Intensional Answers for Provenance Queries in Big Data Analytics

Jan Svacina
Matrikelnummer: 5100190

BACHELORARBEIT

eingereicht am
Lehrstuhl für Datenbank- und Informationssysteme

Institut für Informatik

Universität Rostock

am 29.02.2016

Gutachter:

Prof. Dr. Andreas Heuer
Dr. Holger J. Meyer

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Universität Rostock, am 29. Februar 2016

Jan Svacina

Inhaltsverzeichnis

Erklärung i

Kurzfassung iv

Abstract v

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Aufgabenstellung und Zielsetzung . . . 2
- 1.3 Fortlaufendes Beispiel . . . 3

2 Grundlagen 5
- 2.1 Das Relationenmodell . . . 5
  - 2.1.1 Relationale Operatoren . . . 6
- 2.2 Klassifikation von Attributen . . . 8
- 2.3 Intensionale Antworten . . . 9
- 2.4 Data Provenance . . . 10
- 2.5 Big Data Analytics . . . 10

3 Stand der Forschung 12
- 3.1 Provenance . . . 12
  - 3.1.1 Why-Provenance . . . 12
  - 3.1.2 How-Provenance . . . 15
  - 3.1.3 Where-Provenance . . . 16
  - 3.1.4 Why-Not-Provenance . . . 17
  - 3.1.5 Big Data Provenance . . . 19
- 3.2 Intensionale Antworten . . . 19
  - 3.2.1 Generierung intensionaler Antworten . . . 19
  - 3.2.2 Klassifikation von intensionalen Antworten nach Motro 23
  - 3.2.3 Bewertung intensionaler Antworten nach Motro . . . 24

4 Stand der Technik 25
- 4.1 PERM Provenance Extension of the Relational Model . . . 25
- 4.2 Andere Provenance-Frameworks . . . 27

4.3 Auswahl des Frameworks 28

5 Berechnung von intensionalen Antworten auf Provenance-Anfragen 29
5.1 Methoden 29
5.1.1 Konzept 29
5.1.2 Konzepthierarchie 31
5.1.3 Granularität 32
5.2 Algorithmus zur Erstellung intensionaler Antworten auf Provenance-Anfragen 33
5.2.1 Algorithmus 1 (Erstellung der Konzepthierarchie) . . . 33
5.2.2 Algorithmus 2 (Erstellung von intensionalen Antworten) 33
5.2.3 Algorithmus 3. (Erstellung intensionaler Antworten auf Provenance Queries) 34
5.3 Einige Experimente an realen Datensätzen 36
5.3.1 Anwendungsfall 1 36
5.3.2 Anwendungsfall 2 40

6 Diskussion und Fazit 43
6.1 Diskussion des vorgestellten Lösungsansatzes 43
6.1.1 Bewertung der intensionalen Antworten 43
6.1.2 Vorteile der vorgestellten Methode 44
6.1.3 Nachteile der vorgestellten Methode 44
6.2 Ausblick 45
6.2.1 Automatische Erstellung der Konzepthierarchie 45
6.2.2 Die missing Answers 45
6.2.3 Privatheit 46
6.3 Zusammenfassung 46

A Technische Informationen 47
A.1 Aktuelle Programmversionen 47
A.1.1 Inhalt der DVD 47
A.2 Details zur Implementierung 47
A.3 Automatische Ausführung über eine VM 48
A.4 Ausführung des Programms 49

Quellenverzeichnis 50
Literatur 50

Kurzfassung

Provenance-Management im Bereich der Big Data Analytics bedient sich der extensionalen Form der Antworten. Bei sehr großen Datenmengen kann dies dazu führen, dass der Anwender die Provenance-Antwort nicht richtig interpretieren kann. Im Rahmen dieser Arbeit wird eine Methode entwickelt, die es erlaubt, intensionale Antworten auf eine Provenance-Anfrage zu erhalten. Das etablierte Forschungsfeld des Provenance-Management wird um das Konzept der intensionalen Antworten erweitert, um dem Anwender eine Rückverfolgung der Ergebnisse zu ermöglichen. Die Arbeit beschäftigt sich mit den Grundlagen der Provenance-Forschung und dem gegenwärtigen Stand der Provenance in der Big Data Analytics. Es wird ein dreistufiger Algorithmus vorgestellt, der die Rückverfolgung einer erhaltenen Antwort ermöglicht und die Provenance-Antwort in deskriptiver Form als intensionale Antwort zurückgibt. Das erlaubt dem Anwender, sich große Mengen von Daten explorativ zu erarbeiten. Ein modularer Aufbau der Methode ermöglicht eine Erweiterung durch andere Provenance-Algorithmen und Data Mining Techinken.

Abstract

Provenance Management in the field of Big Data Analytics makes use of extensional answers. This makes it difficult for a user to understand answers consisting of large amounts of data. This thesis proposes a method, that generates intensional answers from provenance queries. The established field of provenance management is expanded with the concept of intensional answers to allow the user to retrace his results. We propose a three layered algorithm to retrace a provenance answer and show it in its intensional form. The user will be capable of exploring the dataset, rather than just asking the questions. The modular structure of the proposed method allows to expand it with different provenance algorithms and data minig techniques.

Kapitel 1

Einleitung

1.1 Motivation

Datenbanksysteme (DBS) und Datenbankmanagementsysteme (DBMS) werden seit vielen Jahren erforscht, entwickelt und von den verschiedensten Berufszweigen genutzt. Seit den siebziger Jahren, insbesondere nach der Arbeit von Edgar F. Codd, der als der Erfinder der relationalen Datenbank gilt [12], ist das relationale Datenbankmodell nicht mehr aus Wissenschaft und Gesellschaft wegzudenken. Kommerzielle DBMS Anwendungen werden häufig für eine Nutzung in der Privatwirtschaft und der Öffentlichen Verwaltung entwickelt. Durch die Anforderungen dieses Marktes ergeben sich die Funktionen des DBMS. Ein Anwender aus dem privaten Bereich nutzt das System, um aus einer überschaubaren Datenmenge eine eindeutige, prägnante Antwort zu erhalten. In den vergangenen Jahren kamen viele Anwendungsfälle im Data-Warehouse Bereich hinzu. Hier sind die Datenmengen beträchtlich größer, allerdings sind die Ansprüche auf eine Antwort für den Anwender immer noch Kürze und Prägnanz. Durch den fortschreitenden Einsatz dieser Systeme unter anderem im wissenschaftlichen Umfeld, werden DBMS auch als Instrument der Wissensfindung genutzt [26]. Kersten et al. beschreiben in dem VLDB 2011 Visions and Challenges Artikel “The Researcher’s Guide to the Data Deluge: Querying a Scientific Database in Just a Few Seconds” die Situation von Wissenschaftlern [26]. Es geht nicht mehr um eine konkrete Antwort, sondern vielmehr darum, die richtigen Fragen zu stellen. Wissenschaftler wollen, anders als die zuvor genannten Anwender ihre Datensätze explorieren, denn anders als zum Beispiel der Angestellte einer Bank, kennen sie ihre Daten nur ungenügend. Eine mögliche Hilfe ist hier das Provenance Management. Es hilft dem Anwender, die durch das DBMS erstellten Antworten auf ihren Ursprung zurückzuverfolgen. Er erfährt , welche Teile seiner Daten an der Erstellung der Antwort beteiligt waren. Dieses kann dem Anwender helfen, seine Antwort zu verstehen und im nächsten Schritt eine neue, vielleicht genauere Frage zu stellen. Diese Daten können

aber selbst wieder sehr viele sein. In diesem Fall ist dieses Wissen unter Umständen nicht besonders hilfreich. Hier setzen die intensionalen Antworten an. Diese sollen aus den vielen Daten eine menschenlesbare Antwort erstellen und dabei keine oder Möglichst wenige Informationen verlieren.

1.2 Aufgabenstellung und Zielsetzung

Das Ziel dieser Arbeit ist es, eine Methode vorzustellen welche die Erstellung von intensionalen Antworten auf Provenance Anfragen in der Big Data Analyse [1] ermöglicht. Die Antworten, die ein DBS auf eine Anfrage liefert, sollen zu ihrem Ursprung zurückverfolgt werden. Diese Ursprungsdaten sollen dem Bereich der Big Data Analytics zuzuordnen sein. Das bedeutet, dass es sich weitgehend um sogenannte kontinuierliche Daten handelt. Dazu gehören zum Beispiel Sensoren, die ständig Messwerte liefern. Des weiteren gibt es beschreibende Daten, die etwas über die Sensoren aussagen, ohne sich mit der Zeit zu ändern. Diese werden in dieser Arbeit als Metadaten bezeichnet, da sie Aussagen über die eigentlichen Daten treffen.

In dieser Arbeit wird eine Methode vorgestellt, die es ermöglicht eine Antwort wieder auf den Ursprungsdatensatz zurück zu rechnen. Die so erhaltene Provenance-Antwort wird in einer intensionalen oder deskriptiven Form wieder ausgegeben.Es gibt drei großen Herausforderungen des Provenance-Managements in der Big Data Analytics:

1. Datenmengen. Die Ausgabe von großen Datenmengen als Anwort einer Provenance-Anfrage ist für den Anwender nicht immer hilfreich, da er sie nicht überschauen kann.
2. Kontinuierliche Daten. Diese werden unter Umständen vorgefiltert, deshalb sind zum Zeitpunkt der Analyse, die Originaldaten nicht mehr vorhanden.
3. Analysefunktionen der Big Data Analytics. Diese Unterscheiden sich zum Teil von klassischen Analysetechniken. Hier werden Algorithmen angewandt die aus dem Bereich des Machine Learning stammen und komplexere Datenbankfunktionen nutzen.

In dieser Arbeit wird der Versuch unternommen einen Beitrag zu den ersten beiden Punkten zu erbringen.

Um dieses Ziel zu erreichen, werden verschiedene bekannte Methoden zu einer neuen Methode verknüpft. Zunächst wird ein fortlaufendes Beispiel eingeführt, auf dessen Grundlage die Konzepte dieser Arbeit erläutert werden. Danach werden die Grundlagen der relationalen Datenbanken, des Provenance-Managements, der intensionalen Antworten, sowie der Begriff der Big Data Analytics eingeführt. In den darauf folgenden Kapiteln wer-

[1] Intensional Answers on Provenance Queries

den der Stand der Forschung und der Technik in den genannten Bereichen vorgestellt. Im Hauptteil der Arbeit werden Methoden und Algorithmen zur Lösung der genannten Frage vorgestellt und anhand von zwei Anwendungsfällen erläutert.

Zum Schluss werden die erhaltenen Ergebnisse und die Methoden diskutiert. Zusätzlich werden Anregungen für eine Fortführung der Arbeit gegeben.

1.3 Fortlaufendes Beispiel

Die Beschreibung der Grundlagen und Methoden dieser Arbeit ist auf ein Beispiel fokussiert, welches hier kurz vorgestellt wird. Es handelt sich um einen fiktiven Datensatz, der aus zwei wesentlichen Teilen besteht. Den Sensordaten, welche kontinuierlich sind, und den Metadaten welche die einzelnen Sensoren beschreiben.
Zunächst folgt eine Beschreibung des fiktiven Versuchsaufbaus. In einem abgeschlossenen Raum beliebiger Größe befinden sich eine festgelegte Anzahl i an Temperatursensoren. Diese Temperatursensoren senden in festgelegten Abständen von jeweils einer Zeiteinheit die von ihnen erhobenen Daten. In diesem Fall soll es sich um die Umgebungstemperatur in unmittelbarer Nähe des Sensors handeln. Jeder dieser Sensoren hat eine eindeutige Identifikationsnummer *(_sensorid) und sendet zu einem bestimmten Zeitpunkt* (_timestamp) *einen Wert* (_temperature)*. In Tabelle 1.1 sieht man diese Attribute in der Tabelle mit dem Namen Sensortabelle (data). Die rechte Tabelle (metadata)* enthält zusätzliche Daten zu jedem einzelnen Sensor. Die *metadata*-Tabelle enthält Informationen über den Hersteller*(_manufacturer)* des Sensors, die Position *(_position)* des Sensors im Raum, sowie eine dritte Information, die sich hier Isolation *(_isolation)* nennt. Sensoren mit verschiedenen Isolationen können für die gleiche Temperatur unterschiedliche Werte anzeigen. Zusätzlich können Sensoren verschiedener Hersteller verschiedene Werte anzeigen. Die Position im Raum, kann bei unterschiedlichen Wärmequellen von entscheidender Bedeutung sein. Zusätzlich sind die Sensoren über verschiedene Räume, die sich alle in einem Haus befinden, verteilt. Diese Erweiterung wird später dabei helfen, einige Konzepte zu veranschaulichen.

Das Schema der Relationen wird in Anfrage 1.1 in SQL-Notation angegeben. Die jeweiligen Attribute können die folgenden Werte annehmen:

1. *_sensorID* Die Identifikationsnummern sind natürliche Zahlen beginnend mit der Eins.
2. *_temperatur* Hier sind positive Fließkommazahlen erlaubt.
3. *_timestamp* Die Zeiteinheiten sind auch natürliche Zahlen.
4. *_id* Dieses Attribut steht in einer Beziehung zu sensorID und hat denselben Wertebereich.

Anfrage 1.1: Erstellung der Relationenschemata in PostgreSQL-Notation.

```
CREATE TABLE data (
  _sensorID integer,
  _temperature float,
  _timestamp integer
  );

CREATE TABLE metadata (
  _id integer PRIMARY KEY,
  _manufacturer varchar(1),
  _position varchar(10),
  _isolation boolean
  );
```

5. *_manufacturer* Es gibt vier verschiedene Hersteller, namentlich A, B, C und D.
6. *_position* Mögliche Positionen sind C1, C2, C3, ..., C17. Ein Zusammenhang zwischen den einzelnen Positionen wird im Verlauf der Arbeit hergestellt.
7. *_isolation* Entweder es gibt eine Isolation oder auch nicht, der Wertebereich ist also TRUE und FALSE.

data

_sensorID	_temperature	_timestamp
1	12,2	1
2	12,1	1
1	12,2	2
...	12,4	...
i	12,4	l

metadata

_ID	_manufacturer	_position	_isolation
S1	A	C1	TRUE
S2	C	C2	TRUE
S3	B	C1	FALSE
...	...	(x,y)	...

Tabelle 1.1: Tabellen zum fortlaufenden Beispiel.

Kapitel 2

Grundlagen

In nachfolgendem Abschnitt werden die grundlegenden Methoden und Definitionen erläutert, die in dieser Arbeit verwendet werden.

2.1 Das Relationenmodell

Grundlegend für diese Arbeit ist das Relationenmodell von Edgar F. Codd [12]. Die Definitionen sind sowohl aus dem Buch "The Theory of Relational Databases" von David Maier aus dem Jahr 1983 entnommen [29] sowie aus dem Lehrbuch "Datenbanken: Konzepte und Sprachen" von Andreas Heuer und Gunter Saake [33]. Auf Beweise wird weitgehend verzichtet und auf die entsprechende Quelle verwiesen.

Definition 1 (Relationenschema) *Ein Relationenschema R ist eine endliche Menge von Attributnamen $\{A_1, A_2, \ldots, A_n\}$. Zu jedem Attributnamen A_i existiert eine Menge $D_i, 1 \leq i \leq n$, die Domäne (oder Wertebereich) von A_i. Es gilt, der Wertebereich von A ist gleich $dom(A)$.*

Definition 2 (Relation (Instanz)) *Eine Relation r über $R = \{A_1, \ldots, A_n\}$ mit $n \in \mathbb{N}$ ist eine endliche Menge von Abbildungen.*

$$t : R \rightarrow \bigcup_{i=1}^{m} D_i$$

die Tupel genannt werden [33].

Definition 3 (Datenbankschema) *Es gilt $t(A) \in dom(A)$. Ein Datenbankschema ist eine Menge von Relationenschemata $S := \{R_1, \ldots, R_p\}$ mit $p \in \mathbb{N}$.*

In Tabelle 1.1 sieht man die Attributnamen der Tabellen *data* und *metadata* jeweils in der ersten Zeile. Dies sind die jeweiligen Relationenschemata

der Tabellen. Weiter sind die Wertebereiche oder Domänen bei der Erstellung der Tabellen festgelegt worden. Dies sieht man in Anfrage 1.1, wo für jeden Attributnamen ein Datentyp festgelegt wurde. Die Relation ist die Menge der Tupel, die in diese Tabellenspalten eingefügt wurden. Eine Relation nennen wir auch eine Instanz des Schemas.

Durch die Definition der Relation als eine Menge von Abbildungen wird verhindert, dass es eine explizite Reihenfolge der Attribute im Relationenschema gibt. Eine Festlegung der Reihenfolge führt nicht zu mehr Information innerhalb der Relation [29]. Ein Tupel ist demnach keine Sequenz von Werten in einer festgelegten Reihenfolge, sondern eine Menge an Werten, jeweils aus der Domäne der festgelegten Attribute des Relationenschemas. Der Attributwert w eines Tupels t an der Stelle A ist $w = t(A)$.

Definition 4 (Schlüssel) *Ein Schlüssel einer Relation r, in einem Relationenschema R ist eine Teilmenge $K = \{B_1, B_2, \ldots, B_m\}$ von R mit der folgenden Eigenschaft: Für zwei verschiedene Tupel t_1 und t_2 in r existiert ein $B \in K$ so dass $t_1(B) \neq t_2(B)$. [29]*

In den Tabellen des fortlaufenden Beispiels haben wir sowohl einen einzigen Schlüssel in der Tabelle *metadata* namentlich *_id*, als auch einen zusammengesetzten Schlüssel in der Tabelle *data*, nämlich {_sensorid, _timestamp}.

2.1.1 Relationale Operatoren

Um mit einer relationalen Datenbank arbeiten zu können, benötigt es Operationen, die auf ganzen Relationen verwendet werden. Die in dieser Arbeit verwendeten Operationen werden hier kurz erläutert.

Die Selektion Der *SELECT*-Operator ist eine einstellige Verknüpfung auf Relationen [29]. Wenn man diesen auf eine Relation anwendet, dann erhält man eine zweite Relation, die eine Teilmenge von Tupeln der ersten Relation ist. In [29] wird er wie folgt definiert:

Definition 5 (Selektion) *Sei r eine Relation auf dem Relationenschema R. A ein Attribut in R und a ein Element aus dom(A). Dann gilt: Die Selektion des Attributs A mit dem Selektionsprädikat $A = a$, $\sigma_{A=a}(r)$ ist die Relation*

$$r'(R) = \{t \in r | t(A) = a\}$$

Somit bekommen wir nach der Selektion eine Relation r' , die eine vollständige Teilmenge der Relation r ist. In der später ausschließlich verwendeten SQL-Notation, sieht diese Selektion folgendermaßen aus: “SELECT * FROM r WHERE A = a". Wir erhalten alle Tupel t aus r mit der Eigenschaft, dass alle Attribute A den Wert a haben.

Die Projektion Auch der Projektionsoperator ist eine einstellige Verknüpfung auf Relationen. Anstatt wie die Selektion eine Zeile in einer Relation zu wählen, wählt die Projektion eine Teilmenge von Spalten der Relation.

Definition 6 *Sei r eine Relation auf dem Relationenschema R und sei X eine Teilmenge von R. Die Projektion von r auf X, $\pi_X(r)$ ist die Relation $r'(X)$ ohne die Spalten $R - X$.*

Zusätzlich werden laut Maier et al. [29] in der verbleibenden Multimenge von Tupeln Duplikate eleminiert.

Mengenoperationen Wir können die üblichen Mengenoperationen, Vereinigung ($\cup$), Differenz ($-$) und Durchschnitt ($\cap$) auf Relationen anwenden.

Definition 7 (Mengenoperationen) *Diese Mengenoperationen sind für die Relationen $r_1(R_1)$ und $r_2(R_2)$ wie folgt definiert:*

$$r_1 \cup r_2 := \{t | t \in r_1 \vee t \in r_2\}$$

$$r_1 - r_2 := \{t | t \in r_1 \wedge t \notin r_2\}$$

$$r_1 \cap r_2 := \{t | t \in r_1 \wedge t \in r_2\}$$

Für diese Arbeit ist hier jedoch nur die Vereinigung interessant.

Natürliche Verbund Der *join*-Operator, oder der *natürliche Verbund* ist ein zweistelliger Operator, der auf zwei Relationen ausgeführt wird.

Definition 8 (natürlicher Verbund) *Sei r_1 eine Relation auf R_1 und r_2 eine Relation auf R_2, dann ist der natürliche Verbund:*

$$r_1 \bowtie r_2 := \{t | t(R_1 \cup R_2) \wedge [\forall i \in \{1, 2\} \exists t_i \in r_i : t_i = t(R_i)]\}$$

Im natürlichen Verbund werden 2 Relationen über denselben Attributwerte verbunden. Die Definition des natürlichen Verbundes wurde Heuer et al. [33] entnommen. Zusätzlich kennen die Datenbankforschung und deren Anwendung eine Vielzahl von weiteren Verbundarten. Diese werden in dieser Arbeit nicht verwendet. In der SQL-Notation stellt sich der natürliche Verbund wie folgt dar: “SELECT * FROM r1 NATURAL JOIN r2 on r1.ID = r2.ID”.

Aggregatfunktionen Anders als die vorhergehenden, skalaren Operationen, sind Aggregatfunktionen komplexe Operationen, die auf mehr als einem Tupel operieren können. Die folgenden Aggregatfunktionen werden in dieser Arbeit verwendet:

- COUNT() gibt die Anzahl der Zeilen eines Attributes zurück. COUNT(*) gibt zum Beispiel die Anzahl der Zeilen einer Relation wieder.
- SUM() gibt die Summe der Werte aus einer Spalte zurück. Dies funktioniert nur für Spalten, die einen numerischen Wertebereich haben.
- AVG() gibt das arithmetische Mittel von numerischen Werten einer Spalte zurück.

2.2 Klassifikation von Attributen

In dieser Arbeit benötigen wird eine Klassifikation der Attribute innerhalb der Relationen benötigt. Hierfür wird eine Erweiterung der Definition von E.Park und S. Yoon [32] verwendet. Eine Konzepthierarchie ist eine hierarchische, semantische Gliederung von Begriffen. Sie ist aufgebaut wie ein Baum und wird von der Wurzel zu den Blättern hin immer spezieller. Ein einfaches Beispiel einer Konzepthierarchie ist die Liste {Stadt, Haus, Zimmer}, wobei der Begriff Stadt die Wurzel des Baumes ist. Park et al. gehen davon aus, dass man keine Konzepthierarchie [32] aus einem Attribut erstellen kann, wenn das Attribut eine der folgenden Bedingungen erfüllt.
Fall 1: Es existiert eine hohe Anzahl an verschiedenen nichtnumerischen Werten, für die sich kein “sinnvolles” übergeordnetes Konzept finden lässt.
Fall 2: Es existiert eine hohe Anzahl an verschiedenen numerischen Werten, für die sich keine endliche Anzahl an “sinnvollen” Intervallen finden lässt.
Die Bedingung “sinnvoll” zu sein, wird durch einen Domain-Experten festgelegt, der auch für die Konstruktion der notwendigen Konzepthierarchie zuständig ist. Betrachten wir zum Beispiel ein Schlüsselattribut, welches nur die Tupel der Attribute identifiziert. Hier gibt es bei n Werten des Schlüssels n verschiedene Werte. Also eignet sich dieses Schlüsselattribut nicht für die Erstellung einer solchen Konzepthierarchie. Andere Schlüsselattribute eignen sich besser, so Beispielsweise ein Zeitstempel, der für jedes Tupel nur einmal auftaucht. Hier ist sowohl die Schlüsseleigenschaft gewährleistet, als auch die Möglichkeit, diese Werte zum Beispiel in die Konzepte “früh” und “spät” einzuteilen.

Für die Zwecke dieser Arbeit benötigen wir folgende Definition einer Konzepthierarchie.

Definition 9 (Konzepthierarchie) *Eine Konzepthierarchie ist ein Baum, der aus mindestens drei Ebenen besteht. Die erste Ebene, oder Wurzel genannt, enthält den allgemeinsten Begriff. Die Zwischenebenen müssen, semantisch, in einer verallgemeinernden Beziehung stehen. In den Blättern befinden sich jeweils die Werte der Konzepthierarchie. Jeder Vorgängerknoten eines Blattes, hat somit genau einen Nachfolger.*

2.3 Intensionale Antworten

Intensionale Antworten werden von Amihai Motro in "Intensional Answers to Database Queries" [31] zuerst beschrieben. Die Intension einer Datenbank ist die Definition der Struktur dieser Datenbank. Die Struktur der Datenbank nennen wir das Datenbankschema. Dieses Schema im Relationenmodell ist eine intensionale Information. Im Gegensatz dazu steht die Extension einer Datenbank. Die zu diesem Schema abgelegten Informationen, die Daten selbst, sind extensionale Informationen. Wenn wir von intensionalen Informationen sprechen, meinen wir die Struktur des Datenbankschemas, aber auch "die Definition der Basisrelationen, die Definition der Sichten und die Integritätsbedingungen [1]" [31]. Eine Sicht ist eine intensionale Information, da sie einen Rahmen vorgibt, welcher von extensionalen Informationen bevölkert werden kann. Bei Integritätsbedingungen sieht man, das hier keine Daten selbst gemeint sind, sondern Informationen über mögliche Daten. Man erkennt, dass intensionale Informationen nicht abhängig von extensionalen Informationen sind. Umgekehrt ist dies jedoch der Fall, denn eine extensionale Information benötigt eine Form. Um den Begriff der intensionalen Antwort erläutern zu können, betrachten wir zunächst die extensionale Antwort. Sie ist ein Ergebnis einer Anfrage an eine Datenbank. Extensional ist die Antwort deshalb, da sie konkrete Daten aus der Datenbank als Ergebnis liefert. Die intensionale Antwort liefert als Ergebnis nur eine Information über die Daten in der Datenbank und nicht die Daten selbst. Ein Beispiel aus der Mathematik soll dies deutlich machen.

Beispiel Die Frage lautet: Was ist eine natürliche Zahl? Die extensionale Antwort wäre die Aufzählung aller natürlichen Zahlen. Die intensionale Antwort wären die Peano-Axiome. Hier ist die Antwort eine Beschreibung der Eigenschaften der natürlichen Zahlen, statt nur eine Aufzählung aller natürlichen Zahlen. Übertragen auf eine relationale Datenbank, erkennt man, warum das Schema der Datenbank selbst eine intensionale Information sein muss. Zusätzlich kann es intensionale Informationen geben, die aus der extensionalen Antwort nicht zu erkennen sind. Im Beispiel mit den natürlichen Zahlen geht die Reihenfolge der Zahlen nicht aus der reinen extensionalen Antwort hervor. Die Ausgabereihenfolge kann muss aber nicht die Richtige sein. Die intensionale Antwort der Peano-Axiome gibt die Reihenfolge vor. Motro schreibt dazu in [31], dass die intensionale Information der Datenbank auch eine zusätzliche Charakterisierung der extensionalen Antwort beinhalten kann.

[1] "Specifically, the intension of a relational database includes the definitions of the base relations, the definitions of views, and the integrity constraints." [31]

2.4 Data Provenance

Der Begriff “Data Provenance” beschreibt den Ursprung eines Datums und den Prozess, wie es an seinen Ort gelangt ist [9]. Der Ursprung des Wortes, welches übersetzt Provenienz heißt, wird außerhalb der Informatik hauptsächlich in der Forschung über die Herkunft und von Kunstwerken und Kulturgütern verwendet. In dieser Arbeit wird aus Gründen der Abgrenzung und besseren Verständlichkeit, das englische Wort “Provenance” verwendet. Provenance beschreibt eine Reihe von Eigenschaften, des Datums in seinem Lebenszyklus. Zum Beispiel den Ursprung oder den Verarbeitungsgrad der Daten. Hier stellt sich die Frage, ob dieses spezielle Datum in seinem Lebenszyklus schon einmal verändert wurde und ob die ändernde Instanz vertrauenswürdig gewesen ist. Im Umfeld der relationalen Datenbanken benutzen wir die verschiedenen Provenance Techniken, um mehr über ein erhaltenes Ergebnis zu erfahren. Nach einer gestellten Anfrage an die Datenbank erhalten wir das Anfrageergebnis, welches aus keinem oder mehreren Tupeln bestehen kann. Eine Provenance-Anfrage ist dann beispielsweise, aus welchen Tupeln in der Ursprungsdatenmenge dieses Antworttupel entstanden ist. In der Forschung werden verschiedene Arten von Provenance unterschieden. Eine genaue Beschreibung dieser Formen folgt in Abschnitt 3.1.

2.5 Big Data Analytics

Das Feld der Big Data Analytics (BDA) hat sich in den letzten Jahren massiv entwickelt. Eine Vielzahl von neuen Datenquellen, sowohl im wissenschaftlichen Bereich als auch im Geschäftsumfeld, führen zu neuen Herausforderungen und Chancen im Bereich der Datenverarbeitung. Labrandis et al. beschreiben in ihrem Community Whitepaper “Challenges and Opportunities with Big Data” [27] die Auswirkung auf die wissenschaftliche Gemeinschaft, und die Geschäftswelt. Die Möglichkeiten dieser Technik ergeben aber wieder neue Probleme. Auch in der Datenbankforschung und insbesondere in der Provenance-Forschung ist dies ein aktuelles Thema [15].

In dem Whitepaper von Oracle [13] wird Big Data als in einen der folgenden drei Teilbereiche gehörend definiert.

1. Klassische Geschäftsdaten. Kundendaten aus CRM-Systemen, Transaktionsdaten aus ERP-Systemen und Buchhaltungsdaten.
2. Sensordaten und andere maschinell erstellte Daten.
3. Social data, also Daten aus sozialen Netzwerken.

Von den drei Datenquellen wird in dieser Arbeit hauptsächlich die zweite betrachtet. Weiter gibt es vier Hauptcharakteristika von Big Data, welche im folgenden kurz besprochen werden[13].

- Volume. Maschinell erstellte Daten können in wenigen Minuten Terabyteweise Daten erstellen. Ein Hauptbestandteil von Big-Data ist also

die schiere Masse an Daten, die es zu bearbeiten gibt.

- Velocity. Nicht nur die Masse der Daten ist relevant, es kommt auch auf die eintreffende Geschwindigkeit an. Sensordaten lassen sich vielleicht auf wenige Byte zusammenfassen und übertragen. Allerdings ist die Frequenz der Dateneingänge möglicherweise so hoch, dass sich am Ende trotzdem große Datenmengen akkumulieren.
- Variety. Im Gegensatz zu traditionellen Datenformaten, welche auf einem zuvor festgelegten Schema basieren, hat man es in der Big Data Analytics (BDA) mit vielen verschiedenen Datenquellen zu tun. Hier ist also eine große Heterogenität der Daten das Problem.
- Veracity. Die Vertrauenswürdigkeit der Daten, kann nicht immer sichergestellt werden. Es ist wichtig, dass die Herkunft der Daten geprüft werden kann.

Im Rahmen dieser Arbeit wird Big Data, also die Datenquelle als Strom von Daten betrachtet. Es geht demnach um Daten die kontinuierlich eintreffen. Das Speichern dieser Daten ist nicht unbedingt möglich und auch nicht notwendig. Im diesem Sinne betrachten wir heterogene Datenquellen, die in einer hohen Geschwindigkeit eine große Zahl an Datensätzen zur Verfügung stellen.

Die Herausforderungen, die durch die BDA entstehen, beschreiben Kersten et al. in ihrem Whitepaper “The Researcher’s Guide to Data Deluge: Querying a Scientific Database in Just a few Seconds” [26]. Sie beschreiben den Paradigmenwechsel von der Geschäftsdatenbank, in der es auf Korrektheit und Vollständigkeit ankommt, hin zu einem Datensatz, in dem der Benutzer nicht mehr genau weiß, was darin zu finden ist. Hier steht die Exploration der Daten im Vordergrund [26]. Der Wissenschaftler oder der Analyst kennt die genaue Datenlage nicht mehr und versucht sich in der großen Datenmenge zurechtzufinden. Hieraus ergeben sich einige Herausforderungen, die in traditionellen Datenbanken schon weitgehend gut erforscht sind. Ein entscheidender Aspekt ist dabei auch wieder die *Provenance* der Daten.

Kapitel 3

Stand der Forschung

Dieses Kapitel beschäftigt sich mit dem aktuellen Stand der Forschung in den Bereichen Provenance und den intensionalen Antworten.

3.1 Provenance

Boris Glavic unterscheidet in [14] drei verschiedene Typen von Provenance-Informationen:

- Daten: Welche Daten wurden verwendet um eine Antwort zu erhalten.
- Transformation: Welche Transformationen wurden verwendet um die Antwort zu erhalten?
- Agents, Auxilliary and Environment: In dieser Kategorie sind Informationen wie der Benutzer, der das Datum erstellt hat, oder die Maschine mit der das Ergebnis berechnet wurde.

Je nach Anwendungsbereich sind die Typen von unterschiedlicher Wichtigkeit. Gerade bei wissenschaftlichen Datenbanken ist eine Verifizierung der Daten von entscheidender Bedeutung. Lynch [28] behauptet, dass dies bei Onlinedaten nicht mehr der Fall sein kann. Im Umfeld von Big Data kann also nicht immer von einer sicheren Quelle ausgegangen werden. Im folgenden werden die verschiedenen Arten von Provenance beschrieben.

3.1.1 Why-Provenance

Die sogenannte *Why-Provenance* wurde eingeführt von Peter Buneman, Sanjeev Khanna und Wang-Chiew Tan in dem Artikel: “Why and where: A characterization of data provenance” aus dem Jahre 2001 [9]. Es geht um die Frage, warum sich ein bestimmtes Tupel in der Ergebnisrelation befindet. Um diese Frage beantworten zu können, muss die Information aus den Ursprungstabellen mit in die Ergebnisrelation übernommen werden. Jedes Ergebnistupel trägt also die Information seiner Herkunft mit sich. Dies kann

für jedes Tupel bedeuten, dass die vollständige Ursprungstabelle mitgenommen wird. Buneman et al. bezeichnen diese Annotationen als *wittnesses* des Ergebnistupels. Durch diese Methode lässt sich für jedes Tupel im Anfrageergebnis eindeutig sagen, aus welchen Ursprungsdaten es entstanden ist. Die Definition der *Why-Provenance* in dieser Arbeit, orientiert sich an der Arbeit von Boris Glavic, der in [24] eine eigene Semantik, die "Perm Influence Contribution Sematics" kurz PI-CS vorstellt. Sowie an dem Artikel "Provenance in Databases: Why, How and Where" [11] indem die *Why-Provenance* auf das relationale Modell einer Datenbankanfragesprache die ausschließlich Selektionen, Projektionen, Joins und Unions (SPJU) angewendet wird.

Definition 10 (Witness) *Sei r ein Relation , eine Anfrage Q und ein Tupel $t \in Q(r)$. Ein* witness*, für t in Bezug auf Q, ist dann eine Relation $r' \subseteq r$, so das gilt $t \in Q(r')$.*

Es ist klar, dass jedes Tupel aus R, das an der Entstehung von t beteiligt war, ein *witness* für t ist. Die Menge an *witnesses* für t bezüglich der Relation r und der Anfrage Q ist also:

$$Wit(Q, r, t) = \{l \subseteq r | t \in Q(l)\}$$

$Wit(Q, r, t)$ ist zwar eine endliche Menge an Tupeln, kann aber exponentiell groß werden, da viele "irrelevante" Tupel Teil des *witness* sein können[11]. Cheney et al. gehen davon aus, dass nur monotone Anfragen verwendet werden (SPJU), somit folgt daraus, dass $Wit(Q, r, t) = \emptyset$ wenn $t \notin Q(r)$ ist. Die Menge der Zeugen (*witnesses*) ist also nur leer, wenn das Tupel t nie in der Ursprungsrelation r gewesen ist. Das Problem des eventuellen exponentiellen Wachstums der *witnesses* wird durch die Einführung der *witness-basis* gelöst. Diese *witness-basis* ist nicht größer als die Ursprungsrelation selbst und erfüllt alle nötigen Eigenschaften und repräsentiert die notwendigen Eigenschaften an eine Menge von *witnesses* wie Cheney et al. in [11] zeigen. In dieser Arbeit bedienen wir uns der schon ausgearbeiteten Software PERM von Boris Glavic [17].

Beispiel Um die *Why-Provenance* zu verdeutlichen, wird diese am fortlaufenden Beispiel erklärt. Wir gehen von den folgenden zwei Schemata aus.

$$data := \{\_sensorID, \_temperatur, \_timestamp\}$$

$$metadata := \{\_ID, \_manufacturer, \_position, \_isolation\}$$

Eine mögliche Anfrage an die Beispielrelation wäre die Anfrage 3.1. Gesucht ist die Durchschnittstemperatur der Sensoren, abhängig von Position, Hersteller und Zeitpunkt.

data			Tupel#
_sensorID	_temperatur	_timestamp	
1	12,2	5	t_1
2	12,1	4	t_2
3	12,2	8	t_3
4	12,4	1	t_4
5	12,4	9	t_5
6	12,4	3	t_6

Tabelle 3.1: Ursprungstabelle “data” mit nummerierten Tupeln

metadata				Tupel#
_ID	_manuf	_pos	_iso	
S1	A	C1	TRUE	t_7
S2	C	C2	TRUE	t_8
S3	B	C1	FALSE	t_9
S4	A	C2	TRUE	t_{10}
S5	B	C3	TRUE	t_{11}
S6	D	C4	TRUE	t_{12}

Tabelle 3.2: Ursprungstabelle “metadata” mit nummerierten Tupeln

Die Anfrage 3.1 führt zu der Antwort $R1$ “12,5”. Dies ist die Durchschnittstemperatur dieser Sensoren. Nun kommen wir zur *Why-Provenance*. Die Antwort auf die Frage, welche Tupel an der Erstellung von $R1$ beteiligt waren kann man in Tabelle 3.3 sehen. Diese Tupel bezeichnen wir auch als die *Witnesslist* der Antwort $R1$. Denn sie bezeugt die Herkunft des Ergebnisses. Jedes Tupel ist ein *witness* des Ergebnisses $R1$. Die *Witnesslist* ist dann: $\{\{t_1\}\{t_2\}\{t_3\}\{t_5\}\{t_7\}\{t_8\}\{t_9\}\{t_{11}\}\}$

Anfrage 3.1: Beispielanfrage zur Provenance

```
1    SELECT AVG(_temperatur)
2    FROM data as ST
3    JOIN  metadata as MT
4    ON ST._sensorID = MT_.ID
5    WHERE MT._position='C1'
6    AND MT._manufacturer = 'A'
7    AND ST._timestamp BETWEEN 4 and 9;
8
```

Witnesslist

_sensorID	_temperature	_timestamp	_manuf	_pos	_iso
1	12,2	5	A	C1	TRUE
2	12,1	4	A	C1	TRUE
3	12,2	8	A	C1	FALSE
5	12,4	9	A	C1	FALSE
6	12,4	3	A	C1	FALSE

Tabelle 3.3: Ergebnis der Why-Provenance

3.1.2 How-Provenance

Die soeben beschriebene *Why-Provenance* gibt uns eine Menge an *witnesses* für ein gegebenes Ergebnistupel. Allerdings bekommt man keine Informationen darüber, wie aus diesen Ursprungstupeln das Ergebnistupel geworden ist und welche Operationen auf dem Tupel ausgeführt wurden. Zuerst ist diese sogenannte *How-Provenance* von Green et al. [18] beschrieben worden.

In Anfrage 3.1 ist es durch die Attribute in den Selektionsprädikaten (*_position*, *_manufacturer* und *_timestamp*) offensichtlich, auf welchem Weg das Ergebnis $R1$ zustande gekommen ist. Jedes Tupel t1, t2, ... ,t6 aus der *witness* - List in Tabelle 3.3 wurde einmal in der Aggregation AVG() verwendet. Wenn wir die Anfrage 3.1 zu Anfrage 3.2 erweitern, dann ist die Antwort nicht mehr so offensichtlich.

Anfrage 3.2: Beispielanfrage 2

```
1    SELECT AVG(_temperature)
2    FROM (
```

```
3     SELECT _sensorID, ROUND(_temperature,0) as Data,
      _timestamp
4     FROM data
5     ) as ST
6     JOIN  metadata as MT ON ST._sensorID = MT._ID
7     WHERE MT._position='C1'
8     AND MT._manufacturer BETWEEN 4 and 9;
9
```

Durch die versteckte ROUND()-Funktion werden Daten verändert. Die *How-Provenance* liefert dann als Ergebnis, dass die Tupel aus der Sensortabelle, anders als in der ersten Anfrage, gleich zweimal verarbeitet wurden. In der Notation von [18] ist die Ausgabe der *How-Provenance* für die Tupel aus den Ursprungstabellen folgende:

$$AVG(\_temperature) = t_1^2 \cdot t_6 + t_2^2 \cdot t_7 + \ldots + t_5^2 \cdot t_6$$

Dies bedeutet, dass jedes Ursprungstupel t_1, t_2, t_3, t_4, t_5 zweimal verarbeitet wurde und jeweils mit seinem Verbundpartner gejoint wurde. Die Verbundpartner t_6, t_7 wurden jeweils nur einmal verarbeitet . In der Notation stellt ein (.) einen JOIN, und ein (+) eine UNION der Tupel dar. Diese Semantik der *How-Provenance* basiert auf sogenannten *Provenance-polynomials*, welche einen Hauptbestandteil des von Green et al. entwickelten Frameworks [18] bilden.

3.1.3 Where-Provenance

In diesem Abschnitt wird die von Buneman et al. [9] eingeführte Idee der *Where-Provenance* erläutert. Im Vergleich zur *Why-Provenance* und *How-Provenance* die uns Ursprungstupel und die Transformation der Tupel liefern, liefert die *Where-Provenance* die Position, aus der ein Wert in das Ergebnis herauskopiert wurde. Cheney et al. beschreiben dies wie folgt "Während *Why-Provenance* und die Beziehung zwischen Eingabe- und Ausgabetupeln beschreiben, beschreibt die where-Provenance die Beziehung zwischen Eingabe- und Ausgabe-Orten. [1]" [11]. Die *Where-Provenance* beschreibt die Tabellen, aus denen die Antwort herauskopiert wurde. Es ist also nicht das Eingabetupel ausschlaggebend, wie in der *Why-Provenance*, sondern es wird eine Beziehung zwischen Eingabeort (Tabelle) und dem Ergebnis hergestellt. Sei d eine Relation und $Q(d)$ eine Anfrage an diese Relation. Die *Where-Provenance* eines Wertes, der sich an einem Ort l in $Q(d)$ befindet, besteht aus Orten die sich in d befinden und aus denen der Wert an den Ort l kopiert wurde. [11]

[1] While why-provenance is about the relationship between source and output tuples, where-provenance describes the relationship between source and output locations

Beispiel In Tabelle 3.4 ist die bekannte *metadata*-Tabelle in zwei Tabellen aufgespalten worden. Die *Where-Provenance* beantwortet die Frage, aus welcher der beiden Tabellen ein Wert mit dem Attributnamen _ID herauskopiert wurde.

metadata

_ID	_manufacturer	_position
S1	A	C1
S2	C	C2
S3	B	C1
...	...	(x,y)

metadata2

_ID	_isolation
S1	TRUE
S2	TRUE
S3	FALSE
...	...

Tabelle 3.4: Beispiel zur *Where-Provenance*

3.1.4 Why-Not-Provenance

Die *Why-Not-Provenance* beschäftigt sich mit der Frage, warum ein erwartetes Tupel nicht im Anfrageergebnis steht. Eine Antwort wäre, dass das Selektionsprädikat in diesem Fall zu beschränkend auf den Ursprungsdatensatz gewirkt hat und dadurch das erwartete Tupel ausgeschlossen hat. Mithilfe der *Why-Not-Provenance* lässt sich dieses Selektionsprädikat isolieren. Im Idealfall schlägt das System eine erweiterte Selektionsbedingung vor, sodass das Tupel in der Antwort auftaucht. Beschrieben wird diese Art der Provenance von Chapman et al. [10] in dem Artikel “Why Not?”. Die vom Anwender erwartete, aber nicht im Anfrageergebnis stehende Antwort, wird als *missing answer* bezeichnet [21]. In der Literatur gibt es dazu drei verschiedene Ansätze. Die *instance-based explanations* von Herschel et al. [21] und Huang et al. [22], *query-based explanations* von Chapman et al. [10], sowie die *modification-based explanations* von Tran und Chan [34]. Die instanzbasierte Erklärung (*instance-based explanations*) gibt dem Anwender den Hinweis, dass die Daten in der aktuellen Relation (Instanz) nicht der von ihm gestellten Anfrage entsprechen. Die Erklärung hat damit ihren Ursprung in der Relation. Die anfragebasierte Erklärung (*query-based explanations*), geht von der gestellten Anfrage aus. Sie erkennt, dass es das Tupel in der Relation gibt, die gestellte Anfrage dieses aber nicht einschließt. Eine modifikationsbasierte Erklärung(*modification-based explanations*) passt die Anfrage dann in dem Fall an. Eine Anpassung ist die Modifikation der Selektionsbedingungen durch eine Erweiterung der Parameter. Instanzbasierte Erklärungen sind extensionale Daten, hingegen sind modifikations- und an-

fragebasierte Erklärungen intensional.

Beispiel Im laufenden Beispiel stellen sich die Erklärungen für die fehlenden Antworten wie folgt dar. Wir betrachten die Anfrage 3.3, die auf den Relationen aus Tabelle 3.5 ausgeführt wird.

Anfrage 3.3: Beispiel missing answer

```
1    SELECT _sensorID, _temperature, _manufacturer
2     FROM data join metadata
3     WHERE _temperature < 13
4     AND _position = 'C3';
5
```

Das Ergebnis der Anfrage ist leer. Eine Frage der *Why-Not-Provenance* wäre, warum das Tupel $< 3; 13, 4; C3 >$ nicht im Anfrageergebnis ist. Eine instanzbasierte Erklärung wäre der Hinweis des Systems, dass ein in Frage kommendes Tupel die Eigenschaft $\_temperature < 13$ nicht erfüllt. Die *missing answer* wird dann angegeben. Diese ist dann ein Tupel das alle anderen Eigenschaften erfüllt. $< 3; \_temperature; \_temperature > 13; C3 >$ Die anfragebasierte Erklärung hingegen weist darauf hin, das zwar ein Tupel mit einigen gestellten Eigenschaften existiert, die Anfrage aber zu genau gestellt ist. Ein Hinweis auf das Selektionsprädikat "$\_temperature < 13$" wird gegeben. Die modifikationsbasierte Erklärung ändert die Anfrage gleich dahingehend ab, dass das gewünschte Tupel im Ergebnis erscheint. Beispielsweise durch die Änderung "$\_temperature < 14$".

data

_sensorID	_temperature	_timestamp
1	12,2	1
2	12,1	1
1	12,2	2
3	13,4	1
5	12,4	1

metadata

_ID	_manufacturer	_position	_isolation
S1	A	C1	TRUE
S2	C	C2	TRUE
S3	B	C3	FALSE
S5	A	C4	FALSE

Tabelle 3.5: Beispiel zur *missing answer*

3.1.5 Big Data Provenance

Im bereits erwähnten Community Whitepaper [27] wird die Verfolgung der Herkunft (Provenance) der Daten als eine wichtige Anforderung in Big-Data-Anwendungen identifiziert. In "Big Data Provenance: Challenges and Implications for Benchmarking" bezeichnet Boris Glavic es als "unmöglich für einen Anwender, die Relevanz von Daten und ihre Qualität zu beurteilen, sowie unerwartete und falsche Ergebnisse zu ermitteln[2]"[15]. Ikeda et al. [23] stellen eine Methode vor, um Provenance in MapReduce-Aufgaben zu verfolgen. Von Malik et al. gibt es einen ähnlichen Ansatz, mit dem die Provenance von Antworten über verteilte Anwendungen verfolgt werden kann [30].

3.2 Intensionale Antworten

3.2.1 Generierung intensionaler Antworten

In der von E.Park et al. beschriebenen Methode werden intensionale Antworten aus extensionalen Daten generiert [32]. Park et al. verwenden dafür sogenannte Konzepthierarchien. Diese müssen zuvor von einem Domain-Experten erstellt werden. Sie bieten die Orientierung für das System. Ein kurzes Beispiel aus der Literatur soll dies verdeutlichen.

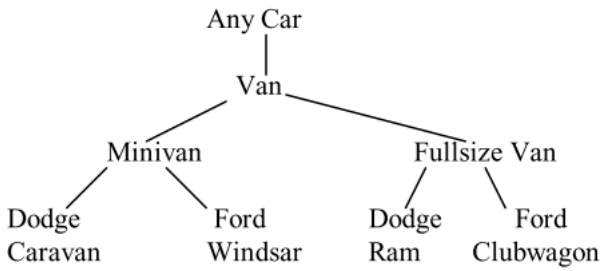

Abbildung 3.1: Beispiel Konzepthierarchie

Beispiel Das Attributmodell kann wie in Abbildung 3.1 einfach in eine Konzepthierarchie gebracht werden. Man kann sehen, dass sich die Autos aus Tabelle 3.6 einfach in eine Konzepthierarchie bringen lassen. Hingegen gibt es für die Käufernamen keine geeignete Methode, dies zu tun.

Hier wird der von Park et al. [32] verwendete Algorithmus beschrieben. Dieser basiert auf der Arbeit "Data-Driven Discovery of Quantitative Rules

[2] "Without provenance information, it is impossible for a user to understand the relevance of data, to estimate its quality, and to investigate unexpected or erroneous results."

Käufer	Auto
PersonA	Dodge Caravan
PersonB	Ford Windsar
PersonC	Dodge Ram
PersonD	Ford Clubw.

Tabelle 3.6: Beispieltabelle zur Konzepthierarchie

in Relational Databases" von Han et al. [25].
Die Methode wird eingeteilt in drei Phasen: Die *Preprocessing Phase*, die *Query Execution Phase* und die *Answer Generating Phase*. Die drei Phasen werden zuerst allgemein und anschließend an einem Beispiel verdeutlicht.

Preprocessing Phase Zunächst wird für jedes Attribut des Datenbankschemas eine Konzepthierarchie erstellt. Wie im vorherigen Beispiel gesehen, ist dies nicht immer möglich. Park et al. geben dazu zwei Fälle vor.

- *1. Fall:* Eine große Menge von nicht numerischen Werten, für die kein sinnvolles hierarchisches Konzept existiert.
- *2. Fall:* Eine große Menge von numerischen Werten, aber eine endliche Anzahl an sinnvollen Intervallen, basierend auf der Verteilung der Werte.

Wie im Beispiel zu sehen, werden Attributwerte benötigt, die sich abstrahieren lassen. Für das Attribut "Auto" ist das der Fall, für das Attribut "Name" nicht. Im Allgemeinen kann man sagen, dass sich Schlüsselattribute nicht in eine Konzepthierarchie bringen lassen [32].

Query Execution Phase Die Phase wird von Park et al. in drei Schritte geteilt. Erstens wird die extensionale Antwort mit Hilfe der gewünschten Anfrage generiert[32]. Dies passiert mit all den bekannten Mitteln, die ein DBMS zu bieten hat, das heißt auch unter Zuhilfenahme von Anfrageoptimierungen.
Als zweites werden alle Attribute eliminiert, die in der gewünschten Fragestellung keinen Mehrwert bringen. Das betrifft alle Schlüsselattribute und alle Attribute ohne korrespondierende Konzepthierarchie.
Als letztes werden alle Attribute entfernt, die keine sinnvolle Korrelation haben. Es liegt in der Hand des Domain-Experten, hier die richtigen oder

wichtigen Attribute zu benennen.

Answer Generation Phase In dieser Phase werden die verbliebenen extensionalen Daten verarbeitet, um am Schluss der Phase eine intensionale Antwort zu erhalten. Die Phase findet in drei Schritten statt:

- Schritt 1: Jeder Wert wird in Richtung seines höherliegenden Konzepts generalisiert. Der Wert wird durch das Konzept ersetzt.
- Schritt 2: Es wird geprüft, ob es ein Konzept gibt, das alle Werte eines Attributes ersetzt. Wenn das der Fall ist, wird der Prozess für dieses Attribut beendet. Falls das nicht der Fall ist, dann wird das darüberliegende Konzept gewählt.
- Schritt 3: Wiederhole Schritt 2, bis alle Werte durch Konzepte ersetzt wurden.

Zusätzlich schlagen Park et al. vor, eine Schwelle einzurichten, bis zu welcher die Konzepte ersetzt werden sollen. Die Vorgehensweise soll an einem Beispiel erläutert werden. Die Tabelle 3.7 ist die Grundlage für dieses Beispiel. Die Attribute Name, Type-of-car, Age und Income beschreiben verschiedene Personen, das Auto, das sie bevorzugen, ihr Alter sowie ihr Einkommen.

Name	Type-of-car	Age	Income
Miller	sports	28	58000
Fisher	mini van	37	45000
Johnson	sports	33	60000
...	...	....	....
Smith	truck	58	70000
Wallace	sports	30	65000

Tabelle 3.7: Beispiel Konzepthierarchie

Anfrage 3.4: Beispiel Anfrage

```
1    SELECT * FROM autokaeufer WHERE Type-of-car = 'sports';
2
```

Name	Type-of-car	Age	Income
Miller	sports	28	58000
Johnson	sports	33	60000
Wallace	sports	30	65000

Tabelle 3.8: Resultset für die Anfrage 3.4

Wir wollen nun wissen, wer die typischen Käufer von Sportwagen sind. Die dazugehörige Anfrage ist zu sehen in Anfrage 3.4. Wir erhalten als Anfrageergebnis $R1$ aus Tabelle 3.7. Alle weiteren Schritte beziehen sich auf die Tabelle 3.7. Weiter werden alle Attribute ausgewählt, die sich in eine Konzepthierarchie einbetten lassen. Dies sind in diesem Fall die Attribute “income” und “age”. Mögliche Konzepthierarchien für die Attribute sieht man in Abbildung 3.2

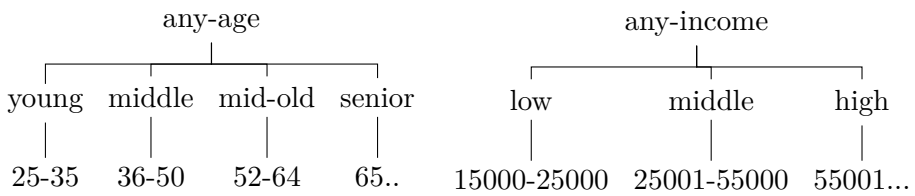

Abbildung 3.2: Hierarchieebenen des Datenmodells

Nun kann man mit der Generalisierung der Werte beginnen. Es ist zu erkennen, dass die Spalte “age” durch das Konzept “young” ersetzen werden kann und die Spalte “income” durch das Konzept “high”. Dadurch erhält man die Tabelle 3.9.

Zuletzt kann man die Zeilen der Tabelle 3.9 zusammenfassen und erhält eine einzige Spalte, aus der sich direkt die intensionale Antwort ablesen lässt. Diese lautet: “Junge Menschen mit hohem Einkommen kaufen Sportwagen.” Diese Methode wird später dazu verwendet, eine intensionale Antwort aus der Provenance-Anfrage zu generieren.

Type-of-car	Age	Income
sports	young	high
sports	young	high
sports	young	high

Tabelle 3.9: Tabelle mit den, durch die Konzepthierarchie, ersetzen Werten

3.2.2 Klassifikation von intensionalen Antworten nach Motro

Amihai Motro [31] unterscheidet grundsätzlich vier Klassen von intensionalen Antworten, die hier kurz erläutert werden.

1. Datenmodell und intensionale Information sind vorhanden,
2. extensionale Information innerhalb einer intensionalen Antwort,
3. Vollständigkeit der intensionalen Charakterisierung und
4. Unabhängigkeit von extensionalen Informationen.

Der erste Aspekt geht davon aus, dass das Datenmodell und die intensionalen Informationen wie zum Beispiel die Integritätsbedingungen vorhanden sind und verwendet werden. Der zweite Aspekt unterscheidet zwischen reinen intensionalen Informationen und intensionalen Informationen die noch durch extensionale Informationen angereichert sind. Eine solche gemischte intensionale Antwort wäre beim Beispiel mit den natürlichen Zahlen, eine Definition der natürlichen Zahlen ab der Eins. Zusätzlich dazu käme die extensionale Antwort Null. Die daraus zusammengesetzte gemischte intensionale Antwort wären die natürlichen Zahlen von Null an. Der dritte Aspekt unterscheidet zwischen intensionalen Antworten, die eine vollständige Charakterisierung der extensionalen Antwort sind und solchen, die nur eine partielle Charakterisierung der extensionalen Antwort sind. Eine vollständige Charakterisierung wäre eine alternative Spezifikation der Anfrage, wohingegen eine partielle Charakterisierung nur zusätzliche Informationen zur extensionalen Antwort aufweist [31]. Die Aussage: 'Die natürlichen Zahlen sind eine Teilmenge der ganzen Zahlen' wäre eine solche partielle Charakterisierung der natürlichen Zahlen ohne einfach nur die Definition zu wiederholen. Der vierte Aspekt unterscheidet zwischen intensionalen Antworten, die abhängig bzw. unabhängig von extensionalen Informationen sind. Gemischte intensionale Antworten wie im ersten Aspekt sind natürlich abhängig.

3.2.3 Bewertung intensionaler Antworten nach Motro

Motro schlägt Kriterien vor, nach denen sich eine intensionale Antwort bewerten lässt. Diese sind Vollständigkeit, Nichtredundanz, Optimalität, Relevanz sowie Effizienz [31] . Motro definiert diese Kriterien wie folgt.

- **Vollständigkeit:** "Eine Methode ist vollständig, wenn sie alle existierenden intensionalen Antworten entdeckt" [31]. Hier muss explizit zwischen einer vollständigen intensionalen Antwort und einer vollständigen Methode zur Erstellung von intensionalen Antworten unterschieden werden. Erstere ist die Anfrage selbst.
- **Nichtredundanz:** Die Wiedergabe der Anfrage wäre eine redundante Information, die intensionale Antwort soll eine zusätzliche Information enthalten. Dadurch wird aus dem ersten Aspekt der Vollständigkeit die Forderung alle nichtredundanten Antworten zu finden.
- **Optimalität:** Hier wird eine Metrik vorgeschlagen, die verschiedene intensionale Antworten bewerten kann. Eine Methode ist dann optimal, wenn sie nach der Metrik das beste Ergebnis liefert. Motro schlägt vor, die Optimalität nach Präzision und Prägnanz zu bemessen.
- **Relevanz:** Die Antwort soll einen Bezug zur gestellten Frage haben. Motro schreibt dazu: "Die Frage der Relevanz ist eine der schwersten Herausforderungen für die Effektivität von intensionalen Antworten" [31].
- **Effektivität:** Diese misst die Kosten der Erstellung der intensionalen Antwort aus der extensionalen Information.

Am Schluss dieser Arbeit soll die vorgestellte Methode nach eben diesen Kriterien bewertet werden.

Kapitel 4

Stand der Technik

In diesem Abschnitt werden die in dieser Arbeit verwendete Software zur Ermittlung der Provenance sowie einige Alternativen genannt.

4.1 PERM Provenance Extension of the Relational Model

PERM wurde von Boris Glavic [24] entwickelt. Die Grundidee von PERM ist, dass die Provenance-Information zusammen mit dem Ergebnis der Anfrage in einer Tabelle sichtbar wird. Hierbei verlässt er sich auf sogenannte *query rewrites*, um dieses Ziel zu erreichen. PERM ist eine Erweiterung einer PostgreSQL-Distribution und wird zur Zeit in der Version 0.1 bereitgestellt. Glavic, Miller und Alonso beschreiben in vier zentrale Forderungen an ein Provenance-System welche hier kurz skizziert werden sollen.

1. **Anforderung 1.** Verschiedene Arten der Provenance mit einer korrekten Semantik. Eine Semantik wird als korrekt betrachtet, wenn sie nachgewiesenermaßen das intuitive Verständnis von Provenance wiedergibt [17].
2. **Anforderung 2.** Um eine sinnvolle Verwendung finden zu können, muss das System auch komplexere SQL-Funktionen nutzen können, zum Beispiel *nested subqueries* und *Aggregate*.
3. **Anforderung 3.** Die Provenance-Informationen müssen selbst durch komplexe SQL-Anfragen bearbeitet werden können.
4. **Anforderung 4.** Das Provenance-System soll nur die notwendigen Provenance-Informationen liefern. Die vollständigen Provenance-Informationen können die Größe der Datenbank leicht übersteigen [9].

PERM stellt die Provenance-Information als Relation dar, diese wird bei Bedarf direkt mit Standard SQL-Befehlen generiert [17]. Der Benutzer stellt eine Anfrage Q in der SQL-PLE-Erweiterung an das System. Die Antwort enthält sowohl die Provenance von Q als auch die normalen

Anfrage 4.1: Why-Provenance

```
1  SELECT PROVENANCE ON CONTRIBUTION (INFLUENCE)
2  md._position,AVG(da._temperature)
3  FROM data da  JOIN metadata md ON da._sensorid = md._id
4  GROUP BY md._position, da._temperature
5
```

Anfrage 4.2: Beispiel PERM

```
1  SELECT * FROM
2  (SELECT PROVENANCE AVG(_temperature)
3  FROM data) as prov;
4
5
```

Ergebnisse der Anfrage Q. PERM ist eine Erweiterung einer PostgreSQL-Datenbank Implementierung und erweitert den PostgreSQL-SQL-Diaklekt um Provenance-Anteile. Wenn diese Provenance-Erweiterungen nicht verwendet werden, verhält sich das System wie eine gewöhnliche PostgresSQL-Datenbank. Das beinhaltet auch, dass die Provenance-Informationen nur auf Bedarf (lazy evaluation) berechnet werden. Die konkrete Beschreibung der einzelnen Funktionen befindet sich im folgenden Abschnitt.

SQL Language Extension SQL-PLE

PERM verwendet den SQL-Dialekt SQL-PLE, mit welchem der Nutzer Provenance-Anfragen an die Datenbank stellen kann. Mit dem Schlüsselwort `PROVENANCE` wird PERM angewiesen, die Provenance einer Anfrage auszugeben. Zusätzlich lässt sich durch den Zusatz `ON CONTRIBUTION` eine spezifische Provenance Berechnung spezifizieren. Für die *Why-Provenance* ist das Schlüsselwort `INFLUENCE` zu wählen, bei der *Where-Provenance* das Schlüsselwort `COPY`. In Anfrage 4.1 ist die Syntax für die *Why-Provenance* zu sehen.

Zusätzlich bleiben alle Funktionen von PostgreSQL erhalten und werden durch die PERM-Erweiterung nicht beeinflusst. Die Ergebnisse der Provenance-Berechnung werden in einer normalen Tabelle ausgegeben. Der Nutzer hat die Möglichkeit, andere SQL-Anfragen auf diesem Ergebnis auszuführen. Dies wird in Anfrage 4.2 gezeigt. Zusätzlich lässt sich die Berechnung der Provenance in Teilen der Anfrage durch das Schlüsselwort BASERELATION deaktivieren [16].

4.2 Andere Provenance-Frameworks

An dieser Stelle werden einige Arbeiten vorgestellt, die Frameworks bereitstellen, mit denen sich einige Formen der Provenance automatisch darstellen lassen. Dieser kurze Überblick basiert auf den Arbeiten von Herschel et al. [20] und Glavic et al. [17].

Why-Not Chapman und Jagadish [10] zeigen einen Algorithmus zur Berechnung von *missing answers* mit Hilfe von *query-based explanations*. Er ermittelt Tupel in der Ursprungsrelation, welche die Anforderungen der *missing answer* erfüllen und sich nicht in der Provenance-Berechnung eines Ergebnistupels befinden. Für diese Tupel wird dann ermittelt, welche Operatoren der Anfrage sie in die Ausgabe gebracht hätten. Der Algorithmus funktioniert für Anfragen, die Selektionen, Projektionen, Verbünde und Aggregationen enthalten (SPUJA) sowie Vereinigungen solcher SPJA-Anfragen.

NedExplain von Bidoit et al. [7] funktioniert auf die gleiche Weise wie der Why-Not Algorithmus von Chapman und Jagadish, hat aber laut Herschel et al. [20] den Vorzug, dass er eine vollständigere, korrektere und detailliertere Anwort liefert.

ConQueR von Tran und Chan [34] ist ein Algorithmus, der *modification-based answers* ausgibt. Er funktioniert mit Anfragen, die aus Selektionen, Projektionen, Verbünden, Vereinigungen und Aggregation bestehen (SPUJA). Er funktioniert ähnlich dem Why-Not-Algorithmus, ändert aber zusätzlich die Anfrage so, dass die *missing answer* beim nächsten Mal ausgegeben wird.

Conseil von Melanie Herschel [20] zeigt einen hybriden Algorithmus, der zusätzlich zu SPUJA noch Mengendifferenzen benutzen kann.

GProM von Arab et al. [6] ist eine plattformunabhängige Middleware zur Berechnung der Provenance von Anfragen, Updates und Transaktionen. Das System kann Provenance-Berechnungen auf nebenläufigen Datenbanktransaktionen berechnen [6].

Layer Based Architecture for Provenance in Big Data von Agrawal et al. [1] stellen eine Technik vor, um Provenance im Big-Data-Umfeld zu berechnen. Dafür verwenden sie eine Schichtenarchitektur. Sie trennen die Datenhaltung, die Berechnung und Anzeige der Provenance-Informationen, sowie den Zugriff auf die Daten. Die Zugriffsschicht ermöglicht in diesem Ansatz insbesondere Sicherheits- und Privacy-Aspekte.

HadoopPrv von Hopper et al. [2] ist eine modifizierte Version von Hadoop und ermöglicht eine Provenanceberechnung in MapReduce-Aufgaben. Eine vom Benutzer bereitgestellte Map-Funktion liest und filtert die Daten aus einer Eingabe. Diese Daten werden dann verteilt berechnet und durch eine Reduce-Funktion wieder zusammengeführt. Hadoop ist eine sehr verbreitete Open Source Implementierung dieses MapReduce-Verfahrens. [2].

4.3 Auswahl des Frameworks

In dieser Arbeit wird das zuerst vorgestellte Framework PERM verwendet. Zum einen wird die Provenance der Antworten nur auf Bedarf berechnet. Deshalb lässt sich das zugrundeliegende DBMS auch ohne die Provenance-Berechnung performant nutzen. Weiter gibt PERM die Ergebnisse seiner Berechnung als Relation in der Datenbank wieder aus, dies erlaubt eine komfortable Weiterverarbeitung. Trotzdem ist die in dieser Arbeit vorgestellte Methode nicht abhängig von einem bestimmten Provenance-Framework. Die richtige Auswahl des Frameworks hängt vom Einsatzgebiet der Methode ab.

Kapitel 5

Berechnung von intensionalen Antworten auf Provenance-Anfragen

Die hier beschriebene Methode soll es ermöglichen, eine Datenbank in mehr als eine Richtung zu durchsuchen. In der klassischen Richtung gehen wir von den Ursprungsdaten aus und erwarten eine Antwort auf eine Anfrage. Zusätzlich soll hier dem Anwender die Möglichkeit gegeben werden auch die andere Richtung, von der Antwort zur Ursprungsrelation zu gehen. Es wird eine Lösung für die großen Datenmengen in der Provenance-Antwort vorgeschlagen. Die Herausforderung der kontinuierlichen Daten soll durch die Auswahl der Anwendungsfälle kurz skizziert werden.

5.1 Methoden

5.1.1 Konzept

Hier wird das Konzept zur Beantwortung der Forschungsfrage vorgestellt. Die grundsätzliche Vorgehensweise wird zunächst in einem Diagramm skizziert und dann beschrieben. In diesem Abschnitt werden die zuvor erläuterten Grundlagen und Methoden zu einem vollständigen Vorgehen zusammengefügt.

In Abbildung 5.1 sieht man einen Ablauf der vorgestellten Methode. Am Anfang des Diagramms stehen die Ursprungsrelationen aus der Datenbank.

1. Im ersten Schritt müssen die Daten in eine Konzepthierarchie gebracht werden.
2. im zweiten Schritt ist die eigentliche Anfrage $Q1$ auf dem Datenbestand. Das benutzte Tool PERM [24] von Glavic et al. unterstützt

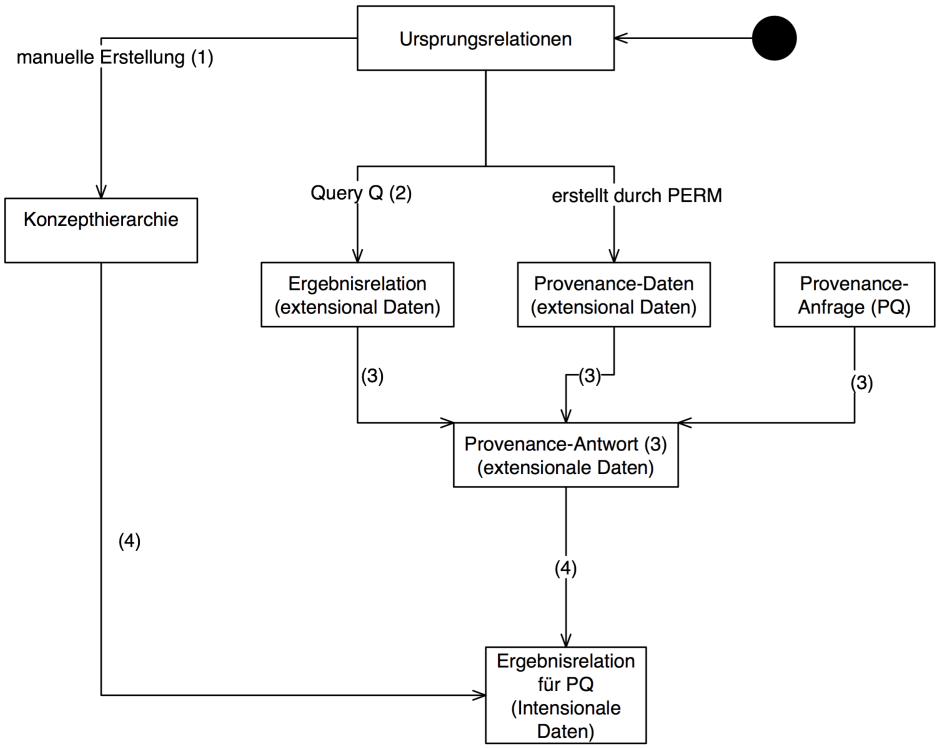

Abbildung 5.1: Konzeptdiagramm

SPJA-Anfragen, d.h. Selektionen, Projektionen, Verbünde und Aggregate.

3. Der dritte Schritt findet zeitgleich mit dem zweiten Schritt statt. Die Software PERM berechnet während der Ausführung von $Q1$ die Provenance der Anfrage. Dies wird erläutert in Kapitel 4.1. Man erhält dann das Anfrageergebnis von $Q1$ inklusive aller Provenance-Informationen bezüglich $Q1$ und dem Ursprungsdatensatz.
4. Schritt 4 besteht aus drei einzelnen Komponenten, die alle gleichzeitig abgearbeitet werden.
   (a) Eine Provenance-Anfrage $PQ1$ wird gestellt.
   (b) Die durch $PQ1$ gefilterten Daten werden Mithilfe der bereits beschriebenen Methode in eine intensionale Antwort umgewandelt.

5. Im letzten Schritt wird die intensionale Antwort ausgegeben. Die ausgegebene intensionale Antwort soll dem Anwender die Provenance-Anfrage beantworten. Die Qualität der Antwort wird am Ende der Anwendungsfälle diskutiert.

5.1.2 Konzepthierarchie

In Abschnitt 3.2.1 wurde eine Methode zur Erstellung und Verwendung von Konzepthierarchien nach Park et al. [32] vorgestellt. Für die Zwecke dieser Arbeit wird die Methode erweitert. Um eine automatische Generierung von intensionalen Antworten zu erreichen, wollen wir alle Attribute einer beliebigen Tabelle in eine Konzepthierarchie einordnen. Dies ist eine syntaktische Anpassung, um die Verarbeitung der Tabellen im Programm zu erleichtern. Wie bereits in Abschnitt 3.2.1 beschrieben, gibt es zwei Fälle, in denen es nicht möglich ist, eine Konzepthierarchie für das jeweilige Attribut zu erstellen. Um dies zu umgehen, werden wir für solche Attribute eine generische Konzepthierarchie erstellen. Diese ist semantisch nicht hilfreich, lässt aber zu, im weiteren Verlauf alle Attribute gleich zu behandeln. Dadurch erhalten wir drei verschiedene Typen an Konzepthierarchien, die im folgenden definiert werden.

Attribute mit kategorisierbaren numerischen Werten (Typ 1) Diese Art von Konzepthierarchie basiert auf numerischen Werten, die sich durch den Domain-Experten sinnvoll in kleinere Intervalle zerlegen lassen. Es ist nicht ausgeschlossen, dass diese Intervalle automatisch generiert werden können. Je nach Komplexität des Attributes kann die Hierarchie mehrere Stufen enthalten. Eine solche Hierarchie sieht dann aus wie in Abbildung 5.2. Man sieht, dass die Genauigkeit der Angaben zur Wurzel immer kleiner wird. Den höchsten Grad der Genauigkeit erhält man in den Blättern des Baumes, in denen die eigentlichen Werte (value) stehen. Der Baum besitzt zwar tatsächlich 5 Ebenen, davon tatsächlich aussagekräftig sind nur die Stufen 2, 1 und die Wurzel.

Attribute mit nicht numerischen Werten ohne sinnvolle Hierarchie (Typ 2) Für diese Attribute wird eine generische Hierarchie erstellt. In Abbildung 5.3 sieht man, dass die Konzepthierarchie so gut wie keinen Mehrwert an Informationen trägt. Es gibt jedoch einen Fall, wenn alle Werte (value) gleich sind, in dem man in Stufe 1 endet. In der überwiegenden Mehrzahl der Anwendungsfälle endet man in der Wurzel. Aus der Sicht des Programms sehen diese Attribute aber genauso aus wie die des ersten Typs.

Kategorisierbare Attribute mit nicht numerischen Werten (Typ 3) Attribute von diesem Typ werden vom Domainexperten in eine Hierarchieform gebracht. Dies ist das Anfangsbeispiel aus Abbildung 3.6. Dieser Typ

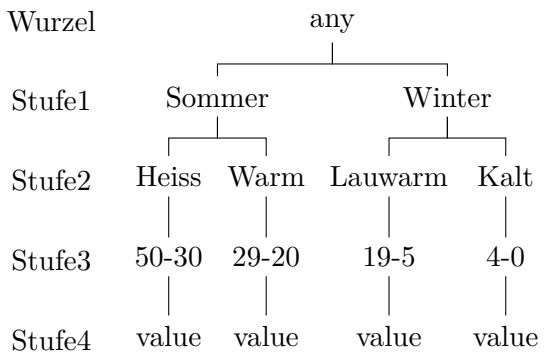

Abbildung 5.2: Attribute mit kategorisierbaren Werten

ist wie der erste Typ sehr aussagekräftig und wird vom Blatt zur Wurzel hin immer allgemeiner.

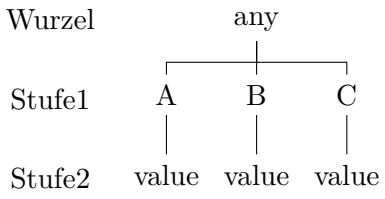

Abbildung 5.3: Attribute mit nicht kategorisierbaren Werten

5.1.3 Granularität

In Abschnitt 3.2.1 wird eine Schwelle beschrieben, die festlegt, wann der Algorithmus abbricht. Diese kann vom Anwender zuvor gewählt werden. In dieser Arbeit wird diese Schwelle "Granularität" genannt. Eine Granularität von Eins bedeutet, dass alle Werte eines Attributes nur einen Wert haben dürfen, damit der Algorithmus abbricht. Die Granularität kann für jedes Attribut einzeln bestimmt werden. Die maximale Granularität ist festgelegt durch die Breite der Konzepthierarchie auf der untersten Stufe. Die maximale Granularität g in Abbildung 5.3 ist $g = 3$. Für diesen Fall bricht der Algorithmus gleich am Anfang ab.

Definition 11 (Granularität) *Die Granularität g eines Attributes A ist definiert als $g \in \{1, \ldots, |dom(A)|\}$, wobei $|dom(A)|$ die Anzahl aller in der Domäne von A vorkommenden Werte ist.*

5.2 Algorithmus zur Erstellung intensionaler Antworten auf Provenance-Anfragen

Die dargelegten Grundlagen und Methoden werden in diesem Abschnitt zu einem einheitlichen Handlungsablauf zusammengestellt. Zunächst haben wir ein Datenbankschema S mit einer Menge von Relationenschemata $S := \{R_1, \ldots, R_p\}$ mit $p \in \mathbb{N}$. Jedes dieser Relationenschemata hat eine endliche Menge von Attributen A_i mit $i \in \mathbb{N}$.

5.2.1 Algorithmus 1 (Erstellung der Konzepthierarchie)

1.1 Kategorisierung Jedem Attribut A wird ein Typ aus Abschnitt 5.1.2 zugewiesen. Dies geschieht manuell durch den Anwender.

1.2 Hierarchisierung Für jedes Attribut wird gemäß seinem ermittelten Typ eine Konzepthierarchie nach dem in Abschnitt 5.1.2 beschriebenen Vorgehen erstellt. Dies wird vom Anwender manuell gemacht. Nach der ersten Phase gibt es für jedes Attribut einen Baum, der die jeweilige Konzepthierarchie des Attributs darstellt. Wir erhalten eine Menge $M := \{KH(A_1), KH(A_2), \ldots, KH(A_i)\}$. Damit ist die Vorbereitung des Datensatzes abgeschlossen. Jede Konzepthierarchie besteht mindestens aus einem Baum mit 3 Ebenen wie in Abbildung 5.3.

Algorithmus 5.1: Erstellung der Konzepthierarchie. Manuell durch den Anwender

```
1: ErstelleKonzepthierarchie(S)                    ▷ S := {R_1, ..., R_p}
   Gibt einen Konzepthierarchie KH jedes Attribut im Schema S zurück.
2:     for R_j ← 1, ..., j ∈ S do                  ▷ Für jede Relation aus S
3:         for A_k ← 1, ..., i ∈ R_j do            ▷ Für jedes Attribut aus R_j
4:             A_k ← Typ()                         ▷ Typ zuweisen
5:             A_k ← Konzepthierarchie             ▷ Konzepthierarchie zuweisen
6:         end for
7:     end for
8:     return KH(S)
9: end
```

5.2.2 Algorithmus 2 (Erstellung von intensionalen Antworten)

Der Anwender wählt eine Granularität des Algorithmus. Dies ist ein Wert $g \in \mathbb{N}$, der die Anzahl der verschiedenen Werte innerhalb eines Attributes festlegt. Bei einer Granularität von $g = 2$ ist die maximale Anzahl von verschiedenen Werten innerhalb eines Attributes genau Zwei.

2.1 Eingabe: Die Eingabe ist eine Relation RS, für deren Attribute zunächst eine Konzepthierarchie erstellt wurde.

2.2 Falls sich Attribute A_k in der Relation RS befinden, die keine korrespondierende Konzepthierarchie aus der Menge M haben, werden diese entfernt.

2.3 Jeder Wert wird in Richtung seines höherliegenden Konzepts generalisiert. Der neue Wert wird in die Relation RS eingetragen und ersetzt einen Wert.

2.4 Es wird für jedes Attribut $A_i \in RS$ geprüft, ob die Anzahl der verschiedenen Werte der zuvor bestimmten Granularität entspricht. Für jedes Attribut, auf das dies zutrifft, bricht der Algorithmus ab.

2.5 Für jedes Attribut, für das die Bedingung aus 2.4 nicht zutrifft, geht der Algorithmus wieder in 2.3.

Algorithmus 5.2: Generierung einer Intensionalen Antwort aus einer Relation

```
1:  ErstelleIntesionaleAnwort(RS, g, KH(S))        ▷ i ∈ RS, g ∈ ℕ
    Gibt eine Relation zurück.
2:      stringlen ← length of string
3:      for A_j ← 1, ..., i ∈ RS do                ▷ Für jedes Attribut in RS
4:          for A_j ← 1, ..., g(A_j) do  ▷ Für die Granularität des Attributes
5:              for value ← A_j do          ▷ Für jeden Wert in der Spalte A_j
6:                  value ← KH      ▷ Wert durch höheres Konzept ersetzten
7:                  if Granularität g erreicht then continue
8:                  end if
9:              end for
10:         end for
11:     end for
12:     return RS
13: end
```

5.2.3 Algorithmus 3. (Erstellung intensionaler Antworten auf Provenance Queries)

3.1 Ausführung von Algorithmus 5.1 zur Erstellung der Konzepthierarchie

3.2 Anfrage Im nächsten Schritt kann der Anwender mit seiner Arbeit fortfahren und eine Anfrage Q an die Datenbank stellen, wodurch er eine

Ergebnisrelation RS erhält. Die Komplexität der Anfrage Q ist abhängig vom verwendeten Provenance-Algorithmus und soll in diesem Teil nicht eingeschränkt werden.

3.3 Provenance-Anfrage Die Provenance-Anfrage PQ wird auf ein Tupel der Ergebnisrelation RS angewendet $PQ(RS)$ wodurch der verwendete Provenance-Algorithmus die Ursprungstupel ausgibt. Wir erhalten eine neue Relation PR, in der alle Tupel aufgeführt sind, die der Provenance-Algorithmus herausgefunden hat. Dies ist die extensionale Antwort auf die Anfrage PQ.

3.4 Ausführung von Algorithmus 5.1 zur Erstellung der intensionalen Antwort.

3.5 Alle Tupel, die Duplikate eines anderen Tupels sind, werden aus der Relation gelöscht.

5.3 Einige Experimente an realen Datensätzen

5.3.1 Anwendungsfall 1

Erläuterung des Versuchsaufbaus

Wie im laufenden Beispiel haben wir zwei Tabellen, die jeweils folgendes Schema aufweisen.

$$\_data := \{\_sensorID, \_temperature, \_timestamp\}$$
$$metadata := \{\_sensorID, \_manufacturer, \_position, \_isolation\}$$

In der Sensortabelle finden wir drei Attribute. Der Schlüssel dieser Tabelle setzt sich zusammen aus _sensorID und Time. Hierbei kennzeichnet die *_sensorID* einen Temperaturfühler, der an einer bestimmten Position befestigt ist. Die Position ist aus dieser Tabelle *data* nicht erkennbar. Das Attribut *_temperature* ist ein Temperaturwert, der zu einem gewissen Zeitpunkt aus der Spalte *_timestamp* gemessen wurde. Verschiedene Temperatursensoren können zur selben Zeit einen Wert ermitteln. Hieraus ergibt sich dann die Schlüsseleigenschaft von *_sensorID* und *_timestamp*, denn ein Temperatursensor kann zu einem bestimmten Zeitpunkt (*_timestamp*) nur einen Wert (*_temperature*) ausgeben. In der Tabelle *metadata* findet sich das Attribut *_ID*, dies ermöglicht einen Verbund mit der Sensortabelle. Weiter gibt es zwei Attribute *_manufacturer* und *_isolation*, die Zeichenketten bzw. boolsche Werte haben.

Die hier beschriebene Sensortabelle misst also die Temperatur in verschiedenen Räumlichkeiten.

Durchführung der Algorithmen

1. Kategorisierung der Attribute Die Attribute der Relationen *data* und *metadata* werden gemäß dem Algorithmus aus Abschnitt 5.1 kategorisiert.

- *_sensorID* (Typ 2) Hier verwenden wir die generische Typ-2-Hierarchie. Für die Identifikationsnummern der Sensoren gibt es zunächst keine sinnvolle Hierarchie.
- *_temperature* (Typ 2) Diese Entscheidung liegt in der Hand des Anwenders. Im vorliegenden Beispiel werden Temperaturen mit einer geringen Streuung betrachtet, weshalb eine Einteilung in Intervalle nur schwer zu realisieren ist.
- *_timestamp* (Typ 2) Ähnlich wie bei _temperature, könnte man auch hier den Typ 1 benutzen.
- *_ID* (Typ 2) Für dieses Attribut gilt dasselbe wie für die _sensorID.
- *_manufacturer* (Typ 2) Hier wird es auch nur die generische Hierarchie geben.

- *_isolation* (Typ 2)
- *_position* (Typ 3) Für dieses Attribut lässt sich eine Konzepthierarchie wie in Abbildung 5.5 erstellen. Die Wurzel des Baumes in der Abbildung ist frei gewählt.

2. Erstellung der Konzepthierarchie In Abbildung 5.4 sieht man die Ergebnisse der Erstellung der Konzepthierarchie der mit Typ 2 kategorisierten Attribute. Jeder mögliche Wert der Relation ist ein eigenes Blatt. Die Konzepthierarchie von *_temperature* zeigt, dass es sich um eine Abstraktion handelt. In der eigentlichen Umsetzung wird natürlich kein Baum mit potentiell unendlich vielen Bättern aufgebaut, sondern jeder real gemessene Wert wird als ein Blatt des Baumes betrachtet. Die Konzepthierarchie für das Attribut *_position* sieht man in Abbildung 5.5.

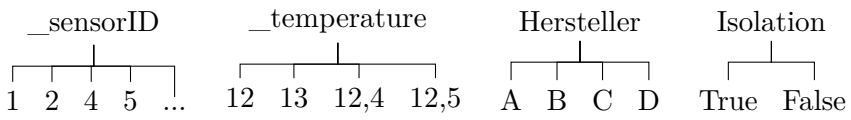

Abbildung 5.4: Einstufige Konzepthierarchien

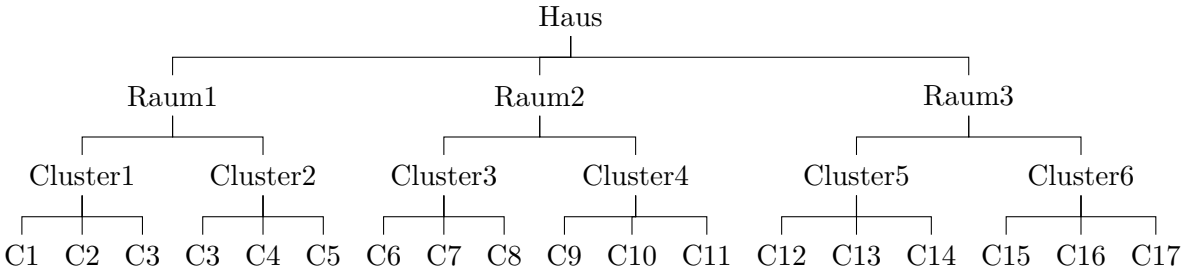

Abbildung 5.5: mehrstufige Konzepthierarchie

3. Ausführung der Anfrage In Anfrage 5.1 sieht man die gestellte Anfrage in SQL-Notation. Es soll die Durchschnittstemperatur aller Sensoren im Zeitraum 1 bis 5 und mit einer Mindesttemperatur von 13 angezeigt werden. Als Ausgabe erhält der Anwender den Durchschnittswert 14,5.

Anfrage 5.1: Anfrage Anwendugsfall 1

```
1    SELECT AVG(\_temperature)
2     FROM data as ST
```

```
3      JOIN  metadata as MT
4         ON ST.\_sensorid = MT.ID
5     WHERE ST.\_timestamp BETWEEN 1 AND 5
6      AND ST.\_temperature > 13;
7
```

4. Provenance-Anfrage Unter Benutzung des Programms PERM und der dadurch erhaltenen Anfragefunktion `PROVENANCE` ändern sich Anfrage und Ergebnis wie in Anfrage 5.2 bzw. Tabelle 5.1 zu sehen ist. Wir erhalten eine Relation, in der jedes Tupel aus der Ursprungsrelation aufgeführt ist, das die Selektionsprädikate der Anfrage 5.1 erfüllt und somit zur Berechnung des Ergebnisses 14,5 beigetragen hat.

Anfrage 5.2: Anfrage mit der Verwendung des Schlüsselwortes `PROVENANCE`

```
1     SELECT PROVENANCE AVG(\_temperature)
2     FROM data as ST
3     JOIN  metadata as MT
4     ON ST.\_sensorID = MT.\_id
5     WHERE ST.\_timestamp BETWEEN 1 AND 5
6     AND ST.\_temperature > 13;
7
```

AVG(_temp)	_sensorID	_temp	_time	_manu	_pos	_isol
14,5	1	14	100	A	C1	TRUE
14,5	14	15	100	B	C2	TRUE
14,5	1123	23	101	A	C3	TRUE
14,5	112	21	201	C	C5	TRUE
14,5	431	13	201	D	C4	TRUE

Tabelle 5.1: Ergebnis von Anfrage 5.2

5. Erstellung der intensionalen Antwort Das Ergebnis des vierten Schritts ist eine extensionale Antwort. Auf jedes Attribut wird nun gemäß dem Algorithmus die zuvor definierte Konzepthierarchie angewendet. Nach der ersten Iteration sieht das Ergebnis aus wie in Tabelle 5.2. Der Algorithmus soll aber erst abbrechen, wenn er für alle Werte eines Attributes ein

einheitliches Konzept gefunden hat. Dies ist für alle Attribute der Fall, außer für das Attribut *_position* der Fall. Nach einer weiteren Iteration ist das Ziel erreicht und der Algorithmus bricht an dieser Stelle ab. Das Ergebnis ist in Tabelle 5.3 zu sehen. Da nun alle Tupel der Relation dieselben Werte haben, können wir die Relation auch auf ein Tupel reduzieren und erhalten die intensionale Antwort als Tupel: $< any, any, any, any, any, Raum1, TRUE >$

AVG(_temp)	_sensorID	_temp	_time	_manu	_pos	_isol
14,5	1	14	any	Hersteller	Cluster1	TRUE
14,5	14	15	any	Hersteller	Cluster1	TRUE
14,5	1123	23	any	Hersteller	Cluster1	TRUE
14,5	112	21	any	Hersteller	Cluster2	TRUE
14,5	431	13	any	Hersteller	Cluster2	TRUE

Tabelle 5.2: Nach der ersten Iteration

AVG(_temp)	_sensorID	_temp	_time	_manu	_pos	_isol
14,5	1	14	any	Hersteller	Raum1	TRUE
14,5	14	15	any	Hersteller	Raum1	TRUE
14,5	1123	23	any	Hersteller	Raum1	TRUE
14,5	112	21	any	Hersteller	Raum1	TRUE
14,5	431	13	any	Hersteller	Raum1	TRUE

Tabelle 5.3: Nach der zweiten Iteration

Ergebnis Anwendungsfall 1

Die Interpretation der intensionalen Antwort liegt dann wieder in der Hand des Anwenders. Er erfährt durch die intensionale Antwort, dass die Sensoren in Raum1 für das Ergebnis verantwortlich waren und kann seine nächste Anfrage mit diesem erworbenen Wissen neu eingrenzen. Man sieht auch,

das beim Attribut *_isolation* zwar überall derselbe Wert herausgekommen ist und dadurch auch Bestandteil der intensionalen Antwort geworden ist. Allerdings ist dies keine Aussage darüber, ob dies auch relevant für das Ergebnis ist. Ohne Zweifel ist aber, dass die intensionale Antwort deutlich übersichtlicher ist als das Ergebnis aus Tabelle 5.1.

5.3.2 Anwendungsfall 2

In einem weiteren Anwendungsfall soll mittels des Datensatzes gezeigt werden, dass der Algorithmus auch mit komplizierteren Anfragen umgehen kann. Zusätzlich soll die Granularität des Algorithmus geändert werden.

Erläuterung des Versuchsaufbaus

In diesem Fall will der Anwender wissen, wie sich die Durchschnittstemperatur im Laufe der Zeit geändert hat. Die Konzepthierarchie und die Kategorisierung bleiben dieselben wie in Anwendungsfall 1. Darum springen wir gleich in den Schritt zwei des Algorithmus aus 5.2.3.

3. Anfrage Im zweiten Anwendungsfall will der Anwender eine Temperaturveränderung über alle Sensoren wissen. Anfrage 5.3 ist die dazugehörige Anfrage. Es wird jeweils die Temperatur über den gesamten Zeitraum gemessen, sowie parallel dazu die Temperatur fünf Zeiteinheiten später.

Anfrage 5.3: Anfrage für Anwendungsfall 2

```
1
2 select avg(ST._temperature) AS T, avg(ST_1._temperature) AS
      TPLUS1
3 from \_temperature as ST
4 join data as ST_1
5 on ST._timestamp +5 = ST_1._timestamp and ST._sensorID =
     ST_1._sensorID
6 join metadata as mt on st._sensorID = mt._id
7 where ST_1._temperature < ST._temperature
8       )
9
```

Das Ergbnis dieser Anfrage 5.3 ist das Tupel $< 11,7; 11,4 >$. Die Durchschnittstemperatur ist in der Tat gesunken. Nun weiß der Anwender allerdings nicht, wo und warum dies geschehen ist. Es ist natürlich möglich, dass er seine Anfrage anders stellt und auch auf einem traditionellen Weg zu der Antwort auf diese gelangt.

4. Provenance-Anfrage Durch das Hinzufügen des Kommandos `PROVENANCE` nach dem `SELECT` in der Anfrage 5.3. Erhalten wir wieder eine extensiona-

le Antwort auf die Frage nach den Ursprungstupeln. Die Tabelle wird in diesem Fall schon sehr lang, weshalb sie im Anhang zu finden ist. Eine stark verkürzte Form mit allen wesentlichen Informationen befindet sich in Abbildung 5.4. In der vollständigen Relation sind auch noch alle anderen Attribute aufgeführt. Diese würden in den folgenden Schritten aber alle in die Wurzel ihrer Konzepthierarchie abgebildet werden, weshalb sie hier zur besseren Übersicht nicht mit aufgeführt werden.

t	tplus1	_sensorID	_position
11,7	11,05	14	C6
11,7	11,05	14	C6
11,7	11,05	14	C6
11,7	11,05	14	C6
11,7	11,05	18	C6
11,7	11,05	18	C6
11,7	11,05	18	C6
11,7	11,05	18	C6
11,7	11,05	18	C6
11,7	11,05	18	C6
11,7	11,05	18	C6

Tabelle 5.4: Ergebnisrelation Anwendungsfall 2

5. Erstellung der intensionalen Antwort Wie anfangs beschrieben bricht der Algorithmus ab, nicht wenn alle Werte eines Attributs in der selben Hierarchieebene sind, sondern wenn nur noch zwei verschiedene Werte in einem Attribut vorhanden sind. Das Ergebnis nach nur einer Iteration ist in Tabelle 5.5 zu sehen.

t	tplus1	_sensorID	_position
11,7	11,05	14	C6
11,7	11,05	18	C6

Tabelle 5.5: Intensionale Antwort im Anwendungsfall 2

Ergebnis Anwendungsfall 2

Diese Antwort ist keine reine intensionale Antwort, sondern eine gemischte. Die Erhöhung der Granularität hat eine zusätzliche Information zu Tage ergeben. Ausschließlich die Sensoren 14 und 18 haben eine Absenkung der Temperatur gemessen. Mit einer niedrigeren Granularität wäre diese Information verloren gegangen.

Die Ergebnisse beider Anwendungsfälle werden im nächsten Kapitel bewertet und diskutiert.

Kapitel 6

Diskussion und Fazit

6.1 Diskussion des vorgestellten Lösungsansatzes

Die Ergebnisse der Anwendungsfälle werden zunächst anhand der von Amihai Motro vorgeschlagenen Kriterien beurteilt [31]. Anschließend werden die Vorteile und die Nachteile der vorgestellten Methode diskutiert.

6.1.1 Bewertung der intensionalen Antworten

Vollständigkeit Die Vollständigkeit der intensionalen Antwort in der vorgestellten Methode ist eng verbunden mit der Konzepthierarchie. Die Granularität des Algorithmus 5.2.2 kann vom Anwender frei gewählt werden. Wenn der Algorithmus in der ersten Stufe abbricht, erhält man eine sehr spezielle vollständige Antwort, welche der Relation entspricht. Damit ist die intensionale Antwort deckungsgleich mit der extensionalen Antwort. Das andere Extrem erhält man, wenn man den Algorithmus bis zum Ende durchlaufen lässt, man also zwingend nur ein Ergebnistupel in der intensionalen Antwort hat. Hier kann es durchaus passieren, dass Informationen verloren gehen. Im Anwendungsfall 1 war die intensionale Antwort vollständig, da keine Tupel durch die Erstellung dieser Antwort verloren gegangen sind und zugleich keine Information verloren gegangen ist. Dies betrifft zumindest keine Information die initiale Frage betreffend. Im Anwendungsfall 2 ist die intensionale Antwort vollständig, weil der Algorithmus durch die eingestellte Granularität früher abbrach. Dadurch kam die intensionale Antwort aus Tabelle 5.5 zu Stande. Wäre der Algorithmus eine Iteration weiter gelaufen, wäre ein Informationsverlust betreffend das Attribut *_sensorID* aufgetreten.

Nichtredundanz Ähnlich wie bei der Vollständigkeit hängt es davon ab, welche Granularität eingestellt wurde. Durch die vorherige Erstellung einer Konzepthierarchie ist die intensionale Antwort auf keinen Fall nur ein Abbild der gestellten Anfrage, sondern enthält mehr oder andere Informationen.

Optimalität Die Präzision der Antworten ist, wie bei den anderen Kriterien zuvor, auch abhängig von den Einstellungen der Granularität. Mit wachsender Präzision geht die Prägnanz der Antworten verloren. Beide Aspekte hängen sehr stark mit der Auswahl der richtigen Konzepthierarchie zusammen, was wiederum an der Beschaffenheit der Attribute hängt. Mit sehr komplexen Konzepthierarchien für alle Attribute der Relationen lassen sich mit dieser Methode optimale Antworten finden.

Relevanz Auch dieser Aspekt ist abhängig von der gewählten Konzepthierarchie. In den beiden Anwendungsfällen kann man von einer Relevanz der Antworten in Bezug auf die gestellte Frage sprechen.

Mit der vorliegenden Methode können wir sowohl reine, als auch Kombinationen von intensionalen und extensionale Antworten darstellen. Eine Abhängigkeit von extensionalen Daten innerhalb der Antwort ist nicht gegeben. Die extensionalen Daten können aber zu Genauigkeit der Antwort beitragen.

6.1.2 Vorteile der vorgestellten Methode

Die vorgestellte Methode erfüllt zunächst die an sie gestellte Aufgabe, eine intensionale Antwort zu generieren. Sie macht dies aus extensionalen Daten, die im Sinne der Provenance *witnesses* eines Ergebnisses sind. Es ist gelungen, eine intensionale Antwort auf eine Provenance-Anfrage zu erhalten. Die Form der intensionalen Antwort lässt sich zudem vom Anwender durch die Einstellung der Granularität verändern. Der Aspekt der Big Data Analytics erscheint in dieser Arbeit durch die Form der gewählten Beispieldaten. Die manuelle Arbeit an der Konzepthierarchie ist eben nicht abhängig von der Menge der Stromdaten, weshalb die Methode auch auf sehr große Datenmengen anwendbar ist.

Die Unabhängigkeit zwischen der Erstellung intensionaler Antworten und der Berechnung der Provenance erlaubt es, beide auch durch einen anderen Algorithmus zu ersetzen. Hier bieten sich die im Abschnitt 4.2 vorgestellten Frameworks an.

6.1.3 Nachteile der vorgestellten Methode

Hier ist in erster Linie die aufwendige Erstellung der Konzepthierarchie durch den Domain-Experten zu nennen. Auch die manuelle Einstellung der Granularität kann ein Hindernis sein, wenn der Anwender zuvor nicht weiß, welche Granularität ein optimales Ergebnis liefert. Die Einschränkung der Methode ist durch die verwendeten Frameworks und Algorithmen gegeben. Im folgenden Kapitel Ausblick werden einige Vorschläge zur Erweiterung

der Methode gemacht. So ist eine Verwendung der *Where- How- und Why-Not-Provenance* in der vorliegenden Fassung nicht vorgesehen.
Zusätzlich lässt sich am Algorithmus von Park et al. noch vieles verbessern. Die Unterstützung des Anwenders bei der Erstellung der Konzepthierarchie ist hier zu nennen.

6.2 Ausblick

In diesem Abschnitt soll ein Ausblick gegeben werden, wie die hier vorgestellte Methode erweitert werden kann.

6.2.1 Automatische Erstellung der Konzepthierarchie

Im vorgestellten Algorithmus wird die Erstellung der Konzepthierarchie vollständig durch den Anwender übernommen. Bei Daten mit vielen numerischen Werten ist eine Implementierung eines Clustering-Algorithmus denkbar, der verschiedene Intervalle aus dem Ursprungsdatensatz errechnet. Um diesen Prozess zu automatisieren, bedürfte es einer Metrik als Maß für die Verwendbarkeit der Intervalle. Ein anderer Ansatz für die leichtere Erstellung der Konzepthierarchie kommt aus einem Teilbereich der Informatik, der sich mit Ontologien beschäftigt. Mit der Hilfe dieses Forschungszweiges ließe sich die Erstellung zum Teil automatisieren. Weiter können Assoziationsregeln aus dem Forschungszweig des Data Mining helfen Konzepthierarchien zu erstellen. Dann rückt deren Erstellung in den Vordergrund. Dies kann positive Auswirkungen auf die erhaltenen intensionalen Antworten haben.

6.2.2 Die missing Answers

Das finden der *missing answers* ist mit einigem Aufwand auch in die Methode implementierbar. Eine *instance-based explanation* könnte mit Hilfe der Konzepthierarchien wie folgt implementiert werden. Der Anwender stellt die Frage, warum sich ein bestimmtes Tupel nicht in der Antwort befindet. Das System lässt das fragliche Tupel mit derselben Granularität durch den Algorithmus laufen. Als Ergebnis erhält man die Konzepte aus der Hierarchie des Tupels. Diese werden sich in gewissen Punkten unterscheiden. Daran kann der Anwender oder das System erkennen, wo das Tupel modifiziert werden muss, um im ursprünglichen Ergebnis zu erscheinen.

Beispiel: Hierfür bedienen wir uns des Ergebnisses aus dem Anwendungsfall 2. Die intensionale Antwort bestand aus 2 Tupeln:

$$< 11,7; 11,05; 14; C6 >$$

$$< 11,7; 11,05; 18; C6 >$$

Der Anwender will nun wissen, warum der Sensor mit der Nummer 17 nicht im Ergebnis aufgetaucht ist. Um diese Frage zu beantworten, werden alle Tupel des Ursprungsdatensatzes, die den Sensor 17 betreffen, durch den Algorithmus verarbeitet. Die Anfrage bleibt hier dieselbe, allerdings ohne die Selektionsbedingungen. Als Ergebnis erhält der Anwender dann das folgende Tupel:

$$< 10,3; 11,04; 17; C7 >$$

Man sieht, dass das Tupel das Selektionsprädikat aus Anfrage 5.3 nicht erfüllt. Denn die Temperatur des Sensors ist gestiegen und nicht wie verlangt gesunken. Auf dieser Basis lässt sich die *Why-Not Provenance* in das Konzept integrieren.

6.2.3 Privatheit

Es kann Fälle geben, in denen der Ursprungsdatensatz Informationen enthält, die nicht für den Anwender bestimmt sind. Trotzdem soll dem Anwender die Möglichkeit gegeben werden, die vorgestellten Provenance-Berechnungen zu machen. Durch die Ausgabe einer intensionalen Antwort sind gewisse Informationen schon verallgemeinert. Um den Aspekt der Privatheit und des Datenschutzes in dieses Konzept zu integrieren, kann man an folgenden Punkten ansetzen.

- Einschränkung der Granularität. Wenn man verhindert, dass die Granularität auf die höchste Stufe gestellt wird, kann der Anwender immer nur eine verallgemeinerte Antwort sehen. Zusätzlich darf der Ersteller der Konzepthierarchie nicht dieselbe Person sein wie der Anwender.
- Sperrung gewisser Attribute. Wenn man von Anfang an gewisse Attribute von der Ausgabe ausschließt, tauchen sie nicht mehr in der intensionalen Antwort auf. Trotzdem wird die Berechnung der intensionalen Antwort mit Hilfe dieser gesperrten Attribute erfolgen, wodurch sich ein allzu großer Informationsverlust vermeiden lässt.

Diese Maßnahmen erfordern eine Bewertung nach in der Privacy-Forschung anerkannten Kriterien. *K-anonimity* und *l-diversity* [19] können vor die Ausgabe der intensionalen Antwort geschaltet werden.

6.3 Zusammenfassung

Die vorgestellte Methode erlaubt es dem Anwender, bei großen, unübersichtlichen Datenmengen eine verallgemeinerte und deskriptive Antwort zu erhalten. Für die anfangs beschriebenen Wissenschaftler und Ingenieure ist die Erstellung der Konzepthierarchie, durch ihr Domain-Wissen, möglich. Durch den modularen Aufbau der Methode ist es möglich andere Arten der Provenance-Berechnung zu implementieren.

Anhang A

Technische Informationen

A.1 Aktuelle Programmversionen

Diese Pakete werden benötigt um die Implementierung ausführen zu können

Programm	Version
VirtualBox VM	5.0.8
Java	8
Flex	2.5.4
Bison	1.875
libxslt	
libxml	
libz	
GNU make	

A.1.1 Inhalt der DVD

Datei	Beschreibung
Implementierung	Quellcode
exampledata.sql	Datensatz
perm0.1	Kopie der Perm Software
Implementierung10.jar	lauffähige jar-Datei
Bachlorarbeit.ova	Virtuelle Maschine

A.2 Automatische Ausführung über eine virtuelle Maschine

Die Implementierung ist auf einer virtuellen Maschine installiert, die der Arbeit auf der DVD beiliegt. Benutzt wird die Software Virtual Box der Firma Oracle[1]. Die notwendigen Programme sind alle lauffähig auf der virtuellen

[1] www.virtualbox.org

Maschine installiert. Hier folgt eine kurze Bedienungsanleitung für die Einbindung der VM und für die Ausführung des Programms. Die VM muss in das Programm Virtual Box importiert werden. Danach kann man sie starten. Die Benutzerdaten sind: `Username:bachlor Password:Bachlor`. Dies sind zugleich auch die Nutzerdaten der Datenbank. Wenn die VM geladen ist, dann kann man über ein Terminalfenster die Datenbank wie folgt starten. Zuerst ins Verzeichnis:

```
/home/bachlor/trunk/INSTALLDIR/bin/
```

wechseln und dann mit

```
postgres -D bachlordir >logfile 2>&1 &
```

den Datenbankserver starten. Dann kann im gleichen Verzeichnis das Program mit folgendem Befehl gestartet werden:

```
/home/bachlor/Schreibtisch/Implementierung/java -jar Implementierung10.jar
```

Das Program läuft nun automatisch im Terminalfenster ab.

A.3 Ausführung ohne die virtuelle Maschine

Dies ist eine Anleitung zur Installation der benötigten Programme.

Installation der Datenbank Zusätzliche Informationen sind auf der Website des Projektes zu erhalten [2]. Auf der DVD ist eine aktuelle Version von PERM enthalten. Für eine neue Version kann man sich das Repository herunterladen mit:

```
svn co http://svn.code.sf.net/p/permdbms/code/trunk
```

Die Installation läuft dann wie folgt:

```
./configure --with-libxml --with-libxslt --prefix=INSTALLDIR
make
make install
mkdir CLUSTERDIR
INSTALLDIR/bin/initdb -D CLUSTERDIR
INSTALLDIR/bin/postgres -D CLUSTERDIR >logfile 2>&1 &
INSTALLDIR/bin/createdb bachlor
INSTALLDIR/bin/psql bachlor
test=# CREATE LANGUAGE plpgsql;
test=# \i /pathToPermCode/contrib/xml2/pgxml.sql
```

Die im Code fest verankerten Zugansdaten sind: bachlor:bachlor

Einlesen des Datensatzes Den Versuchsdatensatz `exampledata.sql` kann man über folgenden Befehl in die Datenbank einlesen.

```
test=# \i /pathTo/exampledata.sql
```

Starten des Testprogramms

```
java -jar implementierung.jar
```

Das Programm läuft nun selbstständig ab und durchläuft zwei Anwendungsfälle.

[2] http://www.cs.iit.edu/ dbgroup/research/perm.php

Quellenverzeichnis

Literatur

[1] Rajeev Agrawal u. a. „A layer based architecture for provenance in big data“. In: *2014 IEEE International Conference on Big Data (Big Data)*. October 2015. IEEE, Okt. 2014, S. 1–7. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7004468 (siehe S. 27).

[2] Sherif Akoush, Ripduman Sohan und Andy Hopper. „HadoopProv : Towards Provenance As A First Class Citizen In MapReduce“. In: *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance* (2013) (siehe S. 28).

[6] B Arab u. a. „A generic provenance middleware for database queries, updates, and transactions“. In: *Proceedings of the 6th USENIX Workshop on the Theory and Practice of Provenance (TaPP)* (2014), S. 8. URL: https://www.usenix.org/system/files/conference/tapp2014/tapp14%7B%5C_%7Dpaper%7B%5C_%7Darab.pdf (siehe S. 27).

[7] Nicole Bidoit, Melanie Herschel und Katerina Tzompanaki. „Query-Based Why-Not Provenance with NedExplain“. In: *Intl. Conf. on Extending Database Technology (EDBT)* c (2014), S. 145–156 (siehe S. 27).

[9] Peter Buneman und Wang-chiew Tan. „Provenance in Databases“. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (2007), S. 1171–1173 (siehe S. 10, 12, 16, 25).

[10] Adriane Chapman und H. V. Jagadish. „Why not?“ In: *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09* (2009), S. 523. URL: http://portal.acm.org/citation.cfm?doid=1559845.1559901 (siehe S. 17, 27).

[11] James Cheney, Laura Chiticariu und Wang-Chiew Tan. „Provenance in Databases: Why, How, and Where“. In: *Foundations and Trends in Databases* 1.4 (2007), S. 379–474 (siehe S. 13, 16).

[12] E. F. Codd. „A relational model of data for large shared data banks“. In: *Communications of the ACM* 26.1 (1983), S. 64–69. URL: http://portal.acm.org/citation.cfm?doid=357980.358007 (siehe S. 1, 5).

[13] Jp Dijcks. „Oracle: Big data for the enterprise“. In: *Oracle White Paper* June (2012), S. 16. URL: http://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:Oracle+:+Big+Data+for+the+Enterprise%7B%5C#%7D0 (siehe S. 10).

[14] Boris Glavic. *A Primer on Database Provenance*. Techn. Ber. IIT/CS-DB-2014-01: Illinois Institute of Technology, 2014, S. 1–10. URL: http://cs.iit.edu/%7B%%7D7edbgroup/pdfpubls/G14.pdf (siehe S. 12).

[15] Boris Glavic. „Big Data Provenance: Challenges and Implications for Benchmarking“. In: *Specifying Big Data Benchmarks* (2014), S. 72–80. URL: http://link.springer.com/chapter/10.1007/978-3-642-53974-9%7B%5C_%7D7 (siehe S. 10, 19).

[24] Boris Glavic. „Perm : Efficient Provenance Support for Relational Databases“. Diss. University of Zurich, 2010 (siehe S. 13, 25, 29).

[16] Boris Glavic und Gustavo Alonso. „The perm provenance management system in action“. In: *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09* (2009), S. 1055. URL: http://portal.acm.org/citation.cfm?doid=1559845.1559980 (siehe S. 26).

[17] Boris Glavic, Renée J. Miller und Gustavo Alonso. „Using SQL for efficient generation and querying of provenance information“. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8000.Figure 1 (2013), S. 291–320 (siehe S. 13, 25, 27).

[18] Todd J Green, Grigoris Karvounarakis und Val Tannen. „Provenance semirings“. In: *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '07* pages.June (2007), S. 31. URL: http://portal.acm.org/citation.cfm?doid=1265530.1265535 (siehe S. 15, 16).

[19] Hannes Grunert und Andreas Heuer. „Generating privacy constraints for assistive environments“. In: *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments - PETRA '15*. New York, New York, USA: ACM Press, 2015, S. 1–4. URL: http://dl.acm.org/citation.cfm?doid=2769493.2769542 (siehe S. 46).

[20] Melanie Herschel. „A Hybrid Approach to Answering Why-Not Questions on Relational Query Results“. In: *Journal of Data and Information Quality* 5.3 (März 2015), S. 1–29. URL: http://dl.acm.org/citation.cfm?doid=2698232.2665070 (siehe S. 27).

[21] Melanie Herschel, Mauricio a Hern a ndez und Wang Chiew Tan. „Artemis: a system for analyzing missing answers“. In: *Proc. VLDB Endowment* 2 (2009), S. 1550–1553. URL: http://dl.acm.org/citation.cfm?id=1687553.1687588$%5Cbackslash$npapers2://publication/uuid/8F2B49D6-EA8D-41B8-9612-96D0B937D8E0 (siehe S. 17).

[22] Jiansheng Huang u. a. „On the provenance of non-answers to queries over extracted data“. In: *Proceedings of the VLDB Endowment* 1.1 (2008), S. 736–747 (siehe S. 17).

[23] Robert Ikeda, Hyunjung Park und Jennifer Widom. „Provenance for generalized map and reduce workflows“. In: *Proceedings of the Fifth CIDR* (2011), S. 273–283. URL: http://ilpubs.stanford.edu:8090/985/ (siehe S. 19).

[25] Han. J., Y Cai und N Cercone. „Data-Driven Discovery of Quantitative Ruels in Relational Databases“. In: *IEEE Transactions on Knowledge and Data Engineering* 5.1 (1993), S. 29–40 (siehe S. 20).

[26] Ml Kersten u. a. „The researcher's guide to the data deluge: Querying a scientific database in just a few seconds“. In: *PVLDB Challenges and ...* 4.12 (2011), S. 1474–1477. URL: http://dblp.uni-trier.de/rec/bibtex/journals/pvldb/KerstenIML11$%5Cbackslash$nhttp://oai.cwi.nl/oai/asset/18546/18546B.pdf (siehe S. 1, 11).

[27] Alexandros Labrinidis und H. V. Jagadish. „Challenges and opportunities with big data“. In: *Proceedings of the VLDB Endowment* 5.12 (Aug. 2012), S. 2032–2033. URL: http://dl.acm.org/citation.cfm?id=2367572%20http://dl.acm.org/citation.cfm?doid=2367502.2367572 (siehe S. 10, 19).

[28] Clifford a Lynch. „When documents deceive: Trust and provenance as new factors for information retrieval in a tangled web“. In: *Journal of the American Society for Information Science and Technology* 52.1 (2001), S. 12–17. URL: http://dx.doi.org/10.1002/1532-2890(2000)52:1%3C12::AID-ASI1062%3E3.0.CO;2-V (siehe S. 12).

[29] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983 (siehe S. 5–7).

[30] Tanu Malik, Ligia Nistor und Ashish Gehani. „Tracking and sketching distributed data provenance“. In: *Proceedings - 2010 6th IEEE International Conference on e-Science, eScience 2010* (2010), S. 190–197 (siehe S. 19).

[31] Amihai Motro. „Intensional Answers to Database Queries“. In: *IEEE Transactions on Knowledge and Data Engineering* 6.3 (1994), S. 444–454 (siehe S. 9, 23, 24, 43).

[32] E K Park und Suk-Chung Yoon. „An Approach to Intensional Query Answering at Multiple Abstraction Levels using Data Mining Approaches“. In: *Proc. HICSS* 00.c (1999), S. 1–9 (siehe S. 8, 19, 20, 31).

[33] Gunter Saake, Kai-Uwe Sattler und Andreas Heuer. *Datenbanken - Konzepte und Sprachen, 4. Auflage.* MITP, 2010, S. I–XVII, 1–783 (siehe S. 5, 7).

[34] Quoc Trung Tran und Chee-yong Chan. „How to ConQueR Why-Not Questions Categories and Subject Descriptors“. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (2010), S. 15–26 (siehe S. 17, 27).