannes and Andreas Heuer: *Datenschutz im PArADISE.* Datenbank-Spektrum, 16(2):107–117, 2016.

[GS18] *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.3 Percent in 2019.* https://www.gartner.com/en/newsroom/press-releases/2018-09-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2019, 2018. Aufgerufen: 27.02.2019, 17:13.

[IBM19] *Collection, record, and object types (PL/SQL).* https://www.ibm.com/support/knowledgecenter/en/SSEPGG_10.5.0/com.ibm.db2.luw.apdv.plsql.doc/doc/c0053895.html, 2019. Aufgerufen: 13.03.2019, 17:39.

[JAK82] Jacobs, Barry E., Alan R. Aronson, and Anthony C. Klug: *On interpretations of relational languages and solutions to the implied constraint problem.* ACM Trans. Database Syst., 7(2):291–315, 1982.

[KL51] Kullback, Solomon and Richard A Leibler: *On information and sufficiency.* The Annals of Mathematical Statistics, pages 79–86, 1951.

[Kle98] Klettke, Meike: *Akquisition von Integritätsbedingungen in Datenbanken*, volume 51 of *DISDBIS*. Infix Verlag, St. Augustin, Germany, 1998.

[Klu80] Klug, Anthony C.: *Calculating constraints on relational expressions.* ACM Trans. Database Syst., 5(3):260–290, 1980.

[LLV07] Li, Ninghui, Tiancheng Li, and Suresh Venkatasubramanian: *t-closeness: Privacy beyond k-anonymity and l-diversity.* In Chirkova, Rada, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis (editors): *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 106–115. IEEE Computer Society, 2007.

[MGKV06] Machanavajjhala, Ashwin, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam: *l-diversity: Privacy beyond k-anonymity.* In Liu, Ling, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang (editors): *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 24. IEEE Computer Society, 2006.

[MNSS15] Matwin, Stan, Jordi Nin, Morvarid Sehatkar, and Tomasz Szapiro: *A review of attribute disclosure control.* In Navarro-Arribas, Guillermo and Vicenç Torra (editors): *Advanced Research in Data Privacy*, volume 567 of *Studies in Computational Intelligence*, pages 41–61. Springer, 2015.

[MSD19] *MySQL Dokumentation.* `https://dev.mysql.com/doc/refman/5.5/en/group-by-functions.html#function_count-distinct`, 2019. Aufgerufen: 24.01.2019, 12:30.

[NPO18] *FAQ: Was wir über den Skandal um Facebook und Cambridge Analytica wissen.* `https://netzpolitik.org/2018/cambridge-analytica-was-wir-ueber-das-groesste-datenleck-in-der-geschichte-von-facebook-wissen/`, 2018. Aufgerufen: 01.12.2018, 14:20.

[OD119] *Using PL/SQL Collections and Records.* `https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/collections.htm#LNPLS005`, 2019. Aufgerufen: 13.03.2019, 17:39.

[PN17] Papenbrock, Thorsten and Felix Naumann: *A hybrid approach for efficient unique column combination discovery.* In Mitschang, Bernhard, Daniela Nicklas, Frank Leymann, Harald Schöning, Melanie Herschel, Jens Teubner, Theo Härder, Oliver Kopp, and Matthias Wieland (editors): *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs ¨Datenbanken und Informationssysteme¨ (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings*, volume P-265 of *LNI*, pages 195–204. GI, 2017.

[SS98] Samarati, Pierangela and Latanya Sweeny: *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression.* 1998.

[SSH13] Saake, Gunter, Kai-Uwe Sattler, and Andreas Heuer: *Datenbanken - Konzepte und Sprachen, 5. Auflage.* MITP, 2013.

[THD19] *TPC-H - Dokumentation.* `http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf`, 2019. Aufgerufen: 13.03.2019, 14:28.

[TPC19] *TPC-H - Homepage.* `http://www.tpc.org/tpch/`, 2019. Aufgerufen: 01.03.2019, 13:33.

Anhang A

Aufbau des Datenträgers

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

Arbeit

- Digitale Fassung der Arbeit (Bachelorarbeit_Florian_Rose.pdf)
- Tex-Dateien der Arbeit
- verwendete Abbildungen (*media*-Ordner)

Code

- vollständige Implementation der vorgestellten Algorithmen

Experimente

- nicht eigenständig erhobene Daten (*foreign_data*-Ordner)
- Skripte zur Erhebung von Messdaten (*measurement_scripts*-Ordner)
- Erhobene Messdaten (*measurements*-Ordner)
- TPC-H-Testdatensatz (*test_data_tpch*-Ordner)

Literatur

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

Selbständigkeitserklärung

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

Rostock, den 18.03.2019