HAW HAMBURG

Bachelorarbeit

Martin Gosch

Experimentelle Evaluation von Frameworks zur Speicherung und Verarbeitung unsicherer Daten

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Martin Gosch

Experimentelle Evaluation von Frameworks zur Speicherung und Verarbeitung unsicherer Daten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 19. Juni 2020

Martin Gosch

Thema der Arbeit

Experimentelle Evaluation von Frameworks zur Speicherung und Verarbeitung unsicherer Daten

Stichworte

Datenbanken, SE, Wahrscheinlichkeit, Frameworks

Kurzzusammenfassung

In dieser Arbeit sollen Frameworks zur Speicherung und Verarbeitung unsicherer Daten mit Experimenten evaluiert werden. Anhand dieser Experimente soll ein Vergleich der Systeme und eine Bewertung erfolgen. Zu Beginn werden Grundlagen aus dem Bereich der Datenbanken vermittelt. Danach werden Konzepte und Begrifflichkeiten aus dem Bereich der unsicheren Daten erklärt. Diese bilden die Grundlage für die Funktionsweisen der evaluierten Frameworks. Dann werden die beiden Frameworks, Trio und MayBMS vorgestellt. Zu jedem Framework gibt es eine Erklärung der Speicherungskonzepte und einen Überblick über den Umfang des Frameworks. Abschließend findet eine Evaluation statt. Dabei werden Hypothesen aufgestellt, die anhand von Experimenten unterlegt werden.

Martin Gosch

Title of Thesis

Experimental evaluation of Frameworks for storing and processing uncertain data

Keywords

Databases, SE, Probability, Frameworks

Abstract

This paper concerns an experimental evaluation of frameworks for storing and processing of uncertain data. Based on those experiments a comparison between the frameworks and a final assessment shall be given. In the beginning basics in the field of databases are defined. After that, concepts out of the field of uncertain data are explained. They build the base for the functionality of the evaluated frameworks. Next, the two frameworks, Trio and MayB;S are presentet. For each framework there is an explanation of its storing concept and an overview on the extent of the framework. Lastly, there is an evaluation. Based on experiments different hyptheses are discussed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Die Wissenschaft schreitet stets voran und mit ihr die Forschung. Unter anderem im informationstechnischen Bereich werden dabei Messergebnisse gesammelt, die uneindeutige Daten liefern, die unsicher genannt werden. Schon seit Beginn der Entwicklung von Datenbankkonzepten gibt es Überlegungen, wie mit unsicheren Daten umzugehen ist. Im Laufe der Zeit wurden mehrere Frameworks entwickelt, um unsichere Daten zu verarbeiten.

1.1 Motivation

Die Moderne Forschung steht oft vor der Herausforderung, große Datenmengen zu verarbeiten. Dies kann am Beispiel von sensorbasierten Messungen demonstriert werden. Schon günstige Sensoren besitzen Taktraten von etwa 50 Messungen pro Sekunde [19]. Wird die Anzahl der Messungen für einen Tag berechnet, werden schon 4,32 Millionen Datensätze von Messungen erhalten, wenn nur einer dieser Sensoren eingesetzt wird. Diese günstigen Sensoren sind aber bei der Messrate noch am unteren Ende des Spektrums angesiedelt. Sensoren in Forschung und Wissenschaft besitzen oft deutlich höhere Messfrequenzen. Diese Menge an Daten muss irgendwie verwaltet werden, wozu im Allgemeinen Datenbanken verwendet werden.

Wenn solche Messergebnisse eindeutig sind, kann dieser Anwendungsfall leicht mit relationalen Datenbanken abgebildet werden. Interferenzen in den Sensordaten können aber dafür sorgen, dass die Ergebnisse unsauber werden, so dass statt eines eindeutigen Ergebnisses für eine Messung ein Ergebnisraum definiert wird. Ist das der Fall, werden diese Messergebnisse “unsicher” genannt. Die Speicherung von unsicheren Daten in einer relationalen Datenbank ist allerdings konzeptionell nicht korrekt umzusetzen, weshalb neue Speicherungskonzepte für unsichere Daten entwickelt werden mussten.

Die Entwicklung dieser Konzepte ist mittlerweile ein viel erforschtes Feld im Bereich der Datenbanken. Diverse Prototypen und Projekte haben den Versuch unternommen, effizient und korrekt unsichere Daten repräsentieren zu können.

1.2 Ziel der Arbeit

In dieser Arbeit sollen zwei Frameworks, namentlich MayBMS und Trio, evaluiert werden, die für den Umgang mit unsicheren Daten konzeptioniert wurden. Dabei soll die Evaluation durch Experimente gestützt und begründet werden. Die Evaluation soll vergleichend durchgeführt werden, so dass die beiden Frameworks direkt gegenüber gestellt werden können. Die Evaluation soll dann eine Grundlage für eine Bewertung bieten, ob eines der Frameworks produktiv für reale Zwecke sinnvoll einsetzbar ist, und falls nein, aus welchen Gründen.

1.3 Struktur der Arbeit

Im ersten Teil der Arbeit werden Grundlagen zum Thema Datenbanken vermittelt. Dabei werden Konzepte und Begrifflichkeiten geklärt, sowie die unterschiedlichen Operatoren der relationalen Algebra beleuchtet.

Danach wird das Thema der unsicheren Daten aufgegriffen. Hier wird erläutert, was unsichere Daten überhaupt sind und wie diese konzeptionell aussehen. Außerdem wird die Semantik der möglichen Welten erklärt, welche ein elementares Konzept beider evaluierten Frameworks darstellt. Des weiteren wird erklärt, was ein Repräsentationsmodell ist.

In den nächsten beiden Kapiteln folgen Beschreibungen der evaluierten Frameworks. Dabei wird zuerst Trio und dann MayBMS vorgestellt. Beide Kapitel enthalten neben generellen Systembeschreibungen Beschreibungen zu den Speicherungskonzepten als auch Beschreibungen der jeweiligen Query-Languages.

Nach der Beschreibung der Frameworks folgt die Evaluation. Dabei wird der Experimentaufbau beschrieben, bevor anhand von definierten Kriterien beide Frameworks analysiert werden. Damit werden Hypothesen beurteilt, die die Frameworks in Relation zu einander setzen.

Zum Schluss wird ein Fazit gezogen und die wichtigsten Aspekte der Arbeit noch einmal zusammengefasst.

2 Grundlagen zu Datenbanken

Die Aufgabe der in dieser Arbeit evaluierten Frameworks liegt in der Speicherung und Verarbeitung von Daten. Um weitergehende Technologien aber ausreichend verstehen zu können, müssen begriffliche und konzeptionelle Grundlagen gelegt werden, die die Basis dafür bereit stellen.

2.1 Daten

Der wichtigste Begriff, der dabei geklärt werden muss, ist der Ausdruck "Daten". Für den Begriff "Daten" gibt es bereichsbezogen mehrere Definitionen. Informationstechnisch betrachtet ist aber eine geläufige Definition, dass Daten "kontextfreie Angaben, die aus interpretierten Zeichen bzw. Signalen bestehen" [40] sind. Daten werden durch kontextbezogene Interpretation zu Informationen, die weiter verarbeitet werden können. Daten können initial einen beliebigen Typ haben. So können damit, je nach Art der Datenbank, beispielsweise Texte, ein Datum, Wahrheitswerte oder Zahlen beliebiger Zahlensysteme, aber auch komplexe Objekte wie JSON-Dokumente beschrieben werden.

2.2 Datenbanken

Eine Datenbank (DB) ist eine "organisierte Sammlung von strukturierten Informationen oder Daten, die typischerweise elektronisch in einem Computersystem gespeichert sind" [27]. Diese Sammlungen werden gespeichert in einem Datenbank-Managementsystem (DBMS), welches mehrere Datenbanken beinhalten kann. Die Art der Organisation dieser Sammlungen ist auch abhängig von der Technologie des Datenbanksystems. Generell werden Datenbanksysteme in zwei Kategorien aufgeteilt:

- SQL-Datenbanken: Datenbanksysteme, die nach dem relationalen Modell aufgebaut sind (siehe Kapitel 2.3)
- NoSQL-Datenbanken: Andere Datenbanktechnologien, die nicht nach dem relationalen Modell aufgebaut sind ("NoSQL": "Not only SQL"), wie zum Beispiel Key-Value-Datenbanken, Dokumentbasierte Datenbanken oder Graph-Datenbanken [25].

Die NoSQL-Datenbanken sollen in dieser Thesis nicht weiter behandelt werden.

2.3 Relationale Datenbanken

Datenbanken, die unsichere Daten organisieren, sind oft auf dem relationalen Modell aufgebaut. Aus diesem Grund sollen an dieser Stelle Grundlagen zu relationalen Datenbanken geschaffen werden, um dem weiteren Verlauf der Arbeit folgen zu können.

Wenn von relationalen Datenbanken die Rede ist, wird sich dabei auf die mathematische Definition einer Relation gestützt. Mathematische Relationen setzen Elemente aus Mengen in Beziehungen zu einander. Für n Mengen an Daten A_1 bis A_n, kann eine n-stellige Relation R wie folgt definiert werden [14]:

$$A_1 \times \cdots \times A_n = \{a_1 \in A_1, \ldots, a_n \in A_n\} = \prod A$$

$$R \subseteq \prod A$$

Das bedeutet, dass eine Relation R eine Menge von n-Tupeln ist, die eine Untermenge des kartesischen Produktes aller einzelnen Mengen A_1 bis A_n zusammen bildet. Relationen in Datenbanken können auch als Tabellen dargestellt werden. Zur Veranschaulichung der Begriffsdefinitionen ist in Abbildung 2.1 eine Relation als Tabelle dargestellt. In dieser Abbildung ist außerdem die Nomenklatur aufgetragen.

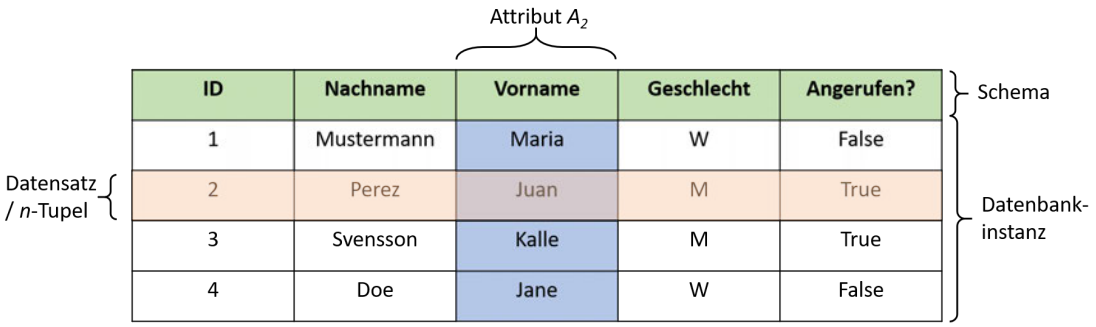

Abbildung 2.1: Relation "Rückrufe"

Jeder Datensatz stellt ein Tupel dar, welches in der Relation enthalten ist. Diese Datensätze bestehen aus mehreren Werten, welche in Spalten angeordnet sind. Jeder Datensatz in der Tabelle hat die gleiche Spaltenanordnung. Jeder Datensatz beschreibt eine Entität. Eine Entität wird beschrieben durch mehrere Eigenschaften, Attribute genannt. Jedes Attribut hat wiederum einen eigenen Wertebereich, der festlegt, welche Werte ein Eintrag an dieser Stelle annehmen kann. Dieser Wertebereich wird die "Domäne" dieses Attributs genannt. [22]. Die Attributnamen sind die Spaltentitel der Tabelle. Wie diese Attribute angeordnet sind, ist in der Relation fest definiert. Diese Definition nennt man das Schema der Relation.

Das Modell der relationalen Datenbanken wurde 1970 erstmals definiert. Nach dieser Definition von E.F. Codd [10] müssen die Daten innerhalb einer Relation folgende Eigenschaften aufweisen:

1. Jeder Datensatz repräsentiert ein n-Tupel aus R.
2. Die Reihenfolge der einzelnen Datensätze ist irrelevant
3. Jeder Datensatz ist einzigartig innerhalb von R.
4. Die Anordnung der Attribute innerhalb eines Datensatzes ist wichtig — sie korrespondiert zur Reihenfolge der Teilmengen in der Definition von R
5. Die Bedeutung einer Spalte innerhalb des Datensatzes wird gekennzeichnet durch den Namen der entsprechenden Quellmenge.

2.4 Queries

Um Daten aus einer Quelle zu extrahieren, werden auf eine Quelle Abfragen durchgeführt. Diese Abfragen werden auch Queries genannt. Die Definition dieser Abfragen basiert auf strukturellen Regeln, die zu verschiedensten komplexen Abfragen kombiniert werden können. Alle diese Regeln bilden zusammen eine Sprache, die Query Language genannt wird. Die Varianz der Quellen, für die Query Languages konzipiert wurden, ist groß. Dazu gehören natürlich Datenbanken, aber zu Beispiel auch XML-Dokumente (XQuery [39]) oder Suchmaschinen wie Google und Bing. Die Sprache variiert von Datenquelle zu Datenquelle. Im Allgemeinen sind die Regeln für Relationale Datenbanken gleich. Die Sprache für relationale Datenbankabfragen wird SQL genannt, was die Abkürzung für "Strcutured Query Language" ist [6]. Wenn auch die Semantik der zugrunde liegenden Regeln sich gleicht, so kann die Syntax sich bei verschiedenen DBMS, die beide relationale Datenbanken verwalten, trotzdem unterscheiden.

2.5 Relationale Algebra

Im Verlauf dieser Thesis werden mehrfach Queries beschrieben, die auf eine jeweils zugrunde liegende Datenbank ausgeführt werden. Um diese zu verstehen, werden im folgenden die grundlegenden Operatoren von Datenbankabfragen und Kombinationen unter ihnen am Beispiel von der relationalen Algebra beschrieben. Diese wurde auch von Codd definiert [10], damit Zugriffe auf Informationen in Relationen auch korrekt und eindeutig in ihrem Verhalten und ihren Auswirkungen definiert sind. Außerdem werden dazu noch die zugehörigen SQL-Befehle dargestellt, damit ein Überblick über die Standard-SQL-Syntax gegeben werden kann.

Dazu werden die Relationen aus den Tabellen 2.1 und 2.2 als Beispieldaten verwendet. Dabei bildet die Relation $H = Hält < Vorlesung, Dozent, Raum >$ eine Liste von Vorlesungen ab. Diese haben jeweils einen Titel, einen Dozenten und einen Raum, in dem sie statt finden. Die Relation $V = Vorlesungsteilnehmer < Student, Semester, Vorlesung >$ zeigt eine Liste von Studenten, die an Vorlesungen teilnehmen. Dabei wird jeder Student mit dem Fachsemester gespeichert, in dem er diese Vorlesung besucht hat.

Vorlesung	**Dozent**	**Raum**
Grundlagen der Informatik	Kai von Luck	NB1.11
Datenbanken	Olaf Zukunft	10.60
Datenbanken	Thomas Thiel-Clemen	10.60
Softwareengineering 1	Stefan Sarstedt	10.60

Tabelle 2.1: Relation $H = Hält$

Teilnehmername	**Semester**	**Vorlesung**
Adam Apfel	1	Grundlagen der Informatik
Adam Apfel	2	Datenbanken
Berta Brot	2	Datenbanken
Claudius Cantz	3	Sofwareengineering 1
Dörte Drehwurm	3	Datenbanken

Tabelle 2.2: Relation $V = Vorlesungsteilnehmer$

2.5.1 Selektion

Der simpelste Operator der relationalen Algebra ist die Selektion σ. Diese selektiert Tupel aus einer Relation R. Eine einfache Selektion $\sigma(R)$ liefert dabei erst einmal alle Tupel

in der Relation als Ergebnismenge. Durch Kombination aus Bedingungen mit logischen Operatoren, als φ bezeichnet, kann eine Selektion $\sigma_{\varphi}(R)$ aber beschränkt werden, so dass nur die Tupel selektiert werden, die die Bedingungen erfüllen. Eine Bedingung sind dabei zwei Attribute oder ein Attribut und ein atomarer Wert, die mit binären Vergleichsoperatoren ($<$,$\leq$,$=$,$\neq$,$\geq$ oder $>$) miteinander verglichen werden.

Wenn beispielsweise alle Vorlesungsteilnehmer selektiert werden sollen, die mindestens im zweiten Semester studieren, sähe das Ergebnis der Abfrage $Q1 = \sigma_{Semester \geq 2}(V)$ aus wie in Tabelle 2.3.

Teilnehmername	**Semester**	**Vorlesung**
Adam Apfel	2	Datenbanken
Berta Brot	2	Datenbanken
Claudius Cantz	3	Sofwareengineering 1
Dörte Drehwurm	3	Datenbanken

Tabelle 2.3: Ergebnis der Abfrage Q1

```
1 SELECT *
2 FROM Vorlesungsteilnehmer
3 WHERE Semester >= 2;
```

Listing 2.1: SQL Syntax für Abfrage Q1

2.5.2 Projektion

Eine Projektion $\pi_L(R)$ (alternativ auch $\Pi_L(R)$) ist nach Codd dafür verantwortlich, nur bestimmte Attribute aus einer Relation als Ergebnis zu liefern. Die Menge L ist hierbei eine Menge von Attributen aus R, welche im Ergebnis existieren sollen. Die anderen in R enthaltenen Attribute werden im Ergebnis nicht erscheinen. Dadurch verringert sich nicht nur die Anzahl der Spalten in der tabellarischen Darstellung des Ergebnisses, sondern auch die Anzahl der Zeilen, weil eventuelle Duplikate dabei entfernt werden.

Wenn als Beispiel aus unseren gegebenen Relationen abgefragt werden soll, welche Vorlesungen die Teilnehmer besuchen, die mindestens im zweiten Semester Studieren, dann wird eine Untermenge aus Tabelle 2.3 gebildet, indem die Operatoren kombiniert werden. Das Ergebnis der Abfrage $Q2 = \pi_{Vorlesung}(\sigma_{Semester \geq 2}(V))$ findet sich dargestellt in Tabelle 2.4. Dabei ist der Wert "Datenbanken" zwar theoretisch mehrfach im Ergebnis enthalten, weil aber Duplikate entfernt werden, taucht dieser nur einmal auf.

Vorlesung
Datenbanken
Sofwareengineering 1

Tabelle 2.4: Ergebnis der Abfrage Q2

```
SELECT Vorlesung
FROM Vorlesungsteilnehmer
WHERE Semester >= 2;
```

Listing 2.2: SQL Syntax für Abfrage Q2

2.5.3 Rename

Ein Rename $\rho_{a/b}(R)$ sorgt dafür, dass im Ergebnis einer Abfrage Attributsnamen geändert werden können. Dabei sind a und b Namen von Attributen und R eine Relation. Im Ergebnis der Abfrage wird nun der Name von b durch den Namen von a ersetzt.

Als Beispiel sei das Ergebnis von Q2 betrachtet. Der Attributsname "Vorlesung" könnte hier irreführend sein und "Vorlesungstitel" wäre eine bessere Alternative. Daher soll dieser ersetzt werden. Das Ergebnis einer Abfrage $Q3 = \rho_{Vorlesungstitel/Vorlesung}(Q2)$ ist in Tabelle 2.5 dargestellt.

Vorlesungstitel
Datenbanken
Sofwareengineering 1

Tabelle 2.5: Ergebnis der Abfrage Q3

```
SELECT Vorlesung AS Vorlesungstitel
FROM Vorlesungsteilnehmer
WHERE Semester >= 2;
```

Listing 2.3: SQL Syntax für Abfrage Q3

2.5.4 Join

Ein Join $R \bowtie S$ (alternativ auch $R * S$) ist eine Kombination zweier Relationen R und S zu einer Ergebnismenge, die die Informationen beider Relationen enthält. Dabei ist wichtig, dass nicht alle Relationen beliebig zu joinen sind. Voraussetzung dafür ist, dass

beide Tabellen ein Attribut U besitzen, so dass sowohl $\pi_U(R)$ als auch $\pi_U(S)$ existieren. Der einfachste Join ist dabei der natürliche Join, der alle möglichen Kombinationen der Verbindungen enthält.

Ein Beispiel für ein Ergebnis eines Joins ist in Tabelle 2.6 aufgeführt. Dabei wird die Abfrage $Q4 = \rho_{S/Semester}(\pi_{Dozent,Raum,Vorlesung}(H) \bowtie \pi_{Vorlesung,Teilnehmername,Semester}(V))$ mit dieser Tabelle beschrieben.

Dozent	Raum	Vorlesung	Teilnehmername	S
Kai von Luck	NB1.11	Grundlagen der Informatik	Adam Apfel	1
Olaf Zukunft	10.60	Datenbanken	Adam Apfel	2
Olaf Zukunft	10.60	Datenbanken	Berta Brot	2
Olaf Zukunft	10.60	Datenbanken	Dörte Drehwurm	3
Thomas Thiel-Clemen	10.60	Datenbanken	Adam Apfel	2
Thomas Thiel-Clemen	10.60	Datenbanken	Berta Brot	2
Thomas Thiel-Clemen	10.60	Datenbanken	Dörte Drehwurm	3
Stefan Sarstedt	10.60	Softwareengineering 1	Claudius Cantz	3

Tabelle 2.6: Ergebnis der Abfrage Q4

```
1 SELECT Dozent, Raum, v.Vorlesung, Teilnehmername, Semester AS S
2 FROM Vorlesung v Join Haelt h
3 On v.Vorlesung = h.Vorlesung
4 WHERE Semester >= 2;
```

Listing 2.4: SQL Syntax für Abfrage Q4

3 Unsichere Daten

Nachdem definiert wurde, was Daten sind und die grundlegenden Operationen und Konzepte von relationalen Datenbanken erläutert wurden, wird in diesem Abschnitt der Fokus auf den Bereich der unsicheren Daten gelegt.

3.1 Unsichere Daten

Bei Betrachtung und Vergleich diverser häufig eingesetzter Datenbanksysteme fällt der Begriff “Unsicherheit” selten bis gar nicht. Unsichere Daten sind Daten, bei denen die Existenz in einer Datenbank nicht vollständig sichergestellt ist. Datenbanken, die unsichere Daten enthalten, werden “unvollständig” genannt. Unsicherheit von Daten wird im Allgemeinen in zwei Arten unterschieden [34]:

- *Tupel-Level-Unsicherheit*: Es kann nicht sicher gesagt werden, ob ein beliebiger Datensatz A in einer Datenbankinstanz enthalten ist. Dieser ist mit einer Variable A beschreibbar, die eine boolesche Wertemenge besitzt: Bei Existenz ist das Wert $wahr$, bei Nichtexistenz ist der Wert $falsch$. Diese Tupel werden auch "Maybe-Tupel"genannt.

- *Attribut-Level-Unsicherheit*: Innerhalb eines Datensatzes kann ein beliebiges Attribut unsicher sein. Das bedeutet, dass der Wert des Attributs durch eine Variable B beschrieben wird. Dabei ist B die Menge aller Werte, die dieses Attribut annehmen kann.

Ein Beispiel dafür ist in Tabelle 3.1 aufgeführt:

Ein Vater befindet sich im Herbst mit seinem Kind im Park, um Vögel zu beobachten. Die Beobachtungen werden in einer simplen Tabelle aufgetragen, bei der die Uhrzeit, die Vogelart und der Beobachter aufgeführt sind. Um 11:30 meint der Vater einen Spatz zu erkennen und trägt diese Beobachtung als Datensatz A ein. Danach bemerkt er, dass seine Brille nicht ganz sauber ist. Infolgedessen ist er sich nicht mehr sicher, ob er wirklich einen Spatz gesehen hat, oder ob dies nur ein herab fallendes Blatt war. Damit ist der Datensatz X auf Tupel-Level unsicher.

$x = \{Blaumeise, Kohlmeise\}$

	Uhrzeit	Vogelart	Beobachter	
A	11:30	Spatz	Vater	?
	11:35	Blaumeise	Vater	
	11:37	Spatz	Kind	
	12:00	Amsel	Kind	
	12:03	Spatz	Vater	
	12:15	Elster	Kind	
B	11:42	x	Kind	

Tabelle 3.1: Tabelle mit Vogelbeobachtungen

Auf dem Heimweg erzählt das Kind ihm, dass es vergessen hat, eine Beobachtung zu melden, die es gemacht hat. Bei der Uhrzeit ist sich das Kind noch sicher, aber bei der Vogelart kann es sich nur noch daran erinnern, dass es eine Meise war. Nicht aber, ob es eine Blau- oder eine Kohlmeise gewesen ist. Trotzdem wird diese Beobachtung in der Tabelle als Datensatz B erfasst. Hier wird als Wert für die Vogelart eine Variable x eingetragen. Diese Variable repräsentiert einen Wert aus der Wertemenge aus den beiden möglichen Arten "Blaumeise" und "Kohlmeise". Damit ist der Datensatz B auf Attribut-Level unsicher.

3.2 Semantik der möglichen Welten

Die Repräsentation von unsicheren Daten in Datenbanken ist mit einer einfachen relationalen Datenbank nicht endgültig abgedeckt. In relationalen Datenbanken werden im Allgemeinen atomare Werte gespeichert. Im Falle der Attribut-Level-Unsicherheit wird aber mit einer Variable ein nicht-atomarer Datentyp als Wert gespeichert. Im Falle der Tupel-Level-Unsicherheit ist nicht sicher, ob ein Tupel überhaupt enthalten ist.

Daher wurde für Relationen mit unsicheren Daten das Modell der möglichen Welten adaptiert [41, 2]. Hierbei handelt es sich um ein Modell, welches seinen Namen aus der Philosophie entlehnt. Dort gibt es die Theorie, dass zur gleichen Zeit mehrere "Welten" existieren, die gleichzeitig jeweils unterschiedliche Stati einer zusammengefassten Realität abbilden. Dabei ist ein Status korrekt, aber es ist nicht bekannt, welcher [20].

Im Kontext von Datenbanken mit unsicheren Daten bedeutet das, dass eine Relation R, die unsichere Daten beinhaltet, eigentlich eine Kombination aller ihrer möglichen Zustände repräsentiert. Ein Zustand wird oft auch mit einer Belegung ν verbunden. Dabei beschreibt ν die Übersetzung von Variablen in Werte. Die Menge aller Zustände wird nicht durch die Relation R selbst, sondern durch eine Funktion $REP(R)$ abgebildet.

Dies soll am Beispiel der Relation $RV =< Uhrzeit, Vogelart, Beobachter >$ aus Tabelle 3.1 erläutert werden.

Um die möglichen Welten darzustellen, werden alle unsicheren Datensätze in jeder ihrer Varianten mit allen anderen Varianten der unsicheren Datensätze permutiert.

Bei unsicheren Datensätzen auf Tupel-Ebene gibt es nur zwei Varianten. Diese sind entweder in der Datenbank enthalten oder nicht, und haben deshalb genau 2 Repräsentationen. So muss für jeden dieser Datensätze jede Mögliche Welt eine dieser beiden Varianten abdecken. Im Beispiel in Tabelle 3.2 wird das Tupel A dafür betrachtet. So ist in Tabelle 3.2a und Tabelle 3.2c dieses Tupel mit seiner regulären Belegung enthalten. In den Tabellen 3.2b und 3.2d hingegen ist das Tupel A nicht enthalten.

Bei unsicheren Datensätzen auf Attribut-Level ist die Anzahl der möglichen Varianten abhängig von der Wertemenge der unsicheren Attribute. Pro Attribut gibt es genau so viele Varianten, wie jeweils Elemente in der Wertemenge enthalten sind. Dies kann exemplarisch für das Tupel B betrachtet werden. hier ist das Attribut x unsicher und hat die Wertemenge $\{Blaumeise, Kohlmeise\}$. Daher gibt es vier verschiedene Belegungen für ν, nämlich folgende:

1. $\nu_1 = \{? = wahr, x = Blaumeise\}$
2. $\nu_2 = \{? = wahr, x = Kohlmeise\}$
3. $\nu_3 = \{? = falsch, x = Blaumeise\}$
4. $\nu_4 = \{? = falsch, x = Kohlmeise\}$

Für Tupel B gibt es daher in Tabellen 3.2a und 3.2c die Variante mit "Blaumeise" und in Tabellen 3.2b und 3.2d die Variante mit "Kohlmeise".

	Uhrzeit	Vogelart	Beobachter
A	11:30	Spatz	Vater
	11:35	Blaumeise	Vater
	11:37	Spatz	Kind
	12:00	Amsel	Kind
	12:03	Spatz	Vater
	12:15	Elster	Kind
B	11:42	Blaumeise	Kind

(a) Mögliche Welt für ν_1

	Uhrzeit	Vogelart	Beobachter
A	11:30	Spatz	Vater
	11:35	Blaumeise	Vater
	11:37	Spatz	Kind
	12:00	Amsel	Kind
	12:03	Spatz	Vater
	12:15	Elster	Kind
B	11:42	Kohlmeise	Kind

(b) Mögliche Welt für ν_2

	Uhrzeit	Vogelart	Beobachter
	11:35	Blaumeise	Vater
	11:37	Spatz	Kind
	12:00	Amsel	Kind
	12:03	Spatz	Vater
	12:15	Elster	Kind
B	11:42	Blaumeise	Kind

(c) Mögliche Welt für ν_3

	Uhrzeit	Vogelart	Beobachter
	11:35	Blaumeise	Vater
	11:37	Spatz	Kind
	12:00	Amsel	Kind
	12:03	Spatz	Vater
	12:15	Elster	Kind
B	11:42	Kohlmeise	Kind

(d) Mögliche Welt für ν_4

Tabelle 3.2: Mögliche Welten von R, $REP(< Uhrzeit, Vogelart, Beobachter >)$

Die Anzahl der möglichen Welten für eine unvollständige Datenbank ist unter gewissen Annahmen mathematisch beschreibbar. Diese Annahmen betreffen die Korrektheit eines Faktes und damit die Interpretation einer unvollständigen Datenbank. Wenn angenommen wird, dass jeder Eintrag der Datenbank, der möglicherweise existieren kann, explizit in der Datenbank gelistet sein muss, sähe die Darstellung der möglichen Welten für Tabelle 3.1 genauso aus, wie in Tabellen 3.2. Dann ist die Anzahl der möglichen Welten wie folgt berechenbar:

$$\mid REP(R) \mid = 2^t \cdot a_1^{|dom(a_1)|} \cdot \ldots \cdot a_n^{|dom(a_n)|}$$

- $REP(R)$ ist die Menge aller möglichen Welten, die eine Relation R repräsentiert, die unsichere Daten enthält
- t ist die Anzahl der Maybe-Tupel
- a_1 bis a_n sind alle Werte für unsichere Attribute mit ihren entsprechenden Domänen $dom(a_1)$ bis $dom(a_n)$

3.3 Open-World-Assumption und Closed-World-Assumption

Die in Abschnitt 3.2 angegebene Formel zur Berechnung der Anzahl möglicher Welten für eine unvollständige Datenbank gilt nur für bestimmte Annahmen konzeptioneller Natur. Dahinter steht die Frage nach der Korrektheit eines Faktes.

Bei der Betrachtung einer Datenbank wird oft einfach angenommen, dass alle Daten, die darin enthalten sind, korrekt sind. Implizit wird aber dadurch gleichzeitig definiert, dass nicht enthaltene Informationen falsch sind. Diese implizite Annahme der Inkorrektheit nicht enthaltener Daten wird "Closed-World-Assumption" (im folgenden mit "CWA" abgekürzt) genannt [32]. Auch bei unvollständigen Datenbanken kann diese Annahme getroffen werden. Wenn bei diesen dann die möglichen Welten aufgezeigt werden, enthält die Repräsentationsfunktion genau die Anzahl an Elementen, die mit der angegebenen Formel berechnet werden kann.

Die Annahme der CWA setzt aber auch einen Allwissenheitszustand voraus, der bei Betrachtung einer Datenbank gekennzeichnet werden muss. Die CWA geht vor allem davon aus, dass jede Wert jedes Attributs vollständig bekannt ist. Dies ist gerade bei unsicheren Daten aber nicht gegeben. Daher gibt es noch eine zweite Annahme, die so genannte "Open World Assumption" (im folgenden mit "OWA" abgekürzt). Diese besagt, genau wie die CWA auch, dass eine Information, die in einer Relation enthalten ist, definitiv korrekt ist. Im Gegensatz zur CWA geht die OWA aber nicht von Allwissenheit aus. Das bedeutet, dass Informationen, die nicht explizit in der Relation enthalten sind, nicht implizit falsch sind, sondern auch richtig sein könnten. Daraus resultiert für die Repräsentationsfunktion für mögliche Welten unter der OWA analog nach [41] folgendes:

$$\forall w \in REP(< Uhrzeit, Vogelart, Beobachter >) : w =< Uhrzeit', Vogelart', Beobachter' >$$

$$Uhrzeit' \subseteq Uhrzeit \vee Vogelart' \subseteq Vogelart \vee Beobachter' \subseteq Beobachter$$

Damit bilden die Tabellen 3.2 nur eine Teilmenge der möglichen Welten ab.

3.4 Repräsentationssysteme

Wie bereits in Abschnitt 3.2 erwähnt, wird bei unvollständigen Datenbanken auch die Repräsentationsfunktion verwendet. Systeme, die die einzelnen Welten die in dieser Funktion enthalten sind repräsentieren, werden Repräsentationssysteme genannt. Im Laufe der

Zeit wurden verschiedene Repräsentationssysteme definiert. Ein Repräsentationssystem τ verwendet immer eine bestimmte Sprache, wie zum Beispiel die relationale Algebra.

Wie exakt die Sprache im Repräsentationssystem integriert ist, unterscheidet sich von System zu System. Diese Exaktheit wird auch Stärke eines Repräsentationssystems τ genannt. Diese wird für eine Tabelle T aus τ und eine Sprache ζ formal wie folgt definiert [16]:

- Sei q eine Query aus ζ
- $q(REP(T)) = \{q(I_1), \ldots, q(I_n) | \forall I : I \in REP(T)\}$
- Eine Tabelle T' repräsentiert $q(REP(T))$
- $REP(T') = q(REP(T))$
- Wenn diese Voraussetzungen erfüllt sind und T eine Tabelle aus τ bedeutet das: τ ist ein starkes Repräsentationssystem fürζ

Wenn in einem Repräsentationssystem eine Query auf eine Tabelle angewendet liefert das Ergebnis eine Tabelle. Wenn die gleiche Query auf die Repräsentationsfunktion der Tabelle angewendet wird, sollte das Ergebnis davon eine Menge von Tabellen zurückliefern. Die Elemente dieser Menge sind die Ergebnisse, die man erhält, wenn man q auf jede einzelne Welt aus $REP(T)$ anwendet. Wenn diese Menge als eine Tabelle T' des Repräsentationssystem repräsentiert wird, muss die Repräsentationsfunktion $REP(T')$ sich gleichen mit $q(REP(T))$. Sind diese Voraussetzungen erfüllt, wird ein Repräsentationssystem als starkes Repräsentationssystem für eine Sprache ζ bezeichnet.
Wenn diese Bedingungen allerdings gelockert werden, kann ermittelt werden, welche Tupel einer Abfrage $q(T)$ mit Sicherheit im Ergebnis enthalten sind [1]. Dies bedeutet formal folgendes:

$$E = \{q(I_1, \ldots, I_n | \forall I : I \in REP(T)\}$$

$$sure(q, T) = \cap E$$

Es gibt für eine Abfrage q aus einer Sprache ζ auf eine Tabelle T aus τ eine Ergebnismenge E. Die Menge aller sicherer Tupel ist die Schnittmenge aller in E enthaltenen Mengen. Als nächstes soll die ζ-Äquivalenz eingeführt werden. Zwei Datenbanken A und B sind ζ-Äquivalent, wenn die Menge ihrer sicheren Tupel sich gleichen. Also gilt:

$$A \equiv_\zeta B \Leftrightarrow \forall q \in \zeta : sure(q, A) = sure(q, B)$$

Nun wird als Lockerung der Bedingungen die exakte Gleichheit der Ergebnismengen für starke Repräsentationssysteme ersetzt durch ζ-Äquivalenz, also gilt:

$$REP(T') \equiv_\zeta q(REP(T))$$

Wenn dies für ein Repräsentationssystem τ und eine Sprache ζ zutrifft, wird ein Repräsentationssystem τ als schwaches Repräsentationssystem für ζ bezeichnet.

3.5 Unvollständigkeit von Daten in Codd-Tabellen

Das relationale Modell von Codd wurde erst von Codd selbst im Jahre 1979 für "NULL"-Werte erweitert. Dabei galt der Wert "NULL" als Synonym für sowohl die Bedeutung "Wert momentan nicht bekannt" als auch für die Bedeutung "Eigenschaft nicht anwendbar" [11]. Außerdem wurde der Wert "NULL" mit einem Symbol substituiert. Im Folgenden wird dafür "@" verwendet. Bei Codd-Tabellen bedeutet das @-Symbol, dass die Werte alle deutlich unterscheidbar sind, wenn sie auch die gleiche Belegung haben können. Dies soll am Beispiel der Relation RC in Tabelle 3.3 dargestellt werden.

A	B	C
1	@	2
2	3	4
2	@	5

(a) Relation RC mit Null-Werten

A	B	C
1	x	2
2	3	4
2	y	5

(b) Relation RC mit eindeutigen Bezeichnungen

Tabelle 3.3: Relation R als Beispiel einer Codd-Tabelle

Codd-Tabellen sollen ein Repräsentationssystem für die relationale Algebra darstellen. Diese können Teile der Relationalen Algebra abbilden. Bei Projektionen werden Duplikate entfernt und keine Filterung im Attribut vorgenommen. Daher werden für Variablenwerte, die bereits an einer anderen Stelle im Attribut enthalten sind, die Einträge in der Reihe übergangen. Variablenwerte, die noch nicht in einer anderen Stelle im Attribut enthalten sind, werden wiederum explizit im Ergebnis aufgeführt. Bei Rename-Operationen ist das Prinzip das gleiche. Daher sind Codd-Tabellen eine starkes Repräsentationssystem für PR-Abfragen der relationalen Algebra. Selektionen sind allerdings nicht eindeutig mit ihnen abbildbar. Als Beispiel sei die Tabelle 3.3b verwendet. Darauf soll eine Query $q = \sigma_{B=1}(RC)$ angewendet werden. Für eine Belegung $\nu_1 = \{x, y = 1\}$ würde ein mögliches Ergebnis das Tupel (2,1,5) enthalten. Für eine Belegung $\nu_2 = \{x, y = 3\}$

wäre ein mögliches Ergebnis eine leere Menge. Es kann aber keine Codd-Tabelle geben, die gleichzeitig beide möglichen Ergebnisse repräsentiert. Daher sind Codd-Tabellen kein starkes Repräsentationssystem für Selektionen. In allen Ergebnissen einer Selektion ist allerdings mit Sicherheit ein leeres Tupel enthalten und das Ergebnis daher mindestens eine leere Menge. Daher sind Codd-Tabellen ein schwaches Repräsentationssystem für SPR-Queries. Dieses gilt allerdings nicht für Join-Operationen. Um das zu zeigen, sei Tabelle 3.4 betrachtet, wobei a, $a\prime$, c und $c\prime$ Konstanten darstellen und x und $x\prime$ Variablen.

A	**B**	**C**
a	x	c
a'	x'	c'

Tabelle 3.4: Relation RJ als Beispiel für Join-Operationen

Weitere Annahmen sind:

$$q_1 = \pi_{A,C}(RJ) \bowtie \pi_B(RJ)$$

$$q_2 = \pi_{A,C}(\pi_{A,B}(RJ) \bowtie \pi_{B,C}(RJ))$$

Angenommen, es gäbe eine Tabelle S, für die gilt $REP(S) \equiv_{SPJ} q(REP(RJ))$. Dann ist $sure(q_1 \circ q_2, RJ)$ in Tabelle 3.5 dargestellt.

A	**C**
a	c
a'	c'
a'	c
a	c'

Tabelle 3.5: Ergebnis von $sure(q_1 \circ q_2, RJ)$

Folgernd aus der Definition eines schwachen Repräsentationssystems muss $sure(q', W)$ gleich sein. Da das Tupel (a', x) für jede Belegung der Variablen ν in $sure(q', W)$ enthalten ist, muss es aufgrund der Definition des Join-Operators Tupel $u, v \in W$ geben, für die gilt:

$$u(A) = a'$$

$$v(C) = c$$

$$\nu(u)(B) = \nu(v)(B)$$

Sei ν eine Belegung bei der $\nu(y) \neq \nu(z)$ für alle Variablen $y, z, y \neq z$ gilt.
Wenn $u = v$ ist, dann muss $u(A) = a'$ und $u(C) = c$ gelten, daher muss $(a', c) \in sure(\pi_{AC}(R), W)$ sein. Da aber $(a', c) \notin sure(\pi_{AC}(R), q_1(REP(RJ))$ ist, entsteht hier ein Widerspruch.
Da $\nu(u)(B) = \nu(v)(B)$ ist und W keine sich wiederholenden Variablen beinhaltet, lässt sich daraus folgern, dass $u(B)$ und $v(B)$ durch jeweils eine Konstante k definiert sein müssen, wenn $u \neq v$ ist. Dann müsste aber ein Tupel $(a', k) \in sure(\pi_{AB}(R), W)$ existieren. Da aber $sure(\pi_{AB}(R), q(REP(RJ)))) = \emptyset$ ist, herrscht auch hier ein Widerspruch.
Da weder $u = v$ noch $u \neq v$ gelten kann, lässt sich feststellen, dass Codd-Tabellen nicht einmal ein schwaches Repräsentationssystem für SPJ-Operationen darstellen. Ein Beweis auf Basis ähnlicher Grundlage lässt sich auch für SPU-Operationen anstellen, dies soll aber hier nicht weiter geführt werden. Dazu wird auf [16] verwiesen.

4 Trio

Nach dem Schaffen der nötigen Grundlagen zum Verstehen der Konzepte der Frameworks, wird in diesem Abschnitt das erste Framework, nämlich Trio, vorgestellt.

4.1 Überblick

Das erste Framework, das in dieser Thesis evaluiert werden soll, ist das Trio-DBMS. Das Trio-DBMS ist ein Projekt der Stanford University unter der Leitung von Jennifer Widom gewesen, dessen Entwicklung Mitte der 2000er Jahre begann. Ziel war es, ein möglichst verständliches System zu bieten, um unsichere Daten zu speichern und zu verarbeiten. In [33] wird dies näher erklärt. Bisherige Modelle seien unter gewissen relationalen Operatoren nicht komplett oder sogar geschlossen. Modelle, die komplett und/oder geschlossen sind, seien wiederum sehr komplex und daher schwierig zu verstehen. Daher soll mit dem dem Trio-DBMS zugrunde liegenden Modell diese Brücke geschlagen und ein leichtes Verständnis möglichst bei Komplettheit und Geschlossenheit ermöglicht werden. Der erste open-source-Release eines Prototyps erfolgte im Juni 2007, seitdem sind nur noch wenige Änderungen geschehen [36].

Als Grundlage zum Speichern von unsicheren Daten werden in diesem Modell ULDBs verwendet. ULDB steht für "Uncertainty-Lineage-Database". Dies sind Datenbanken, die grundlegend auf dem relationalen Modell basieren. Zusätzlich wird das relationale Modell allerdings noch um die Konzepte von Unsicherheit und Data-Lineage erweitert. Zusätzlich wurde eine eigene Query-Language namens TriQL definiert und implementiert. Auch hier wird auf den Operatoren der relationalen Algebra aufgesetzt, diese wird aber auch um einige Funktionen erweitert.

4.2 Umfang und Systemvoraussetzungen

Der letzte Release des Trio Frameworks ist im Jahre 2009 veröffentlicht worden. Darin enthalten sind sowohl die zugrunde liegende Datenbank-Technik, als auch zwei User-Interfaces. Das erste davon ist "trioplus", ein Kommandozeilen-Interface für das Trio-

DBMS. Außerdem wird der "TrioExplorer" mit installiert, der graphische Darstellungen der Datenbank-Schemata und weitere Funktionen erlaubt und ein graphisches User-Interface bietet.

Das Trio-System ist in der Programmiersprache Python implementiert. Daher muss auf dem Zielsystem Python installiert sein, nach den Angaben der Systemersteller mindestens in der Version 2.4. Außerdem wird für die ULDBs ein Datenbanksystem verwendet, welches die relationale Basistechnologie mit sich bringt. Dafür muss PostgreSQL mindestens in der Version 8.2 installiert sein. Da das Verwenden des "TrioExplorer" für die Lauffähigkeit nicht existenziell ist, müssen dafür verwendete Pakete nicht zwingend installiert sein. Dieses benötigt aber "GraphViz", ein Visualisierungstool für Graphen, mit dem die Schema-Darstellungen gerendert werden.

4.3 Konzeptionelle Grundlagen zur Speicherung von Daten

4.3.1 Data Lineage

Data Lineage ist ein massives Feature des Trio-DBMS. Auf diesem basiert vor allem das Konzept der ULDBs. diese allerdings zu verstehen, muss Data-Lineage definiert werden. Data Lineage soll eine detailliertere Beschreibung liefern, wo ein Datum herkommt und nach welchem Prozess es in die Datenbank gelangt ist. In [9] werden drei Arten von Data-Lineage unterschieden: Why-Lineage, How-Lineage und Where-Lineage. Da die How-Lineage für das Trio-System aber nicht zwingend verstanden sein muss, wird sich in den beiden Abschnitten 4.3.1 und 4.3.1 den anderen beiden Lineage-Typen gewidmet.

Zur Erklärung soll dabei ein Beispiel dienen. Im folgenden seien Ticketpreise für zwei Stadien beschrieben. In Tabelle 4.1 wird in Relation $S = Stadion < Name, Stadt, Verein >$ dabei der Name des Stadions in Relation zu seinem Standort (Stadt) und dem Verein gesetzt, der dort seine Heimspiele austrägt. In Tabelle 4.2 sind mit der Relation $S = Preis < Name, Platztyp, Block, Preis >$ dann die Ticketpreise für Plätze in einem bestimmten Block in einem Stadion in Euro aufgeführt.

	Name	**Stadt**	**Verein**
t_1	Volksparkstadion	Hamburg	HSV
t_2	Weserstadion	Bremen	SV Werder Bremen

Tabelle 4.1: Relation S

	Stadionname	**Platztyp**	**Block**	**Kosten**
t_3	Volksparkstadion	Sitzplatz	C24	30
t_4	Weserstadion	Sitzplatz	101	30
t_5	Volksparkstadion	Stehplatz	A26	25
t_6	Volksparkstadion	Stehplatz	A24	24
t_7	Weserstadion	Stehplatz	122	25
t_8	Weserstadion	Sitzplatz	102	30

Tabelle 4.2: Relation P

Darauf sollen zwei Queries ausgeführt werden, die beide das gleiche Ergebnis liefern:

1. $Q5 = \pi_{Name,Verein}(\sigma_{Name=Stadionname \wedge Platztyp=\text{“}Stehplatz\text{”}}(Stadion \times Preis))$
2. $Q6 = \rho_{Name/Stadionname}(\pi_{Stadionname,Verein}(\sigma_{Name=Stadionname \wedge Platztyp=\text{“}Stehplatz\text{”}}(Stadion \times Preis))$

Das Ergebnis beider Queries ist in Tabelle 4.3 aufgeführt. Zusätzlich sind alle Tupel in allen Tabellen mit einem Identifikator t_n versehen, der das Referenzieren im Verlauf der nächsten Abschnitte ermöglicht.

	Name	Verein
t_{10}	Volksparkstadion	HSV
t_{11}	Weserstadion	SV Werder Bremen

Tabelle 4.3: Ergebnis von $Q5$ und Q_2

Why-Lineage

Die Why-Lineage gibt an, warum die Daten, die zu einem Tupel t gehören, überhaupt in der Datenbank sind. Dabei werden alle beteiligten Tupel komplett gelistet, die irgendwelche Daten für t geliefert haben. Dies geschieht nach einem relativ einfachen Prinzip basierend auf den relationalen Operatoren (siehe Abschnitt 2.5). Dabei werden die Definitionen in [12] nur für Aggregation, Selektion, Projektion und Join (Abk.: ASJP) getroffen. Dies sind die in der Praxis am häufigsten auftretenden Operatoren, die einen Großteil aller Abfragen abdecken.

Zuerst sei die Lineage für ein Tupel t definiert, welches im Ergebnis der Anwendung eines Operators O (σ, π, $\bowtie$ oder α) auf eine oder mehrere Tabellen $(T_1, \ldots, T_i)$ enthalten

ist. Diese beinhaltet alle Tupel aus $(T_1, \ldots, T_i)$, die dazu beigetragen haben, dass das t tatsächlich im Ergebnis enthalten ist. Die Lineage $\upsilon(t)$ eines Tupels t ist wie folgt definiert:

$$E = O(T_1, \ldots, T_i)$$

$$t \in E$$

$$O^{-1}_{(T_1,\ldots,T_i)}(t) = \langle T'_1, \ldots, T'_i \rangle = \upsilon(t)$$

$$T'_i \underset{max}{\subset} T_i$$

$$O(T'_1, \ldots, T'_i) = \{t\}$$

Das bedeutet, dass die Invertierung der Anwendung des Operators O im Bezug auf das Tupel t die Lineage für t erzeugt. Diese besteht aus den maximalen Subsets von $(T_1, \ldots, T_i)$, die, wenn man auf diese wiederum den Operator O anwenden würde einzig und allein das Tupel t als einziges Element der Ergebnismenge liefern würden. Somit ist jede Tabelle T'_i die“ Lineage von t in T_i” und jedes in T'_i enthaltene Tupel “trägt zu t bei”.

Nachdem die Lineage für einzelne Operatoren definiert ist, kann jetzt die Lineage einer Query definiert werden. Eine relationale Query ist formal betrachtet eine Verkettung von relationalen Operatoren. Insofern ist durch die Transitivität der einzelnen Operationsergebnisse auch die Transitivität der Data-Lineage gegeben. Es trage beispielsweise ein Tupel t' zu einem Tupel t^* bei und ein Tupel t^* zu trage zu einem Tupel t bei. Dann ist durch die Definition der relationalen Algebra gegeben, dass t' auch zu t beiträgt. Weiterhin können komplexe Queries segmentiert werden. Jede Query ist aus einer oder mehr relationalen Subqueries aufgebaut, die alle rekursiv nach dem gleichen Schema durchlaufen werden. So wird die Data-Lineage von komplexen Queries nach den vorher genannten Definitionen aufgebaut und kann nach den gleichen Prinzipiel bestimmt werden.

Wird die Why-Lineage der Tupel aus den Ergebnissen von $Q5$ und $Q6$ betrachtet, wird dies verdeutlicht. Im Falle von t_{11} ist die Why-Lineage einfach zu bestimmen. Zu diesem Tupel haben die Tupel t_2 und t_7 beigetragen. Daher ist die Lineage $\upsilon(t_{11}) = \{Stadien(t_2), Preise(t_7)\}$. Bei der Betrachtung von t_{10} wird aber die Verwendung des maximalen Subsets deutlich. Wird dessen Why-Lineage betrachtet, so muss zuerst die Why-Lineage in der Operation $Q' = \sigma_{Name=Stadionname \wedge Platztyp=\text{“}Stehplatz''}(Stadion \times Preis)$ betrachtet werden. Das Ergebnis von Q' wird in Tabelle 4.4 dargestellt.

	Name	Stadt	Verein	Stadionname	Platzart	Block	Preis
t_{20}	Volksparkstadion	Hamburg	HSV	Volksparkstadion	Stehplatz	A26	25
t_{21}	Volksparkstadion	Hamburg	HSV	Volksparkstadion	Stehplatz	A24	24
t_{22}	Weserstadion	Bremen	SV Werder Bremen	Weserstadion	Stehplatz	122	25

Tabelle 4.4: Ergebnis von Q'

Das Ergebnis dieser Operation hat drei Tupel. Wenn jetzt der nächste Operator, die Projektion $\pi_{Stadionname,Verein}(Q')$ angewendet wird, gleichen sich die resultierenden Ergebnisse aus t_{20} und t_{21} (nämlich (Vorlksparkstadion, HSV)) und werden somit nach der Mengenlehre zum gleichen Element. Da aber das maximale Subset aller beteiligten Tupel für die Why-Lineage von t_{10} gesucht wird, sind hier beide Varianten zu beachten, auch wenn sie das gleiche Ergebnis bringen. Damit ist die Why-Lineage $v(t_{10}) = Stadien(t_2), Preise(t_5, t_6)$.

Where-Lineage

Während die Why-Lineage die Beziehung zwischen Tupeln in Ergebnissen von Abfragen und ihren Quelltupeln darstellt, stellt Where-Lineage die Beziehung der Daten, die ein Tupel in einem Abfrageergebnis enthält zu ihrer Herkunft dar. Angenommen, ein Tupel $t = (a_1, \ldots, a_n)$ existiert im Ergebnis einer Query. Dabei beschrieben $(a_1, \ldots, a_n)$ die Werte der Attribute des Tupels. Die Where-Lineage beschreibt konkret die Herkunft eines spezifischen Wertes a_i in t. Dazu werden die einzelnen Daten in Relationen und Abfrageergebnissen (oder Zellen in einer Tabelle) als "Locations" beschrieben. Jede Location l_n in t besitzt damit eine Quell-Location. Nach der Definition von [8] gibt es nur dann Quell-Locations, wenn ein Datum nicht direkt in einer Query erzeugt wird. Wenn ein Datum aber nicht direkt in der Query erzeugt wird, gehören genau die Locations zur Where-Lineage von l_i, aus denen der Wert, der an l_i steht, heraus gezogen wurde.

So sei als Beispiel wieder die Tabelle 4.3 betrachtet. Die Location l_1 sei $(t_{10}, Name)$. Unter Verwendung von $Q5$ ist es deutlich, aus welcher Location l' innerhalb der Why-Lineage der Wert "Volksparkstadion" an dieser Location herkommt. Dieser kann nur aus $l' = (t_1, Name)$ gekommen sein und ist damit die Where-Lineage von l_1. Das Tupel t_2 in der Why-Lineage findet hier für die Where-Lineage von l_1 keine Beachtung, denn es kann durchaus in der Why-Lineage eines Tupels Quelltupel geben, die in der Where-Lineage einer Location nicht auftauchen. Allerdings ist es immer so, dass die Locations aus der Where-Lineage in der Why-Lineage enthalten sind.

In einigen Fällen ist die Where-Lineage aber nicht eindeutig bestimmbar. Beispielhaft kann wieder l_1 betrachtet werden, diesmal aber als Ergebnis von $Q6$. Die Why-

Lineage von t_{10} beinhaltet sowohl t_5 als auch t_6. Da bei beiden für "Stadionname" der gleiche Wert enthalten ist und damit beide Locations an dieser Stelle für l_1 die Quelle liefern, ist die Where-Lineage von l_1 unter Verwendung von $Q6$ damit $\{(t_5, Stadionname), (t_6, Stadionname)\}$.

4.3.2 LDBs

Wie in Abschnitt 4.1 bereits erwähnt basiert das Trio-System auf ULDBs. Diese sind eine Erweiterung von LDBs (Lineage-Databases) um Unsicherheit. Eine LDB ist eine Datenbank, die Lineage beinhaltet. Dazu wird nach [7] eine LDB D aus drei Teilen modelliert. Der erste Teil dieser Datenbank sind die Daten. Da die LDBs das relational Modell erweitern, werden die Daten in LDBs in Relationen $R_1, \ldots, R_i$ gespeichert, wobei die Gesamtmenge aller Relationen mit $\overline{R}$ bezeichnet wird. Der zweite Teil ist eine Identitätsfunktion $I(\overline{R})$. Jedes Tupel $t \in \overline{R}$ besitzt einen eindeutigen Identifikator. Die Menge aller Symbole, die in $I(\overline{R})$ bezeichnen wir als S. Den dritten Teil liefert eine Lineage-Funktion λ. Lineage wird hier als Synonym für Where-Lineage verwendet. Diese Lineage-Funktion liefert zu jedem Tupel t eine Liste mit Identifikatoren $\{i_1, \ldots, i_n\}$ zurück. Für jedes Attribut von t wird festgestellt, welches Tupel für die Herkunft des Wertes darin verantwortlich ist. Dieses wird dann an der entsprechenden Stelle in der Liste aufgeführt.

Auch hier sei zur Veranschaulichung wieder ein Beispiel angeführt. Dieses stammt aus [7]. Die Relationen in den Tabellen 4.5 stellen eine Informationsdatenbank zu einem Kriminalfall dar. Dabei werden in der Relation HatGesehen(Zeuge,Automarke) Zeugenbeobachtungen aufgeführt auf die Frage, ob am Tatort ein Auto gesehen wurde. In der Relation Faehrt(Person, Automarke) werden die Fahrzeuge der Verdächtigen aufgetragen. Tabelle 4.5c beschreibt dann als Beschuldigt(Zeuge, Person) das Ergebnis der Query $\pi_{Zeuge,Person}(HatGesehen \bowtie Faehrt)$.

ID	Zeuge	Automarke
21	Anton	Audi
22	Anton	Opel
23	Bert	Volvo

(a) HatGesehen(Zeuge, Automarke)

ID	Person	Automarke
31	Cedric	Audi
32	Cedric	Opel
33	Doris	Audi
34	Doris	Volvo

(b) Faehrt(Person, Automarke)

ID	Zeuge	Person	
41	Anton	Cedric	$\lambda(41) = \{21, 31\}$
42	Anton	Cedric	$\lambda(42) = \{22, 32\}$
43	Anton	Doris	$\lambda(43) = \{21, 33\}$
44	Bert	Doris	$\lambda(44) = \{23, 34\}$

(c) Beschuldigt(Zeuge, Person)

Tabelle 4.5: Beipieldaten für eine LDB

S beinhaltet für das aufgeführte Beispiel alle Werte, die als ID für ein Tupel eingetragen sind, ergo ist $S = \{21, 22, 23, 31, 32, 33, 34, 41, 42, 43, 44\}$. Bei Betrachtung der Tupel 41 und 42 fällt auf, dass das Ergebnis zwar das gleiche ist, die Lineage λ der Daten sich dennoch unterscheidet. Tupel 41 wird erzeugt, in dem der Wert "Anton" aus Tupel 21 mit "Cedric" aus Tupel über die gemeinsame Automarke "Audi" gejoint wird. Tupel 42 hingegen beinhaltet die Werte "Anton" aus Tupel 22 und "Cedric" aus Tupel 32, welche über die gemeinsame Automarke "Opel" gejoint werden. Da sich die Lineage der beiden Tupel unterscheidet, werden diese als unterschiedlich im Ergebnis aufgeführt, auch wenn die Werte des Tupels an sich dieselben sind.

4.3.3 ULDBs

In Tabelle 4.5a werden die Tupel 21 und 22 getrennt aufgeführt. Wenn man diese Tupel als unsichere Daten betrachtet, können sie zu einem Tupel mit Attribut-Level-Unsicherheit vereint werden. Dies kann erreicht werden, indem man Unsicherheit in das Konzept von LDBs integriert und dadurch eine ULDB erzeugt. Bei der Erweiterung durch Unsicherheit wird eine Relation durch eine "x-Relation" ersetzt. X-Relationen werden aus "x-Tupeln" konstruiert. Diese stellen ein Multiset von Tupeln dar, welche die Alternativen für ein x-Tupel darstellen. Außerdem können x-Tupel mit einem Fragezeichen annotiert sein, dann werden sie als maybe-x-Tupel bezeichnet. Die Bedeutung des Fragezeichens ist die gleiche, wie in 3.1 definiert.

Damit repräsentieren x-Relationen nicht nur eine Relation, sondern alle möglichen Instanzen (siehe Abschnitt 3.2) dieser Relation. Wenn mit ähnlichen Daten wie denen in Tabellen 4.5 eine ULDB konstruiert wird, könnte diese aussehen wie in Tabellen 4.6.

ID	Hat_Gesehen′(Person, Automarke)	
21	(Anton, Audi) || (Anton, Opel)	?
23	(Bert, Volvo)	

(a) HatGesehen′(Zeuge, Automarke)

ID	Faehrt′(Person, Automarke)
31	(Cedric, Audi)
32	(Cedric, Opel)
33	(Doris, Audi)
34	(Doris, Volvo)

(b) Faehrt′(Person, Automarke)

ID	Beschuldigt′(Zeuge, Person)		
41	(Anton, Cedric)	?	$\lambda(41) = \{(21,1),(31,1)\}$
42	(Anton, Cedric)	?	$\lambda(42) = \{(21,2),(32,1)\}$
43	(Anton, Doris)	?	$\lambda(43) = \{(23,1),(33,1)\}$
44	(Bert, Doris)		$\lambda(44) = \{(23,1),(34,1)\}$

(c) Beschuldigt′(Zeuge, Person)

Tabelle 4.6: Beipieldaten für eine ULDB

Das grundlegende Konzept der Datenbank ändert sich im Vergleich zur LDB nicht. Durch das Hinzufügen der Unsicherheit ändern sich jedoch die Werte der Lineage-Funktion. Bei Tupel 41 in Tabelle 4.6c wird als Lineage {21,31} und nur die Tupel-ID des für das Wert verantwortlichen Tupels angegeben. In Tabelle 4.6c werden hingegen Tupel aus einer ID und einer Zahl als verantwortlich für einen Wert angegeben. Der zweite Wert der Tupel innerhalb λ beschreibt die Alternative, unter Verwendung welcher dieses Tupel entsteht. So ist für "Person" in Tupel 41 die erste Alternative von Tupel 21 für den Wert "Anton" verantwortlich. Für den gleichen Wert in Tupel 42 ist hingegen die zweite Alternative von Tupel 21 verantwortlich. Außerdem ist Tupel 21 mit einem Fragezeichen annotiert und damit ein Maybe-x-Tupel. Wenn bei einem Quelltupel die Möglichkeit besteht, dass es in einer Instanz nicht vorhanden ist, kann es bei einem Join passieren, dass dieses Tupel auch in einigen Instanzen des Ergebnisses nicht enthalten ist. Daher

werden x-Tupel, die ein Maybe-x-Tupel als Quelle eines Wertes verwenden, ebenso zu Maybe-x-Tupeln.

Weiterhin gibt es in ULDBs die so genannte "Confidence" von Daten, welche durch einen Wert $0 \leq c(\alpha) \leq 1$ belegt ist. Diese gibt für jede Alternative α eines Tupels t eine Wahrscheinlichkeit an, dass dieses in der "korrekten" Instanz einer Relation enthalten ist. Die Berechnung und Verteilung dieser Confidences gleicht derer aus der Theorie der probabilistischen Datenbanken, die in Abschnitt 5.3.3 beschrieben werden.

4.4 Aufbau des Systems

Trio ist eine eigene Anwendung, die keine eigene Speicherungstechnologien mitbringt. Um Daten zu speichern, setzt sich das System auf ein relationales Datenbanksystem auf. Da aber in relationalen Datenbanken ULDBs nicht nativ gespeichert werden können, muss ein Weg gefunden werden, wie die ULDBs so strukturiert werden können, dass sie in relationalen Datenbanken repräsentiert werden können. Um dies zu erreichen, werden ULDBs in mehrere relationale Tabellen aufgeteilt. Dabei werden aus einer ULDB T zwei Tabellen T_C und T_U erzeugt. In Tabellen 4.7 wird demonstriert, wie die Ergebnisse von Tabelle 4.6c im relationalen Datenbanksystem gespeichert werden.

xid	aid	conf
1	1	
2	2	
3	3	
4	4	

(a) Tabelle mit unsicheren Attributen T_U

Zeuge	Person	xid	alts	num
Anton	Cedric	1	1	2
Anton	Cedric	2	1	2
Anton	Doris	3	1	2
Bert	Doris	4	1	1

(b) Tabelle mit sicheren Attributen T_C

aid	srclin	srctable
1	1	Hat_Gesehen
1	1	Faehrt
2	2	Hat_Gesehen
2	2	Faehrt
3	1	Hat_Gesehen
3	3	Faehrt
4	3	Hat_Gesehen
4	4	Faehrt

(c) Lineage-Tabelle T_L

Tabelle 4.7: Beispiel für das Encoding von unsicheren Daten in Trio

In der Tabelle T_C (Tabelle 4.7b)wird für jedes x-Tupel aus der ULDB ein Eintrag erzeugt. Dieser wird mit einer Eindeutigen ID, die mit "xid" bezeichnet wird, versehen, um zu kennzeichnen, welches Tupel gemeint ist. Außerdem wird in einer Spalte "num" mit einer Zahl denotiert, ob ein Tupel ein Maybe-Tupel ist, oder nicht. Schlussendlich werden alle sicheren Attribute dieses Tupels mit in dieser Tabelle gespeichert.

In der Tabelle T_U (Tabelle 4.7a) werden die Alternativen für ein Tupel gespeichert. Für jede Alternative wird ein Tupel in T_U erzeugt, welches alle unsicheren Attribute und ihre Werte in einzelnen Spalten beinhaltet. Auch in dieser Tabelle gibt es für jedes Tupel eine eindeutige ID, hier als "aid" denotiert. In einer weiteren Spalte wird die xid des Tupels, zu dem diese Alternative gehört gespeichert. Die Confidence einer Alternative wird auch in dieser Tabelle als Spalte gespeichert, wenn eine existiert. Somit kann eine ULDB auch als Join über die xid der Tabellen T_C und T_U bezeichnet werden.

Eine dritte Tabelle T_L (Tabelle 4.7c) beinhaltet die Lineage der ULDB. Die Lineage einer Alternative beinhaltet, wie in Abschnitt 4.3.3 beschrieben, die Alternative des Quelltupels, aus der die Daten kommen. Daher ist eine Lineage-Tabelle T_L aufgeschlüsselt nach der aid der Alternative, für die die Lineage gespeichert wird und der "srclin", also

der aid der Quellalternative. Zusätzlich wird noch der Name der Quelltabelle für diese Alternative gespeichert. Die letzte Spalte beinhaltet ein Flag, um zu kennzeichnen, wenn zwei Lineage-Einträge für ein Tupel nicht durch Konjunktion verknüpft sind, wie es der Standard ist, sondern durch Negation oder Disjunktion.

Um diese Repräsentation zu erzeugen, ist Trio nicht direkt in PostgreSQL eingepasst, sondern setzt nur darauf auf. In Abbildung 4.1 wird das Zusammenspiel der Komponenten von Trio dargestellt.

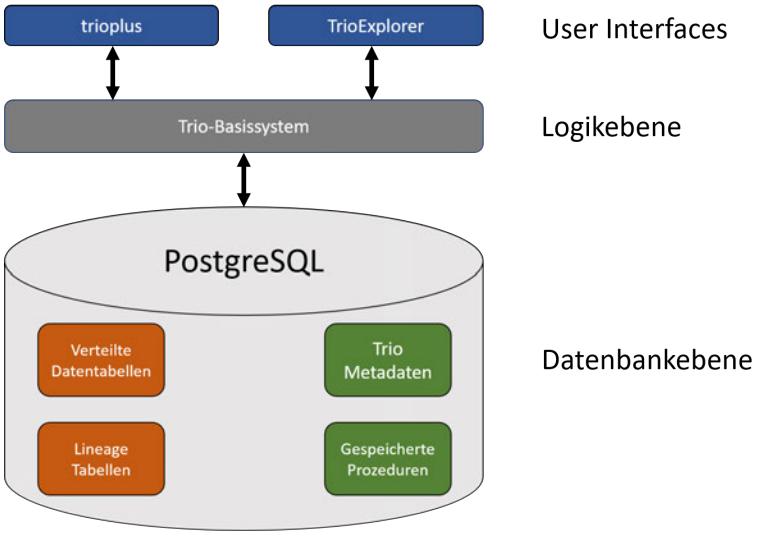

Abbildung 4.1: Systemaufbau des Trio-Systems

Zugrunde liegt dabei das relationale Datenbanksystem in Form von PostgreSQL. Darin werden bei der Installation von Trio Metadaten initialisiert und gespeicherte Prozeduren abgelegt, die beispielsweise für die Berechnung und Verwendung der Confidence-Funktionen benutzt werden. Außerdem enthält das Datenbanksystem die ULDBs, die als relationale Tabellen gespeichert werden und die zugehörigen Lineage-Tabellen.

Darauf aufgesetzt ist eine Python-API, die die Übersetzung zwischen den Statements in TriQL und regulären SQL Statements, sowie auch die Zusammensetzung der Repräsentationen der Ergebnisse übernimmt. Diese verwendet zur Kommunikation mit dem relationalen Datenbanksystem die Python DB 2.0 API.

Um Benutzereingaben zu ermöglichen, wurden die beiden Anwendungen trioplus und TrioExplorer entwickelt. trioplus ist ein Kommandozeilenprogramm, welches dem Program psql, welches die Standard Benutzeranwendung für PostgreSQL ähneln soll. Dieses bietet vollumfängliche Unterstützung von TriQL und Darstellung von Abfragen im Terminal. TrioExplorer ist eine grafische Oberfläche. TrioExplorer baut einen multi-threaded

Webserver via CherryPy, einer Python-Webserverimplementation, auf und bietet eine deutlich umfangreichere Interaktionsmöglichkeit mit Trio. Dabei wird der Mehrbenutzerbetrieb unterstützt, sowie eine grafische Darstellung von Schemata, Abfrageergebnissen, Lineage und Confidence-Berechnungen.

4.5 TriQL

Um in Trio ULDBs mit Daten zu versorgen und diese korrekt zu behandeln und zu verarbeiten, ist für das Trio-System eine eigene Query-Language namens TriQL (gespr. Tri-Quel) entstanden. Da das System mit PostgreSQL auf einem relationalen Datenbanksystem aufsetzt, bildet SQL die Grundlage für die TriQL. Insofern sind die Queries, die zu dieser Sprache gehören, sehr ähnlich dazu aufgebaut. Dennoch gibt es Erweiterungen und Besonderheiten, die die Arbeit mit Alternativen und unsicheren Daten ermöglichen.

Generell werden in TriQL alle Operationen, die Alternativen beinhalten, mit eckigen Klammern ausgedrückt. Trio ermöglicht die direkte Überprüfung von einzelnen Attributen innerhalb der Alternativen. Außerdem lässt sich durch integrierte Funktionen wie "conf()" zur Berechnung der Confidence einer Alternative oder "Lineage()" zum Ausgeben der Lineage mehr Information ausgeben, als auf den ersten Blick in einer Tabelle enthalten. Mit "Flatten" und "GroupAlts" werden zusätzlich Befehle bereitgestellt, mit denen sich eine relationale Tabelle in eine ULDB konvertieren lässt und umgekehrt.

5 MayBMS

Nachdem das erste Framework vorgestellt wurde, folgt nun mit MayBMS das zweite in dieser Arbeit zu evaluierende Framework.

5.1 Überblick

Das zweite Framework welches diese Thesis evaluieren soll ist MayBMS (gesprochen: "Maybe-MS"). Dieses ist ein Open-Source-DBMS welches als Projekt mehrerer Universitäten entstanden ist. Hier haben Studenten der Cornell University und der Oxford University unter der Leitung von Professor Christoph Koch der Oxford University und Dan Olteanu zusammen gearbeitet. Das System baut auf der Anforderung auf, effizient möglichst viele mögliche Welten einer Datenbank gleichzeitig repräsentieren zu können [3]. Die Forschung für dieses Projekt beinhaltet unter anderem auch die eine Query Language analog zu SQL für unsichere Daten, sowie die Bereitstellung einer API für diese. [23].

5.2 Umfang und Systemvoraussetzungen

Der erste Prototyp des Systems wurde auf der ICDE im Jahre 2007 vorgestellt [4]. Dieser basierte noch auf World-Set-Dekompositionen. Der letzte Release des Systems von 2009 in der Version 2.1, basiert allerdings auf U-Relationen-Datenbanken. Dieser wird in einem kompakten Installer für Windows- und Linux-Betriebssysteme bereitgestellt. Darin enthalten ist auch das PostgreSQL Backend in der Version 8.3.3, auf welchem das System aufsetzt. Dadurch, dass MayBMS direkt in das Server-Backend von PostgreSQL integriert wurde, werden SQL Operationen großenteils unterstützt. Da MayBMS keine grafische Oberfläche mitbringt, sind auch keine weiteren Systemvoraussetzungen nötig [24].

5.3 Konzeptionelle Grundlagen zur Speicherung von Daten

MayBMS verwendet als Speicherungsform für enthaltene Daten probabilistische u-Tabellen. Dies sind Tabellen, die mit Wahrscheinlichkeiten versehen sind und eine akkurate Repräsentation der kompletten enthaltenen Informationen innerhalb von standardisierten relationalen Tabellenzulassen. Eine genaue Definition findet in Abschnitt 5.3.4 statt. U-Tabellen sind eine Erweiterung von c-Tabellen, die wiederum auf v-Tabellen aufbauen.

5.3.1 Naive Tabellen

Die erste Erweiterung der Codd-Tabellen stellen nach [16] die naiven Tabellen, auch v-Tabellen genannt dar. Während in Codd-Tabellen Null-Werte mit @ denominiert und mit einem distinkten Variablennamen substituiert werden können, können in v-Tabellen Variablen mehrfach auftreten. Variablen, die mit dem gleichen Symbol beschrieben werden, beinhalten dann auch immer den gleichen Wert.

Ein Beispiel für v-Tabellen und Abfragen darauf ist in Tabelle 5.1 aufgezeigt. Durch diese eindeutige Unterscheidung können auch Join-Operationen durchgeführt werden. Dies kann gezeigt werden, indem die Variablen als Konstanten behandelt werden und die bisher geführten Beweise unter dieser Annahme für v-Tabellen durchgeführt werden. Damit sind v-Tabellen ein schwaches Repräsentationssystem für die positive relationale Algebra.

A	**B**	**C**
1	1	x
x	z	1

(a) Relation R_1 als v-Tabelle

C	**D**
1	1
x	z

(b) Relation R_2 als v-Tabelle

A	**B**	**C**
1	1	x
x	z	1

(c) Ergebnis für $\overline{\pi_B(R_1)}$

A	**B**	**C**	**D**
1	1	x	z
x	z	1	1

(d) Ergebnis für $\overline{R_1 \bowtie R_2}$

Tabelle 5.1: Beispiele für naive Tabellen (v-Tabellen)

5.3.2 Konditionale Tabellen

Um auf Basis der bisher verwendeten Technologien ein starkes Repräsentationssystem für die relationale Algebra zu erhalten, müssen in v-Tabellen gleiche Tupel deutlich voneinander differenzierbar sein. Dazu wurden konditionale Tabellen, auch c-Tabellen genannt, eingeführt.

C-Tabellen beschreiben ein Tripel (T, Φ_T, ϕ). Dabei ist T eine v-Tabelle. Um aber Abhängigkeiten von Belegungen darzustellen, werden mit Φ_T und ϕ Konditionen beschrieben, die erfüllt sein müssen, damit ein Tupel in einer Instanz enthalten ist. ϕ ist eine Funktion, die jedem Tupel eine Kondition zuordnet. Φ_T steht dabei für die globale Kondition einer Tabelle. Diese gilt zusätzlich für alle Tupel. Eine Kondition setzt sich immer wie folgt zusammen:

1. Den Wahrheitswerten "wahr" und "falsch", welche auch durch $x = x$ und $x \neq x$ dargestellt werden können oder im Falle von "wahr" einfach weggelassen werden können
2. $x = y$ und $x \neq y$, wenn x und y Variablen sind, die Null-Werte beinhalten
3. $x = c$ und $x \neq c$, wenn x eine Variable ist, die einen Null-Wert beinhaltet und c eine Konstante ist
4. Beliebigen Verkettungen der vorigen 3 Optionen mit den Operatoren $\wedge$, $\vee$ und $\neg$

Dabei ist es auch möglich, dass in den Konditionen Variablen auftauchen, die in dem Tupel oder der Tabelle, für die diese Kondition gilt, gar nicht existieren. Nur, wenn Die Kondition eines Tupels zum Wahrheitswert "wahr" evaluiert werden kann, ist das Tupel in der Tabelle enthalten. Damit können auch Tupel, die gleichen Werte beinhalten, unterschieden werden, wenn sie unter unterschiedlichen Konditionen in verschiedenen Instanzen enthalten sind.

Dies wird am Beispiel der Tabellen 5.2 dargestellt. Wenn die Domänen der Attribute A und B auf $\{0, 1, 2, 3\}$ festgesetzt werden, dann sind die Tabellen J_1 (Tabelle 5.2b), J_2 (Tabelle 5.2c) und J_3 (Tabelle 5.2d) mögliche Welten von J(Tabelle 5.2a), allerdings mit unterschiedlichen Belegungen $\nu(x, y, z)$. Da in der globalen Kondition Φ_J festgehalten ist, dass $x \neq 2 \wedge y \neq 2$ gilt, wird es keine Welt von J geben, in denen eine der beiden Variablen mit 2 belegt sein wird. Die Bedingung $z = z$ evaluiert immer zu "wahr", daher ist dieses Tupel auch in allen Welten von J enthalten. Für die Belegung, aus der J_1 entsteht, evaluieren auch alle Konditionen der einzelnen Tupel zu "wahr", daher sind auch alle Tupel enthalten. Für die Belegung, aus der J_2 entsteht, ist die Bedingung $y = 0$

nicht erfüllt, daher fält das zweite Tupel aus der Welt raus. Bei der dritten Belegung, die für die Entstehung von J_3 sorgt, wird weder $y = 0$ noch $x \neq y$ erfüllt, weshalb in dieser Welt nur das erste Tupel aus J auftaucht.

A	**B**	$\Phi_T = (x \neq 2 \wedge y \neq 2)$
0	1	$z = z$
1	x	$y = 0$
y	x	$x \neq y$

(a) Beispiel für c-Tabelle J

A	**B**
0	1
1	3
y	3

(b) Welt J_1 für $\nu = (3, 0, 0)$

A	**B**
0	1
1	x

(c) Welt J_2 für $\nu = (0, 1, 0)$

A	**B**
0	1

(d) Welt J_3 für $\nu = (1, 1, 0)$

Tabelle 5.2: Beispiele für konditionale Tabellen (c-Tabellen)

Das Ausführen von Operationen der relationalen Algebra auf eine Tabelle ergänzt im Ergebnis die Konditionen, die für ein Tupel dieser Tabelle gelten. Als Grundlage für die folgenden Beispiele, wie dies im konkreten aussieht, wird in Tabelle 5.3 eine mögliche c-Tabelle T aufgeführt.

A	**B**	**C**	
1	2	z	$z \neq 3$
1	y	3	$y \neq 2$
x	2	3	$x \neq 1$

Tabelle 5.3: Beispiel für c-Tabelle T

Bei einem Join werden die Bedingungen beider enthaltener Tupel mit einem $\wedge$-Operator verknüpft. Zusätzlich wird eine Bedingung hinzugefügt, die dafür sorgt, dass der gemeinsame Wert, über den die beiden Tupel verknüpft werden, auch gleich sind. Dies ist in Tabelle 5.4b als Tabelle U für das Ergebnis der Abfrage $U = \pi_{AB}(T) \bowtie \pi_{BC}(T)$ dargestellt. Dabei treten auch gelegentlich Widersprüche in den Konditionen auf, wodurch bei Evaluation Tupel entfernt werden. Im zweiten Tupel von 5.4a beispielsweise enthält die Bedingung $(y \neq 2) \wedge (2 = y)$. Dies evaluiert immer zu "falsch" weshalb dieses Tupel kein Teil des korrekten Ergebnisses für U ist.

A	B	C	
1	2	z	$(z \neq 3) \wedge (z \neq 3) \wedge (2 = 2)$
1	2	3	$(z \neq 3) \wedge (y \neq 2) \wedge (2 = y)$
1	2	3	$(z \neq 3) \wedge (x \neq 1) \wedge (2 = 2)$
1	y	z	$(y \neq 2) \wedge (z \neq 3) \wedge (y = 2)$
1	y	3	$(y \neq 2) \wedge (y \neq 2) \wedge (y = y)$
1	y	3	$(y \neq 2) \wedge (x \neq 1) \wedge (y = 2)$
x	2	z	$(x \neq 1) \wedge (z \neq 3) \wedge (2 = 2)$
x	2	3	$(x \neq 1) \wedge (y \neq 2) \wedge (2 = y)$
x	2	3	$(x \neq 1) \wedge (x \neq 1) \wedge (2 = 2)$

(a) Komplettes Ergebnis für U inklusive Widersprüchen

A	B	C	
1	2	z	$(z \neq 3)$
1	2	3	$(z \neq 3) \wedge (y \neq 2)$
1	y	3	$(y \neq 2)$
x	2	z	$(x \neq 1) \wedge (z \neq 3)$
x	2	3	$(x \neq 1)$

(b) Komplett evaluiertes Ergebnis mit vereinfachten Konditionen für U

Tabelle 5.4: Ergebnis für U

Bei einer Projektion werden die Bedingungen des Quelltupels an das Ergebnistupel vererbt. Wenn zwei Tupel u und v sich in ihren Werten gleichen, in der Kondition aber unterscheiden, muss diese näher geprüft werden. Als Beispiel dafür wird in Tabelle 5.5 die Tabelle $W = \pi_{AC}(U)$ generiert. Das Ergebnis der Abfrage für das zweite und dritte Tupel aus U ist in beiden Fällen $(1, 3)$. Da bei beiden aber in der Bedingung $(y \neq 2)$ steht, wird das zweite Tupel obsolet, weil, für denn Fall, dass dies gegeben ist, und unabhängig davon, ob $(z = 3)$ wahr ist, das Tupel $(1, 3)$ eh im Ergebnis enthalten ist.

A	C	
1	z	$(z \neq 3)$
1	3	$(z \neq 3)$
x	z	$(z \neq 3) \wedge (z \neq 3)$
x	3	$(z \neq 3)$

Tabelle 5.5: Ergebnis für W

Bei einer Selektion wird als zusätzliche Bedingung die Bedingung der Selektion ergänzt. Dabei wird der zu filternde Wert als Gleichheitsbedingung zu den lokalen Bedingungen eines Tupels hinzugefügt. Sei in Tabelle 5.6 das Ergebnis der Abfrage $L = \sigma_{C=3}(W)$ aufgeführt. Da der Wert, der für jedes Tupel beim Attribut C eingetragen ist, 3 betragen soll, wird jedem Tupel die Kondition hinzugefügt, dass der Wert, der an dieser

Stelle steht 3 sein soll. Da $(z \neq 3) \wedge (z = 3)$ aber immer zu falsch evaluiert wird, sind die Tupel $(1, z)$ und (x, z) nicht mehr im Ergebnis enthalten.

A	**C**	
1	z	$(z \neq 3) \wedge (z = 3)$
1	3	$(y \neq 2) \wedge (3 = 3)$
x	z	$(x \neq a) \wedge (z \neq 3) \wedge (z \neq 3)$
x	3	$(x \neq 1) \wedge (3 = 3)$

(a) Komplettes Ergebnis für L inklusive Widersprüchen

A	**C**	
1	3	$(y \neq 2)$
x	3	$(x \neq 1)$

(b) Komplett evaluiertes Ergebnis mit vereinfachten Konditionen für L

Tabelle 5.6: Ergebnis für L

Durch diese Eindeutigkeit der Existenz von Tupeln in einer c-Tabelle bilden c-Tabellen ein starkes Repräsentationssystem für die relationale Algebra.

5.3.3 Probabilistische c-Tabellen

Nachdem mit c-Tabellen ein starkes Repräsentationssystem für die relationale Algebra beschrieben ist, können darauf Weiterentwicklungen geschehen, die das Konzept der Wahrscheinlichkeit mit integrieren. Die Basis dazu bieten probabilistische c-Tabellen nach [34]. Dafür wird im Folgenden in jedem Fall die CWA verwendet. Wenn mehrere fest definierte Alternativen dargestellt werden sollen, muss der Raum, den ein Wert annehmen kann, begrenzt werden. Zusätzlich werden Alternativen für einen Wert nicht mehr durch Variablen beschrieben. Diese dienen hierbei nur noch zur Zustandsbeschreibung.

Als Beispiel sei ein handschriftlich auszufüllendes Formular verwendet, in dem eine beliebige dreistellige Zahl und der Name der ausfüllenden Person einzutragen sind. Wenn eine Zahl bereits verwendet wurde, wird sie veröffentlicht und darf nicht erneut verwendet werden. Zwei Personen haben dieses Formular ausgefüllt, haben aber beide eine sehr undeutliche Handschrift, weshalb die Zahl nicht ganz eindeutig ist. Das Ergebnis sieht als unvollständige Tabelle F wie in Tabelle 5.7 aus.

Form_ID	**Number**	**Person**
1	(563, 568)	Mustermann
2	(563, 553)	Zeigemann

Tabelle 5.7: Beispieldaten für das beschriebene Formular als Tabelle F

Wenn dieses Beispiel in eine c-Tabelle F' konvertiert wird, wird zu jeder möglichen Werteverteilung eine Tupelvariante erstellt. Zusätzlich werden Bedingungen für Variablen hinzugefügt, die beschreiben, dass die Varianten eines Wertes zu keinem Zeitpunkt gleichzeitig in einer möglichen Welt $F'_i \in REP(F')$ existieren können. Die Wertemenge der zugehörigen Variable beinhaltet die Zahlen $1, \ldots, n$, wobei n die Anzahl der möglichen Alternativen eines Wertes ist. Die Wertemenge wird auch als Dom_x bezeichnet, oder Domäne von x. Da hier bei beiden Personen je zwei Zahlen in die Handschrift zu interpretieren sind, ist die Wertemenge der Variablen x, y, die je ein unsicheres Attribut darstellen $\{1, 2\}$.

Außerdem müssen Bedingungen hinzugefügt werden, die verhindern, dass die beiden Personen die gleiche Zahl gewählt haben. Damit sehen eine mögliche Repräsentation von F' und die möglichen Welten wie in Tabelle 5.8 aus.

ID	**Nr**	**Prs**	
1	563	Mustermann	$x = 1$
1	568	Mustermann	$x \neq 1$
2	563	Zeigemann	$y = 1 \wedge x \neq 1$
2	553	Zeigemann	$y \neq 1 \vee x = 1$

(a) Basistabelle F'

ID	**Nr**	**Prs**
1	563	Mustermann
2	553	Zeigemann

(b) F'_1
$(x \mapsto 1, y \mapsto 1)$
$(x \mapsto 1, y \mapsto 2)$

ID	**Nr**	**Prs**
1	568	Mustermann
2	563	Zeigemann

(c) F'_2
$x \mapsto 2, y \mapsto 1$
-

ID	**Nr**	**Prs**
1	568	Mustermann
2	553	Zeigemann

(d) F'_2
$x \mapsto 2, y \mapsto 2$
-

Tabelle 5.8: Beispieldaten für das beschriebene Formular als c-Tabelle

Aus dieser c-Tabelle kann jetzt eine probabilistische c-Tabelle, auch pc-Tabelle genannt, erstellt werden. Da die Variablen eine feste Wertemenge besitzen, können die Bedingungen normiert werden, so dass sie alle das gleiche Format haben. Alle Bedingungen werden aus Konjunktionen von atomaren Formeln nach dem Schema $x = k$ zusammengesetzt, wobei x eine Variable und k eine Konstante ist.

Wenn Bedingungen mit einer Disjunktion verknüpft werden, bedeutet dies, dass sich das betroffene Tupel innerhalb der Tabelle theoretisch wiederholt. Dabei existiert

das Tupel jeweils einmal für beide Bedingungen in der Disjunktion in der Tabelle. Das kann in Tabelle 5.9a beobachtet werden.

Außerdem wird eine Wahrscheinlichkeitsverteilung gebildet. Das bedeutet, dass jeder möglichen Belegung a für eine Variable x eine Wahrscheinlichkeit $P(x = a) = p \in [0, 1]$ zugewiesen wird. Alle Varianten für Belegungen einer Variable Dom_x müssen in Summe 1 ergeben. Diese Wahrscheinlichkeitsverteilung kann wiederum als Tabelle dargestellt werden. Dabei wird eine Spalte für den Variablennamen (N), eine für die Variablenbelegung (B) und eine für die Wahrscheinlichkeit, mit der diese Belegung auftritt (P). Als Beispiel für eine Wahrscheinlichkeitsverteilung der Tabelle F' sei Tabelle 5.9b betrachtet.

ID	**Nr**	**Prs**	
1	563	Mustermann	$x = 1$
1	568	Mustermann	$x = 2$
2	563	Zeigemann	$y = 1 \wedge x = 2$
2	553	Zeigemann	$y = 2 \vee x = 1$

(a) F' als pc-Tabelle

N	**B**	**P**
x	1	0.2
x	2	0.8
y	1	0.4
y	2	0.6

(b) Wahrscheinlichkeitsverteilung für die Belegungen aus F'

Tabelle 5.9: Beispiel einer pc-Tabelle

Daraus lässt sich dann die Wahrscheinlichkeit berechnen, mit der ein Tupel entsteht, indem die Wahrscheinlichkeit berechnet wird, mit der die Welt korrekt ist, in der das Tupel repräsentiert wird. Um dies zu erreichen, wird für die Belegung der Variablen ν_i, mit der die Welt entsteht, die Wahrscheinlichkeit für jede einzelne Variablenbelegung multipliziert wird. Kann eine Welt durch mehrere mögliche Belegungen entstehen, werden die einzelnen Wahrscheinlichkeiten der Belegungen addiert. Um die entstehende Welt F'_1 aus Tabelle 5.8a zu bekommen, sind zwei Belegungen möglich. Daher gilt unter der Wahrscheinlichkeitsverteilung aus Tabelle 5.9b:

$$p(F'_1) = (0,2 \cdot 0,4) + (0,2 \cdot 0,6) = 0,24$$

5.3.4 U-Tabellen

Die normierten Bedingungen aus pc-Tabellen resultieren in einer neuen Darstellbarkeitsmöglichkeit für Variablenbelegungen. Bisher wurden durch die Verwendung der CWA und die Entfernung von Variablen aus den Werten innerhalb der Tabelle werden folgende Rahmenbedingungen für Tabellen geschaffen:

1. Es gibt k Variablen $x_1, \ldots, x_k$

2. Jede Variable x_i besitzt eine endliche Wertemenge mit $d_i = \{1, \ldots, n\}$

Diese Teilnahmebedingungen erlauben es, dass die Variablenbelegungen direkt in das Tabellenschema integriert werden können. Eine solche Tabelle wird u-Tabelle genannt. Eine u-Tabelle stellt die Bedingungen, die in einer c-Tabelle beziehungsweise pc-Tabelle für ein Tupel gelten, als Attribute dar. Dazu werden für Bedingungen, die aus der Konjunktion von beliebig vielen atomaren Vergleichen nach dem Schema $x = a$, aufgebaut sind, zusätzliche Attribute in der u-Tabelle angelegt. Die Anzahl der zusätzlichen Attribute orientiert sich dabei an der Anzahl der unterschiedlichen Variablen innerhalb der Bedingungen. Wenn innerhalb der Bedingungen der kompletten Repräsentation k verschiedene Variablen vergeben werden, werden $k \cdot 2$ Attribute in der u-Tabelle benötigt, um alle Bedingungen repräsentieren zu können.Das Attribut V_1 beinhaltet dabei die Bezeichnung der Variable, die in der Bedingung auftaucht. Das Attribut D_1 wiederum beinhaltet den Wert, der dieser Variable innerhalb eines atomaren Vergleichs gleich gesetzt wird. Wenn ein atomarer Vergleich nach dem Schema $x = x$ in den Bedingungen existiert, kann dieser offensichtlich direkt zu "wahr" evaluiert werden. Wenn das der Fall ist, können die entsprechenden Attribute für diese Variablen in der u-Tabelle leer gelassen werden. Zusätzlich kann eine u-Tabelle partitioniert werden.

Die Repräsentation der pc-Tabelle aus Tabelle 5.9a als u-Tabelle F^U kann in Tabelle 5.10 betrachtet werden. Zusätzlich werden weiterhin Belegungen für Variablen mit Wahrscheinlichkeiten kombiniert. Dies geschieht nach dem gleichen Prinzip der Wahrscheinlichkeitsverteilung aus Tabelle 5.9b.

Eine u-Tabelle kann zu guter Letzt vertikal partitioniert werden. Das erlaubt das individuelle Beschreiben einzelner Attribute durch eigene Bedingungen. Dadurch können Attribute, die nicht unsicher sind, unabhängig von unsicheren Attributen betrachtet werden. Wenn die einzelnen Partitionen dann über einen oder mehrere Join-Operationen über das gleiche eindeutige Attribut zusammengeführt werden, entsteht wiederum die ursprünglich u-Tabelle. Dies kann an Tabellen 5.11a und 5.11b betrachtet werden. Die Namen der Personen sind keine unsicheren Attribute, weshalb deren Bedingung immer zu "wahr" evaluiert. Daher gibt es für dieses Attribut in Tabelle 5.11b keine Variablenattribute. Bei der Zahl, die im Formular eingetragen werden soll, gibt es allerdings mehrere Alternativen, daher entstehen in Tabelle 5.11a Namens- und Belegungsattribute für zwei Variablen. Tabelle 5.10 repräsentiert damit als Relation $F^U = F^U_{Nr} \bowtie F^U_{Prs}$.

V_1	D_1	V_2	D_2	**ID**	**Nr**	**Prs**
x	1	-	-	1	563	Mustermann
x	2	-	-	1	568	Mustermann
y	2	x	2	2	563	Zeigemann
y	2	x	1	2	553	Zeigemann
y	2	x	2	2	553	Zeigemann

Tabelle 5.10: F' als u-Tabelle F^U

V_1	D_1	V_2	D_2	**ID**	**Nr**
x	1	-	-	1	563
x	2	-	-	1	568
y	2	x	2	2	563
y	2	x	1	2	553
y	2	x	2	2	553

(a) Tabellenpartition F^U_{Nr} für die eingetragene Zahl im Formular

ID	**Prs**
1	Mustermann
2	Zeigemann

(b) Tabellenpartition F^U_{Prs} für den Namen der ausfüllenden Person des Formulars

Tabelle 5.11: Tabellenpartitionen von F^U

5.4 Umsetzung der theoretischen Konzepte im Framework

5.4.1 Erzeugung von u-Tabellen

MayBMS verwendet als Grundlage für u-Tabellen relationale Datenbanken. Daher wird die Datenbasis für die vom Framework erzeugten probabilistischen Datenbanken auch in relationalen Tabellen abgebildet. Ein Tupel t in einer Relation R besteht aus einer Menge von Attributen $A = \{a_1, \ldots, a_k\}$, die im Schema der Tabelle beschrieben sind. Wenn ein Tupel jetzt eine Mege von unsicheren Attributen $A' \subset A$ besitzt, dann lässt sich ein Menge $\overrightarrow{A} \subset A$ feststellen, die im Tupel sicher enthalten sind. Diese Menge, im folgenden "Key" genannt, kann auch eine leere Menge sein. Damit muss $schema(\overrightarrow{A} + A') \hat{=} schema(S)$ gelten.

MayBMS bietet in seiner Query-Language mit der Operation repair $\text{key}_{\overrightarrow{A}@B}(R)$ die Möglichkeit, aus einer relationalen Datenbank eine u-Tabelle zu erzeugen. Dieses Statement führt eine maximale Reparatur für einen selbst gewählten Key $\overrightarrow{A}$ auf der relationalen Tabelle R aus. Eine Reparatur eines Keys erzeugt aus der relationalen Darstellung mehrerer Tupel, die sich in $\overrightarrow{A}$ gleichen, die Repräsentation aller ihrer möglichen Welten

als u-Tabelle mit entsprechenden Gewichtungen. Dazu wird für jedes Attribut, welches nicht im Key enthalten ist, eine V- und eine D-Spalte in die Tabelle eingefügt. Die Inhalte dieser Spalten sind global über die ganze Datenbank für den Spaltentyp einzigartige positive Ganzzahlen, sowohl für die Variablennamen als auch für die möglichen Belegungen. Jede Alternative bekommt damit eine Variablenbelegung zugewiesen, für die es in einer möglichen Welt enthalten ist.

Zusätzlich wird eine Wahrscheinlichkeitsverteilung generiert, die die Wahrscheinlichkeiten für die einzelnen Belegungen enthält. Wenn in der repair-key-Operation eine Spalte B mit einem numerischen Wert zugewiesen ist, der größer als 0 ist, dann werden die Werte in dieser Spalte normiert. Für den Fall, dass die Summe für B aller möglichen Alternativen 1 ergibt, ist keine weitere Normierung nötig. Falls nicht, bewirkt diese Normierung, dass für jedes Element einer Menge an Alternativen eines unsicheren Wertes $C = \{c_1, \ldots, c_i; c \in t\}$ eines Tupels t und dessen numerischen Wertes $B_1, \ldots, B_i$ eine Wahrscheinlichkeit $0 \leq p(c_h) \leq 1$ für jede Belegung des Wertes nach folgender Formel errechnet wird:

$$p(c_h) = \frac{B_h(t)}{\sum_{n=0}^{i} B_i(t)}$$

Ergeben die Werte der Spalte in Summe 1, werden diese Werte durch diese Formel einfach in die Wahrscheinlichkeitsverteilung übernommen.Für den Fall, dass keine numerische Spalte B angegeben wird, wird eine uniformierte Wahrscheinlichkeit für jede Alternative berechnet. Damit wird für jede von n Alternativen für einen Wert eine Belegungswahrscheinlichkeit von $\frac{1}{n}$ angenommen. Die bestimmte Wahrscheinlichkeitsverteilung Für die Beispieldaten aus Tabelle 5.7 stellen die Tabellen 5.12 eine Repräsentation in MayBMS unter Ausführung der Operation $S = \text{repair-key}_{fid,prs@}(R)$ dar. Diese wird in der Query-Language von MayBMS mit dem Statement aus Listing 5.1 beschrieben.

```
1 create table s as (
2         repair key fid,name in (
3                 select * from forms
4         )
5 );
```

Listing 5.1: Statement zum Erzeugen einer u-Tabelle aus einer relationalen Tabelle

ID	Nr	Prs
1	563	Mustermann
1	568	Mustermann
2	563	Zeigemann
2	553	Zeigemann

(a) Basistabelle R

ID	Nr	V_0	D_0	p_0	Prs
1	563	1	1	0.5	Mustermann
1	568	1	2	0.5	Mustermann
2	563	2	3	0.5	Zeigemann
2	553	2	4	0.5	Zeigemann

(b) u-Tabelle S

Tabelle 5.12: Repräsentation der Beispieldaten in MayBMS

5.4.2 Filterung der möglichen Welten

Die repair-key-Operation erzeugt zwar u-Tabellen für die Alternativen eines Wertes, allerdings können keine weiteren Bedingungen, die unabhängig von der Varianz von Werten sind, direkt in die u-Tabelle eingebracht werden. Die Belegung von Variablennamen und Werten durch eindeutige, zufällige Zahlen verhindert dies. Mit jeder Erzeugung einer u-Tabelle mit Hilfe des "repair key" Statements werden neue Zahlenräume für die gleichen Werte verwendet. Das Vergleichen von mehreren Variablen miteinander ist dadurch nicht möglich, weil zwei Variablen, auch, wenn sie einen gleichen Status beschreiben, nicht die gleiche Belegung haben müssen.

Es sei das Beispiel der Formulare aus Tabelle 5.7 zur Erklärung betrachtet. Um sicher zu stellen, dass nicht zwei Formulare die gleiche Zahl beinhalten, sind die Konditionen von allen anderen Eingaben abhängig. Um eine korrekte Wahrscheinlichkeitsverteilung dafür zu erwirken, muss eine Bereinigung der möglichen Welten statt finden, die die Welt eliminiert, die als Wertemenge für die Nummern der Formulare $\{563, 563\}$ enthält. Das ist bei zwei Formularen noch relativ gut darstellbar, weil eine Bereinigung nur für diesen Fall statt finden muss. Je mehr Formulareinträge aber in dem Beispiel hinzugefügt werden, desto mehr Vergleiche müssen für eine Bereinigung angestellt werden. Diese Bereinigung ist nicht generisch darstellbar, sondern immer in Abhängigkeit der Anzahl der Einträge in der Tabelle durchzuführen.

5.4.3 Besonderheiten und Erweiterungen der Query Language

In Abschnitt 5.2 wurde erwähnt, dass das Framework auf dem PostgreSQL-Server Backend aufbaut. Als grundlegende Query Language verwendet das Framework daher SQL und erweitert diese um eigene Befehle. Die Query-Language, die für MayBMS gilt, wird allerdings noch um einige Funktionen erweitert. Diese Erweiterungen wurden ursprüng-

lich für den Prototypen MayBMS1 entwickelt [5]. Mit dem Update auf u-Tabellen sind allerdings einige Operationen aus der Query-Language entfernt oder ersetzt worden.

Die wichtigste Funktion ist wohl die Berechnung der Confidence mit der Funktion "conf()". Damit wird in MayBMS die Confidence berechnet, dass im Ergebnis ein oder mehrere Ergebnisse enthalten sind. Wenn man über die SQL-Gruppierungsfunktionen noch Attribute in die Query aufnimmt, wird die Tupel-Confidence der im Ergebnis enthaltenen Tupel berechnet. Die Confidence-Funktion wird unterstützt durch Approximationsfunktionen für die Confidence "aconf()" und "tconf()", die zwar schneller sind, aber auch Genauigkeit dafür einbüßen. Zusätzlich werden auch Aggregatsfunktionen wie "esum" oder "ecount" bereitgestellt, die Summenberechungen oder Zählungen auf unsicheren Daten erledigen.

6 Vergleich der Systeme

In diesem Abschnitt werden die beiden zu evaluierenden Frameworks gegenüber gestellt. Dafür werden Hypothesen aufgestellt, die anhand von diversen Experimenten mit beiden Frameworks bewiesen oder widerlegt werden sollen.

6.1 Methodik

Die Evaluation soll durch einen direkten Vergleich der Frameworks durchgeführt werden. Ein direkter Vergleich kann nur unter gewissen Voraussetzungen durchgeführt werden. Die erste Voraussetzung ist, dass die den Frameworks zugrunde liegenden Systeme sich gleichen. Wenn die Frameworks installiert werden, muss das auf identischen Maschinen passieren, um einen Vergleich überhaupt zu ermöglichen. Dies kann natürlich dadurch erwirkt werden, dass die Frameworks auf unterschiedlichen Rechnern mit den gleichen Spezifikationen installiert werden. In dieser Arbeit wurde sich aber stattdessen dafür entschieden, die Frameworks auf virtuellen Maschinen, zu installieren, die sich in ihrer Konfiguration gleichen. Dies bildet zwar die gleiche Situation ab, ist aber weniger Ressourcenintensiv.

Zusätzlich sollten die für Frameworks verwendeten Softwarekomponenten jeweils den gleichen Versionsstand haben, um sicher zu gehen, dass nicht durch Weiterentwicklung dieser Verfälschungen der Ergebnisse auftreten. Tests werden dann durch Ausführung von Operationen auf Datenbanken mit den zu evaluierenden Frameworks ausgeführt. Dafür wird im Vorfeld eine Menge an Daten definiert. Diese wird in beide Frameworks mit dem jeweiligen Speicherungskonzept eingefügt. Somit steht in beiden Frameworks die gleiche Datenbasis zur Verfügung. Dafür kann auf Benchmarkdaten zurückgegriffen werden, die Alternativen für Tupeln enthalten, um eine große unsichere Datenbank darzustellen. Als Beispiele dafür seien der TPC-H Benchmark [35] oder auch die Titeldatenbank der Internet Movie Database genannt [15]. In dieser Arbeit wurde sich aber dagegen entschieden, diese Möglichkeiten zu verwenden. Bei diesen Daten ist die Komlexität relativ hoch und die Parametrisierrungsmöglichkeiten im Vergleich dazu relativ gering. Das erschwert effiziente Performancebeobachtungen bei den Frameworks.

Um einen Test durchzuführen wird dann in beiden Frameworks die gleiche Abfrage ausgeführt, jeweils mit der korrekten Syntax für die dem Framework zugeordnete Query-Language. Der Test wird nur als erfolgreich und korrekt betrachtet, wenn das Ergebnis dieser Abfrage bei beiden Frameworks das gleiche ist. Diese Tests bilden die Grundlage für die Extraktion der Daten, nach denen die Frameworks bewertet werden.

6.2 Kriterien

Die Kriterien, nach denen die Systeme, die in dieser Arbeit bewertet werden, orientieren sich an einer Internationalen Norm für Qualitätskriterien von Software. Die ISO Norm 25010, auch unter dem Namen SQuaRE bekannt, kategorisiert diverse Eigenschaften von Computersystemen und Softwareprodukten. Diese beinhalten objektiv messbare Eigenschaften, wie den Ressourcenverbrauch, aber auch nicht konkret messbare Eigenschaften, wie die Benutzbarkeit von Software. Die einzelnen Kategorien bilden gemeinsam ein Qualitätsmodell, anhand dessen die Qualität von betrachteten Systemen gemessen werden kann [18]. Die Kategorisierung aller Kriterien, die in dieser Norm enthalten sind, sind in Abbildung 6.1 dargestellt.

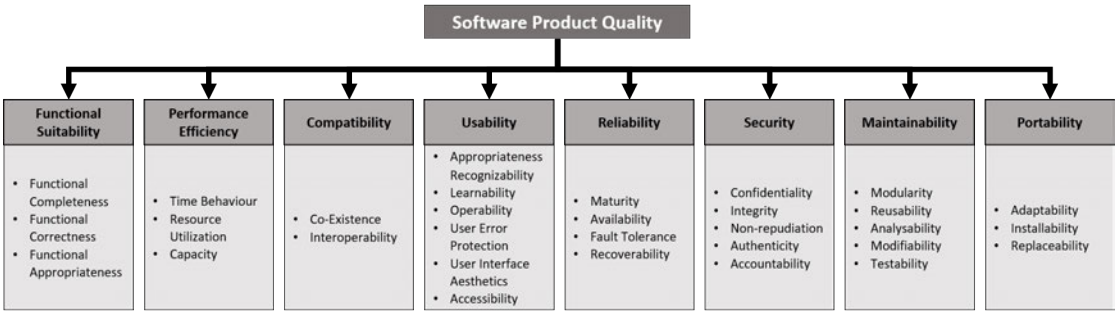

Abbildung 6.1: Kriterien der ISO 25010 (In Anlehnung an [17])

Beide Systeme setzen im Grunde auf der gleichen Technik auf. Wirklich unterschiedlich zwischen ihnen sind vor allem die Art der Client-Anwendung und die Konzepte der Datenspeicherung. Die genaue Betrachtung dieser beiden Aspekte führt zu einer Eingrenzung der Kriterien, nach denen die beiden Frameworks dieser Arbeit bewertet werden.

Die erste Kategorie von Eigenschaften, die bewertet werden soll, ist die Effizienz der Frameworks (Performance Efficiency). Explizit sollen dabei der Ressourcenverbrauch und das Zeitverhalten beleuchtet werden. Mit Ressourcenverbrauch ist vor Allem der Speicherplatz gemeint, den Datenbanken der beiden Frameworks verbrauchen. Die unterschiedlichen Speicherungskonzepte können Auswirkungen auf die Größe von Datenbanken

haben, die in beiden Frameworks jeweils die gleichen Daten repräsentieren. Außerdem kann beobachtet werden, wie sich die Datenbankgröße ändert, wenn die gleichen Daten verändert werden. Um das Zeitverhalten zu beurteilen, werden die Ausführungszeiten bestimmter Operationen verglichen und wie sich diese verändern, wenn Anpassungen sowohl an der Datenbasis als auch an den Operationen selbst vorgenommen werden.

Die zweite Kategorie, die beleuchtet werden soll, ist die Benutzbarkeit (Usability). Um ein System verwenden zu können, sollte die Installation und Einrichtung hinreichend einfach zu erledigen sein. Bei beiden Frameworks wurden die Installationen und Konfigurationen vorgenommen, so dass diese in Kontrast zueinander gesetzt werden können. Ein wichtiger Teil dieser Kategorie bezieht sich auf die Clientanwendungen der Frameworks. Dabei soll beobachtet werden, wie gut sich diese benutzen lassen und wie gut die Prävention von Benutzerfehlern implementiert ist. Zusätzlich wird die Verständlichkeit und Eindeutigkeit der Query-Language bewertet.

6.3 Hypothesen

Um eine gezielte Analyse einzelner Eigenschaften der Frameworks im Bezug auf die vorher benannten Kriterien zu ermöglichen, müssen Hypothesen erstellt werden, die eine Bewertung nach sich ziehen. Die aufgestellten Hypothesen beleuchten jeweils ein Kriterium, welches durch Tests detaillierter betrachtet werden soll.

6.3.1 H1: Die Installations- und Einrichtungszeit von MayBMS ist geringer als die von Trio

Die Softwarearchitektur beider Frameworks unterscheidet sich stark. MayBMS ist direkt in das Server-Backend von PostgreSQL integriert und erfordert damit nur die Installation von PostgreSQL selbst. Trio hingegen setzt nur auf PostgreSQL auf und braucht daher eine zusätzliche Installation des Systems. Daher wird die Zeitdauer, die bis zum Verwenden von Trio nötig ist, größer sein, als die, die bis zum Verwenden von MayBMS.

6.3.2 H2: Die Clientanwendung von Trio ist benutzerfreundlicher als die von MayBMS

MayBMS nutzt als Clientanwendung die von PostgreSQL bereitgestellte Standard-Konsolenanwendung "psql". Trio hingegen verwendet als Konsolenanwendung eine eigens entwickelte Anwendung namens trioplus. Es ist anzunehmen, dass eine eigens für ein Sys-

tem entwickelte Clientanwendung darauf ausgelegt ist, dem Benutzer größeren Komfort und einen individuelleren Funktionsumfang bietet, als ein Standardprodukt.

6.3.3 H3: Welches Framework für Daten mehr Speicher verbraucht, hängt vom Design der Daten ab

Durch die Umsetzung als ULDB werden für Alternativen zwar Lineages gespeichert, allerdings sind dies nach dem Konzept von ULDBs maximal so viele Einträge, wie unsichere Attribute in einem Tupel enthalten sind. Bei den u-Tabellen von MayBMS werden für jede Tabelle eine Spalte angelegt, die für jedes Tupel eine Variable beinhaltet, um deren unterschiedliche Zustände zu beschreiben. Wenn ein Tupel also mit mehreren Zuständen existiert, ist die Speichermenge bei ULDBs in Trio vermutlich größer. Wenn allerdings nicht existente Tupel durch Maybe-Tupel ersetzt werden, wird der Speicherverbrauch bei ULDBs geringer sein als bei u-Tabellen.

6.3.4 H4: Die Berechnungszeiten für Ergebnisse von Abfragen sind bei MayBMS geringer als bei Trio

Die Zeit, die bei der Berechnung des Ergebnisses einer Abfrage verbraucht wird, dürfte auch bei MayBMS geringer sein. Die Architektur der Tabellen in MayBMS führt dazu, dass Daten direkt in den Tabellen gespeichert werden und nicht, wie bei Trio noch in mehreren Tabellen vorgehalten werden, die zur Ergebnisberechnung erst wieder vereint werden müssen.

6.4 Aufbau des Experiments

6.4.1 Ausführung der Testsysteme

Wie bereits in Abschnitt 6.1 beschrieben, wird für die Installation beider Frameworks eine identische Umgebung benötigt. Ausgeführt werden alle Tests innerhalb des gleichen Hostsystems. Dieses ist ein Lenovo ThinkPad T440 mit folgenden Spezifikationen:

- Prozessor: Intel®Core™ i5-4300U (Up to 2.90 GHz, 3MB, L3, 1600 MHz FSB)
- Arbeitsspeicher: 8 GB DDR3 RAM
- Festplatte: 500GB SSD
- Internetverbindung: Download durchschnittlich 80MBit/s, Upload durchschnittlich 20 Mbit/s

Auf diesem Host wird dann das Programm VMWare Workstation Player 15 ausgeführt. Dieses Programm macht es möglich, innerhalb eines Windows- oder Linux-Betriebssystems virtuelle Maschinen laufen zu lassen [38]. Diese besitzen ein eigenes Betriebssystem und stellen virtuell ein komplettes Rechnersystem nach, welches seine Ressourcen aus den Ressourcen des Host-Rechnersystems abzweigt. Die Systemkonfiguration für verwendete Ressourcen, wie die Festplattengröße und die Menge an Arbeitsspeicher ist dem User überlassen. Für die Tests, die in dieser Arbeit ausgeführt werden, bekommen die Testsysteme folgende Ressourcen zugewiesen:

- Anzahl der verwendeten Prozessorkerne: 4
- Arbeitsspeicher: 4 GB
- Festplattengröße: 20 GB
- Internetverbindung: Download durchschnittlich 60MBit/s, Upload durchschnittlich 12 Mbit/s

6.4.2 Softwarekomponenten der Testsysteme

Ubuntu

Als Betriebssystem wurde für beide Systeme Ubuntu gewählt. Ubuntu ist eine Linux-Distribution, die von der Firma Canonical Ltd. angeboten wird. Ubuntu basiert auf Debian und wurde mit dem Ziel entwickelt, darauf ein leistungsfähiges und verständliches Linux anzubieten [13]. Für die virtuellen Maschinen, die für diese Arbeit zum Testen verwendet werden, wurde sich bei der Wahl des Betriebssystem für Ubuntu 16.04 als 32-Bit-Version entschieden. Für MayBMS wird zwar ein 64-Bit Installer als Release angeboten, allerdings enthält dieser noch den Prototypen MayBMS1 anstatt der Version MayBMS2, weshalb hier die 32-Bit Version des Betriebssystems gewählt wurde.

PostgreSQL

PostgreSQL ist ein relationales DBMS. Ursprünglich aus einem Projekt der Berkeley University im Jahre 1986 hervorgegangen, ist dieses seit 1995 als Open-Source-Projekt veröffentlicht, welches mittlerweile von der "PostgreSQL Global Development Group" gemanagt wird [28]. PostgreSQL bietet dem User nicht nur den vollen eines DBMS unter Verwendung der relationalen Algebra, sondern auch weitere Features, wie die Möglichkeit, eigene Datentypen zu definieren [26]. Auch hier war die Auswahl der Version recht

einfach. MayBMS ist fest integriert in das Server-Backend von PostgreSQL und wird daher mit der Version 8.3.3 ausgeliefert. Daher wurde diese Version auch als Basis für Trio verwendet. Aufgrund der Tatsache, diese Version allerdings nicht mehr als Installationspaket zum Download zur Verfügung steht, muss PostgreSQL aus dem heruntergeladenen Quellcode neu erstellt werden.

Python

Python ist eine interpretierte, objektorientierte Programmiersprache, die sich durch hohe Flexibilität auszeichnet. Sie besitzt viele Interfaces zu anderen Softwareprodukten, läuft auf den meisten Betriebssystemen und ist in C oder C++ erweiterbar. Dadurch wird sie oft zum Erstellen von ausführbaren Objekten verwendet, wie auch bei den beiden evaluierten Frameworks [31]. In den Testumgebungen wird die Version 2.7.12 verwendet, welche im Betriebssystem Ubuntu 16.04 vorinstalliert ist.

6.5 Durchführung

6.5.1 Messung der Installationszeit

Struktur der Durchführung

Das erste Experiment, dass durchgeführt wird, ist die Messung der Installationszeit. Damit soll H1 bestätigt werden. Die Messung wird drei mal durchgeführt und die Ergebnisse dokumentiert. Erkenntnisse aus einem Installationsvorgang werden in den nächsten Installationsvorgang übernommen, weil diese dann als Vorwissen angesehen werden können, welches eine eventuelle Fehlersuche und weitere Vorbereitung verbessert und beschleunigt. Als Beginn der Messung wird der Zeitpunkt gewählt, an dem alle nötigen Voraussetzungen zur Installation geschaffen wurden. Das bedeutet, dass sämtliche in den Systemvoraussetzungen definierten Softwarepakete installiert sind und die Quellcodepakete zur Installation der Frameworks heruntergeladen sind. Das Ende der Messung wird an dem Zeitpunkt festgesetzt, an dem zwei Beispielabfragen korrekt ausgeführt wurden, die im folgenden definiert werden sollen. Als erste Beispielabfrage soll die Funktionalität der reinen relationalen Algebra überprüft werden. Dazu werden die SQL Befehle aus Listing 6.1 verwendet, um zu überprüfen, ob die Umsetzung der Befehle auch korrekt geschehen ist. Das Ergebnis der letzten Abfrage muss jeweils ein Tupel mit einem Wert “OK” sein.

```
1 Create table test (IsOK varchar);
```

```
2 Insert Into test values ('OK');
3 Select * from test;
```

Listing 6.1: Beispielabfragen für den ersten Funktionstest

Mit der zweiten Beispielabfrage sollen die Funktionalitäten für unsichere Daten der beiden Frameworks getestet werden. Dafür soll eine unsichere Tabelle mit einem unsicheren Tupel erzeugt werden. Das Tupel soll zwei Alternativen haben, wobei die Confidence-Werte für die Alternativen jeweils $p_1 = 0.3$ bzw. $p_2 = 0.7$ sind. Eine Abfrage der Confidence für die Alternative, die "1" als Wert für "isValid" enthält, soll dann als Ergebnis 0,7 zurückgeben. Die Statements für die Abfragen sind in den Listings 6.2 und 6.3 dargestellt.

```
1 Create table testUncertain (ID int, valid int, p float);
2 Insert Into testUncertain values (1, 1, 0.7);
3 Insert Into testUncertain values (1, 0, 0.3);
4 Create table testUncertain_u as (
5         repair key (ID) in testUncertain weight by p
6 );
7 Select conf() from testUncertain_u where valid = 1;
```

Listing 6.2: Beispielabfragen für den zweiten Funktionstest bei MayBMS

```
1 Create trio table testUncertain (id int, isValid int, uncertain
     (isValid)) with confidences;
2 Insert into testUncertain values (1:1, [ 0:0.3 | 1:0.7 ]);
3 Create table testUncertain_u as (
4         repair key (ID) in testUncertain weight by p
5 );
6 Select conf() from testUncertain_u where valid = true;
```

Listing 6.3: Beispielabfragen für den zweiten Funktionstest bei Trio

Ablauf und Ergebnis

Begonnen wurde die Messung mit den Installationen von MayBMS. Als Installationsanleitung für dieses Framework wird der Einfachheit halber empfohlen, den fertigen Installer zu verwenden. Wenn dies getan wird, werden keine weiteren Pakete vorausgesetzt, die zur Installation benötigt werden. Daher wurde die erste Messung begonnen, nachdem

der Installer heruntergeladen und entpackt wurde. Die Ausführung des Installers verlief problemlos und ohne Fehlermeldungen. Bei der Ausführung des PostgreSQL Servers wurde allerdings auf ein fehlendes Paket hingewiesen, welches daraufhin nachinstalliert wurde. Bei der Ausführung der Clientanwendung trat ein Fehler wegen einer nicht gefundenen Bibliothek auf, welcher durch erstellen eines Verweises behoben werden konnte. Nachdem diese beiden Fehler behoben wurden, konnten die Testabfragen erfolgreich ausgeführt werden. Bei der zweiten und dritten Messung wurde das fehlende Paket, welches die Ausführung des PostgreSQL Servers in der ersten Installation verhinderte, installiert, bevor die Messung gestartet wurde. Der Verweis, um den zweiten Fehler zu beheben, konnte erst erstellt werden, nachdem MayBMS installiert war. Trotzdem war dieses Problem im Vorfeld bekannt und konnte ohne weitere Recherche direkt gelöst werden. Diese Erkenntnisse führten dazu, dass die zweite und dritte Messung kürzer sind, als die erste. Dies ist am blauen Graph in Abbildung 6.2 gut zu erkennen.

Versuch	MayBMS	Trio
1	0:15:01	2:01:12
2	0:04:58	0:17:36
3	0:05:45	0:12:13

Tabelle 6.1: Ergebnisse der Messungen als Zeitdauer

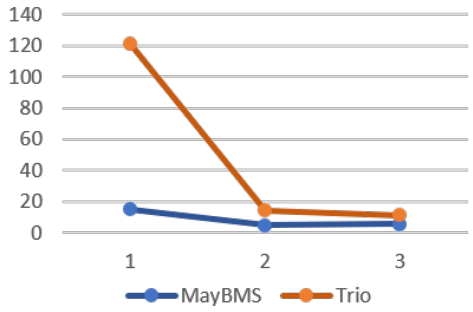

Abbildung 6.2: Installationszeiten in Minuten

Als nächstes wurden Messungen der Installationen von Trio durchgeführt. Noch vor Beginn der Messung traten mit der Installation von PostgreSQL die ersten Probleme auf. Da die Version 8.3.3 nicht mehr mit einem Paketmanager zu installieren ist [29], musste die Version aus dem Quellcode neu erstellt werden. Dabei wurde der mitgelieferten Installationsanleitung von PostgreSQL gefolgt, was dazu führte, dass der Socketpfad nicht dem regulären Einrichtungszustand entsprach. Nachdem dann weitere Voraussetzungen geschaffen wurden, wie die Installation des Pakets "setuptools" und dem Python Paket-Manager "pip", wurde die Messung begonnen. Der erste Versuch, der Anleitung zu folgen, und Trio mit dem Kommando "easy-install" zu installieren, scheiterte, weil einige Abhängigkeiten nicht aufgelöst werden konnten. Daher wurde sich dazu entschieden, auch das Trio-Framework aus dem Quellcode neu zu erstellen. Dies gelang nach Nachinstallieren

einiger Pakete, die durch Fehler bei der Erstellung ersichtlich wurden auch. Mit der Installation war die Einrichtung aber noch nicht abgeschlossen. Der PostgreSQL-Server konnte zwar gestartet werden, aber trioplus nicht ausgeführt werden. Um eine Ausführung von trioplus zu erreichen, musste in diverse Quelldateien der Anwendung eingegriffen werden, um die Zielpfade für den Server anzupassen. Damit ist trioplus ausführbar gewesen und in der Lage reine SQL-Abfragen an PostgreSQL weiter zu leiten. Um aber TriQL-Abfragen weiterzuleiten, müssen die Trio-Metadaten in der Datenbank initialisiert werden. Auch hier waren wieder diverse Recherchen und Anpassungen an Quelldateien nötig, damit das Initialisieren der Metadaten reibungslos auf einer Datenbank funktioniert. Nachdem die Fähigkeit für trioplus theoretisch hergestellt war, TriQL-Abfragen zu verarbeiten, traten aber wiederum Fehler in der Umformung der TriQL-Abfragen in SQL-Abfragen auf, die es verhindert haben, die zweiten Testbefehle auszuführen. Wiederum Änderungen in den Umformungsklassen des Quellcodes brachten hier den Erfolg. Alles in Allem hat damit die Installation des Frameworks bis zum Ende der Messung durch Abschluss der zweiten Testabfragen mehrere Stunden gedauert.

Sämtliches Wissen über die Fehler und Vorkonfigurationen wurden bei der zweiten und dritten Installation mit bedacht. Hier wurde die Installation fehlender Pakete nicht mit eingerechnet. Die Änderungen, die am Quellcode und den Konfigurationen des Frameworks nötig waren, wurden zwar in die Installationszeit eingerechnet, jedoch entfällt sämtliche Recherche dazu. Daher sind auch bei Trio die zweite und dritte Messung signifikant schneller durchgeführt worden als die erste, wie man am orangen Graph in Abbildung 6.2 sehen kann.

6.5.2 Messung der Abfragedauer

Struktur der Durchführung

Das zweite Experiment hat den Vergleich der Abfragedauer zum Zweck. Dabei soll die Zeit einer Abfrage gemessen werden, die in beiden Frameworks ausführbar ist. Der Wert, der als Dauer anerkannt wird, ist der, den die jeweilige Clientanwendung als Dauer angibt. Als Basis für die Abfrage wird das Beispiel der zufälligen Graphen aus dem MayBMS-Manual abgewandelt und verändert [24]. Dabei wird ein ungerichteter Graph mit n Knoten erzeugt, bei dem jeder Knoten mit jedem Anderen verbunden ist. Ein Beispiel für einen solchen Graphen mit $n = 5$ ist in Abbildung 6.3 zu sehen. Jede Kante des Graphen ist allerdings nicht sicher in der Datenbank enthalten, sondern nur zu einer gewissen Wahrscheinlichkeit $0 < p < 1$. Nun sollen im Graphen k-Kreise gefunden werden. Ein Kreis in einem Graphen ist ein Weg, bei dem der Start- und Endknoten gleich sind [21].

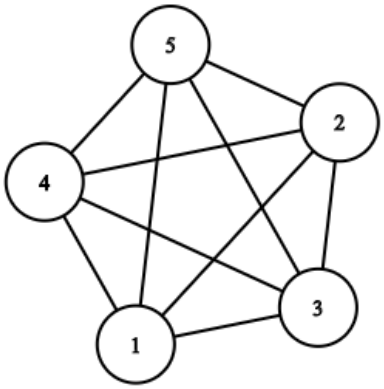

Abbildung 6.3: Beispiel für einen ungerichteten Graphen

Wenn eine Kante $e = (u, v)$ definiert durch ihren Startknoten u und ihren Endknoten v ist, dann wäre in Abbildung 6.3 der Weg $((1, 2), (2, 3), (3, 1))$ ein gültiger 3-Kreis oder auch ein Dreieck. Für alle diese Kreise gibt es eine Wahrscheinlichkeit, mit der sie in der Datenbank vorhanden ist, die sich aus den Existenzwahrscheinlichkeiten der enthaltenen Kanten berechnen lässt. Die Abfrage, die als Test für dieses Experiment gewählt wurde, beinhaltet sowohl SQL-Bestandteile, als auch Elemente der jeweiligen Query-Language des Frameworks. Hier werden alle k-Kreise ausgewählt, die in dem Graphen enthalten sind zusammen mit der Confidence, dass sie wirklich im Ergebnis enthalten sind. Die Zeit, die diese Abfrage braucht, ist das vergleichbare Ergebnis. Die Testergebnisse lassen sich voraussichtlich durch verschiedene Parameter beeinflussen. Diese Parameter sind:

- die Anzahl der im Graphen enthaltenen Knoten n
- die Wahrscheinlichkeit, dass eine Kante im Graphen enthalten ist p
- die Anzahl der Elemente der gesuchten Kreise k.

Ablauf und Ergebnis

Um für beide Frameworks die gleiche Datenbasis zu erstellen und immer die exakt gleichen Graphen zu erzeugen, wurde eine Konsolenanwendung in C# erstellt. Diese Anwendung beinhaltet zwei Funktionen. Auf Wunsch kann ein Graph erzeugt werden, bei dem die

drei beschriebenen Parameter selbst gewählt werden können. Dann wird für beide Frameworks eine Datei erzeugt, die eine komplette Skriptabfolge enthält. Diese beinhaltet sowohl das Anlegen des Graphen als auch die für dieses Experiment verwendete Testabfrage, jeweils in der entsprechenden Query-Language des Frameworks. Wenn dies nicht gewünscht wird, dann werden diverse Skripte für alle Graphen mit Permutationen von $p = 0.5$, $n = \{i|3 \leq i \leq 10\}$ und $k \in \{3, 4, 5\}$ erzeugt und ebenfalls als Textdateien abgelegt. Diese Permutationen bilden auch bei beiden Frameworks die Grundlage für dieses Experiment. Für alle Permutationen wurden Zeitmessungen jeweils in MayBMS und in Trio durchgeführt. Für MayBMS wird die Messung durchgeführt auf eine mit dem repair-key-Statement erzeugte u-Tabelle. Dafür Für Trio wird die Messung in zwei Varianten durchgeführt. Die erste Variante ist eine ULDB, bei der die Aussage, ob eine Kante in dem Graphen enthalten ist, oder nicht, als Alternative enthalten ist. Die zweite Variante beinhaltet nur eine Alternative für enthaltene Kanten, und stellt jede Kante dadurch als ein Maybe-Tupel dar. So kann ein eventueller Unterschied in der Performance beider Varianten beobachtet werden. Ein Messergebnis für dieses Experiment besteht aus dem Durchschnitt aus den Ergebnissen von drei Versuchen der gleichen Abfrage. Die Ergebnisse der Messungen sind in den Tabellen 6.2, 6.3 und 6.4 aufgetragen

n \ k	**3**	**4**	**5**
3	0,77733	1,25367	1,79000
4	0,81333	1,38233	2,57200
5	0,97267	1,89333	3,13767
6	1,48067	3,15633	7,38933
7	1,85000	6,10533	24,01733
8	2,52700	9,04867	84,17033
9	3,35867	22,68133	134,63843
10	4,47467	38,31867	242,40100

Tabelle 6.2: Ergebnisse der Zeitverbrauchsmessung bei MayBMS

6.5.3 Messung des Speicherbedarfs

Struktur der Durchführung

Ein weiteres Experiment soll der Vergleich des Speicherbedarfs beider Systeme angestellt werden. Dieses Experiment soll grundlegende Daten für Hypothese H3 liefern. Der Speicherbedarf der Frameworks wird hier synonym betrachtet mit der Größe der Datenbank, auf der das Framework seine Operationen ausführt. Das Ergebnis einer Messung

n \ k	3	4	5
3	25,43600	25,19171	13,11000
4	160,07765	133,20494	38,16089
5	136,84003	325,19937	328,56369
6	275,35407	836,52035	1854,23366
7	397,55368	1534,22228	5547,52199
8	560,88567	2867,05510	13694,77876
9	777,18830	5000,72265	28786,69771
10	1070,02600	8265,81462	57757,85398

Tabelle 6.3: Ergebnisse der Zeitverbrauchsmessung bei Trio mit Alternativen

n \ k	3	4	5
3	25,22922	9,97345	14,38594
4	85,34702	86,27367	14,75700
5	137,57602	263,77535	278,30593
6	223,81870	684,19528	961,95594
7	332,07838	1549,27564	4869,37737
8	493,87693	2787,85467	12928,76760
9	745,31142	4923,19163	28551,28447
10	1052,28599	8330,14965	56090,36477

Tabelle 6.4: Ergebnisse der Zeitverbrauchsmessung bei Trio mit Maybe-Tupeln

ist die Größe der Datenbank in Kilobytes. Diese wird ermittelt, indem für beide Frameworks auf dem jeweiligen Testsystem die Konsolenanwendung psql ausgeführt wird und mit darin Befehl aus Listing 6.4 ausgeführt wird. Dabei wird statt “dbname” der korrekte Datenbankname eingetragen. Um die Genauigkeit zu erhöhen und Ausreißer in den Messergebnissen zu verhindern, wird jede Messung dreifach durchgeführt und der Durchschnitt der Messergebnisse als Ergebnis betrachtet.

```
1 Select pg_size_pretty( pg_database_size('dbname') );
```

Listing 6.4: SQL-Befehl zum Abfragen der Datenbankgröße

Das Experiment selbst besteht aus mehreren Teilen.
Im ersten Teil soll die grundlegende Größe einer Datenbank ohne Daten für beide Frameworks verglichen werden. Dafür werden in beiden Frameworks leere Datenbanken erzeugt und so initialisiert, dass sie für das Befüllen mit unsicheren Daten geeignet sind. Die Größe dieser leeren Datenbanken bildet einen Grundvergleich über den Speicherverbrauch

beider Frameworks.
Im zweiten Teil dieses Experiments wird der Speicherbedarf unter Berücksichtigung des Datendesigns betrachtet. Dafür werden die Daten zur Beschreibung eines Graphen aus Abschnitt 6.5.2 verwendet. Auch hier werden wieder die drei Speicherungsvarianten der Graphen gegenüber gestellt.

Ablauf und Ergebnis

Begonnen wurde mit der Speicherverbrauchsmessung von leeren Datenbanken. Da MayBMS in PostgreSQL integriert ist, ist dieses einsetzbar, sobald eine Datenbank erschaffen wurde. Trio hingegen erfordert eine Initialisierung von Metadaten auf einer Datenbank, ehe sie für das Framework verwendet werden kann. Die Größe einer leeren Datenbank für MayBMS ist die Größe einer leeren PostgreSQL-Datenbank. Diese beträgt 4296 kB. Bei Trio muss die Größe der Datenbank noch um die Größe der Trio-Features erhöht werden. Eine leere Trio-Datenbank ist 4327 kB groß, damit entfallen 31 kB auf die Trio-Features. Danach wurden dann noch einmal die zufälligen Graphen als Datensätze in die Datenbanken gebracht. Zu diesem Zweck wurde der Scriptgenerator aus Abschnitt 6.5.2 erweitert, so dass auch hier wieder mit einer genormten Datenbasis für beide Frameworks getestet werden konnte. Dabei wurden jeweils Graphen mit n Knoten erzeugt und bei jeder Erhöhung die Veränderung an der Datenbankgröße beobachtet. Die Messergebnisse sind in Tabelle 6.5 und dem dazugehörigen Graph in Abbildung 6.4 aufgetragen. Dabei wurde zuerst der Punkt gesucht, an dem die Frameworks das erste Mal eine unterschiedlich große Änderung im Speicherverbrauch bei Erhöhung der Knoten im Graph bewirken. Ab dann wurde zuerst mit einer Schrittweite von 5 Knoten pro Schritt fortgefahren, um zu beobachten, wie sich im Nahbereich der Speicherverbrauch ändert. Nach $n = 25$ wurden die Schrittweiten erhöht, um die Veränderungen im verbrauchten Speicherplatz besser beobachten zu können.

Teile des Graphen		Speicherverbrauch der Datenbank in kB		
n	Kanten	MayBMS	Trio (Alternativen)	Trio (Maybe-Tupel)
2	4	4328	4479	4479
3	12	4328	4479	4479
4	24	4328	4479	4479
5	40	4328	4479	4479
6	60	4328	4479	4479
7	84	4328	4479	4479
8	112	4328	4479	4479
9	144	4328	4479	4479
10	180	4336	4487	4479
15	420	4344	4503	4495
20	760	4360	4543	4511
25	1200	4384	4599	4559
50	4900	4576	4951	4799
75	11100	4888	5559	5199
100	19800	5336	6399	5759

Tabelle 6.5: Ergebnisse der Speicherverbrauchsmessungen für Graphen

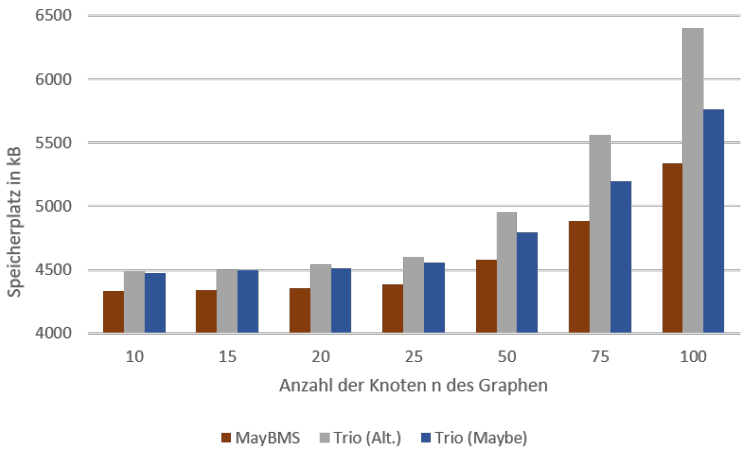

Abbildung 6.4: Graphische Darstellung der Messergebnisse für den Speicherverbrauch)

6.6 Auswertung der Ergebnisse

Nach der Durchführung der Experimente können beide Frameworks nun auf Basis der Hypothesen bewertet werden. Dazu werden die Frameworks natürlich einerseits im Hinblick auf die gestellten Hypothesen betrachtet. Zusätzlich soll aber auch ein Realitätsbezug

hergestellt werden, um die Verwendbarkeit der Frameworks in der Realität zu illustrieren.

Das erste durchgeführte Experiment galt der Installationszeit beider Frameworks. Um ein System produktiv nutzen zu können, sollte es in der Installation und Einrichtung hinreichend einfach sein. Die Länge der Installationszeit kann dafür als Indikator dienen. Im Verlauf der Durchführung konnte beobachtet werden, dass die Ausführung der jeweiligen Installation der Frameworks selbst nur einen Teil der Gesamteinrichtungszeit in Anspruch nahm.

Dies gilt sowohl für MayBMS als auch für Trio. Das bedeutet umgekehrt, dass Nacharbeiten, Einrichtung und Anpassungen ebenfalls einen nicht unerheblichen Anteil daran haben. Bei MayBMS funktionierte dies relativ reibungslos und die Anpassungen, die getroffen wurden, waren im ersten Installationsvorgang so offensichtlich, dass diese auch von nicht allzu erfahrenen Benutzern durchgeführt werden können. Das führt zu einer Reduzierung der Installationszeit durch bessere Vorbereitung. Insgesamt ist mit einer Installationsdauer von ca. 5 Minuten ein adäquater Wert zustande gekommen, der sich mit heutigen Erwartungen an eine solche Software deckt.

Im Falle von Trio ist diese Aussage nicht zu treffen. Hier ist die Differenz zwischen dem ersten und den zweiten beiden Installationsvorgängen enorm. Das Einrichten des Frameworks bis zur Benutzung erfordert signifikante Änderungen am System und am Quellcode. Diese können nur von erfahrenen Technikern oder Programmierern durchgeführt werden und würden einen Laien durchaus überfordern. Die Anpassungen sind recht umfangreich, so dass auch der Prozess der Installation an sich immer weniger Benutzerfreundlich wird. Auch, wenn mit einer knappen Viertelstunde die Installationszeit sich noch im Rahmen des Annehmbaren befindet, ist der Vorgang der Installation selbst belastend.

Betrachtet man die Installationszeiten im direkten Vergleich, so kann die Hypothese H1 aber auf für diese Vorgänge bestätigt werden. Es kann allerdings sein, dass die Entwickler dieser Frameworks die Installation deutlich schneller durchführen können, weil sie mit den Problemen der Produkte vertraut sind.

Der nächste Punkt, der genauer analysiert werden soll, ist die Benutzbarkeit beider Frameworks. Dazu werden explizit die Clientanwendungen unter den in Abschnitt 6.2 genannten Kriterien betrachtet. Bei beiden Frameworks wurden hierbei ausschließlich die Konsolenanwendungen betrachtet.

Die erste Gemeinsamkeit, die dadurch entwickelt wird, besteht im Kriterium der User Interface Aesthetics. Da beides Konsolenanwendungen sind, besitzen auch beide nur die Möglichkeit, Tabellen in ASCII-Zeichen dar zu stellen. Diese Gemeinsamkeit hat

zur Folge, dass auch die Accessibility, also die Benutzungsmöglichkeit für Menschen mit unterschiedlichen Fertigkeiten sich in ihrem Umfang sehr ähnelt. Eine Benutzung für Menschen mit Augenproblemen wie Blindheit und Analphabeten ist schlichtweg nicht möglich. Dafür müsste das Betriebssystem eine entsprechende Audioschnittstelle bereit stellen, was aber dann nichts mehr mit den Frameworks zu tun hat.

Wirkliche Unterschiede zeigen sich vor allem in der Benutzung beider Frameworks, beginnend mit dem Schützen des Users vor Fehlern. Auf Seiten von MayBMS gibt es mit psql eine Anwendung, die Komfortfeatures wie eine Autovervollständigung für reine SQL-Befehle und eine adäquate Syntaxprüfung beinhaltet. Diese gibt dem User eine genaue Antwort, wo in seiner Abfrage ein Fehler steckt.

Auf Seiten von Trio ist dies nicht gegeben. Eine Autovervollständigung wird vergebens gesucht. Die Fehlermeldungen sind häufig ungenau und beinhalten Nachrichten wie "Syntax error near ';' ", ohne eine Aussage zu treffen, was genau falsch ist. Auch sonst ist die Benutzung von trioplus nicht sehr intuitiv. Die Komplexität der Sprache macht es nicht einfacher, wobei zusätzlich gesagt sein muss, dass das Language Manual zu TriQL [37] nicht zu den Abfragen passt, die wirklich in Trio eingegeben werden müssen. Es scheint veraltet und wird allem Anschein nach auch nicht mehr gepflegt. Hinzu kommt, dass zu jeder Datenbank, die in Trio verwendet werden soll, manuell die Metadaten für diese Datenbank angelegt werden muss

Alleine diese Aspekte zeigen, dass nicht zu einem produktiven Einsatz von Trio zu raten ist. Die Clientanwendung, die im Standard benutzt wird, ist nicht auf einem Niveau, welches es ermöglicht, dass auch nicht allzu erfahrene Benutzer sie verwenden. Im Vergleich dazu ist MayBMS minimal besser aufgestellt. Die Clientanwendungen gleichen sich fast im Bezug auf ihre Ästhetik, aber psql bringt doch einige Komfortfeatures mit, die die Benutzung vereinfachen und den Benutzer besser unterstützen. Damit kann die Hypothese H2 als widerlegt betrachtet werden.

Nachdem die benutzerbezogenen Aspekte der Frameworks diskutiert wurden, sollen nun die Ergebnisse zu den technischen Kriterien ausgeführt werden. Dabei soll mit dem Ressourcenverbrauch begonnen werden.

Bei einer Analyse der Ergebnisse aus Abschnitt 6.5.3 lassen sich die Systeme sehr akkurat vergleichen. Der Speicherverbrauch einer leeren Datenbank ist, wie bereits beschrieben, aufgrund der zusätzlichen Metadaten in Trio höher. Je größer der Datensatz wurde, desto höher wurde logischerweise auch der Speicherverbrauch beider Frameworks. Die Differenz im Speicherverbrauch zwischen den einzelnen Testschritten ist bei Trio aber höher als bei MayBMS. Das hat den Grund, dass bei MayBMS alle Daten einer u-Tabelle auch nur in einer relationalen Tabelle vorgehalten werden. Bei Trio hingegen wird der In-

halt einer ULDB auf mehrere Tabellen aufgeteilt, so dass der Speicherverbrauch für einen gleichen Datensatz wie in MayBMS aber trotzdem größer ist. Hinzu kommt auch noch, dass während des Experiments keine Lineage-Daten in der Datenbank enthalten waren. Diese hätten die Datenbank nur weiter vergrößert und für noch größere Unterschiede zu MayBMS gesorgt.

Damit ist Trio rein vom Ressourcenverbrauch her auch nicht wirklich geeignet, um produktiv mit großen Datenmengen damit arbeiten zu können. Schlaues Datendesign kann das Ressourcenmanagement verbessern, aber im Vergleich zu MayBMS, welches einen Konstanten Anstieg beobachten konnte, der keine weiteren Abhängigkeiten beinhaltete, war der Anstieg trotzdem größer. Im Bezug auf die Hypothese H3 kann das als Bestätigung gesehen werden.

Das letzte Kriterium, was es zu betrachten gilt, ist der Zeitverbrauch. Dabei können die Ergebnisse aus 6.5.2 als Bestätigung von Hypothese H4 verstanden werden.

Bei der Betrachtung der Ergebnisse in Trio fällt zuerst auf, dass es keinen Unterschied macht, ob der Graph als ULDB mit Alternativen oder mit Maybe-Tupeln gespeichert wurde. Die Abfragezeiten zwischen diesen beiden Varianten ähneln sich sehr stark. In beiden Fällen müssen die Tabellenverknüpfungen mit den Untertabellen der ULDB für Alternativen und Lineage hergestellt werden, so dass die Zeit dafür bei beiden Varianten verbraucht wird.

Wenn diese Ergebnisse allerdings mit den Ergebnissen aus MayBMS verglichen werden, zeigt sich schnell der Vorteil der Datenbankarchitektur von MayBMS. Das Speichern der Daten in nur einer Tabelle minimiert die Join-Operationen. Während bei einem Join über drei Tabellen in MayBMS auch wirklich nur diese drei Tabellen gejoint werden, wird in Trio ein Join über neun Tabellen durchgeführt.

Die Menge an Daten, die in diesem Experiment verarbeitet wurde, ist verhältnismäßig klein. Angenommen, es wird das Beispiel der Sensormessung aus Abschnitt 1.1 aufgegriffen und es gäbe drei Tabellen, in denen jeweils ein Sensor seine Daten hinterlegt. Bei einer Abfrage mit einem Join über diese drei Tabellen hätte die grundlegende Ergebnismenge $4{,}32^3$ Millionen Datensätze. Wenn Trio für die im Experiment verwendete Datenmenge, die deutlich geringer war, bis zu einer Minute braucht, kann das nicht der Anspruch sein, der an eine produktiv eingesetzte Software gestellt wird. Damit könnte höchstens MayBMS dafür verwendet werden, nicht aber Trio.

Bei Betrachtung aller dieser Ergebnisse können Schlüsse auf die Eignung beider Frameworks für die Speicherung und Verarbeitung von unsicheren Daten gezogen werden. Technisch gesehen sind beide Frameworks in der Lage, mit unsicheren Daten umzugehen.

Beide Frameworks sind auch fähig, probabilistische Daten zu speichern, oder Daten, die in mehreren Varianten in unterschiedlichen Mengenverhältnissen vorkommen.

Vergleicht man beide Frameworks aber näher, wird deutlich, dass Trio nicht für den produktiven Einsatz geeignet ist. Dazu sind die Schwächen dieses Frameworks einfach zu groß. Die Performanceprobleme machen es ineffizient für den Einsatz und die Benutzbarkeit leidet durch die nur befriedigende Clientanwendung auch.

Im Vergleich dazu hat MayBMS gezeigt, dass es die nötigen Eigenschaften besitzt, als produktives DBMS für unsichere Daten eingesetzt zu werden. Trotzdem ist das Produkt sehr veraltet und die Technologie in der Zwischenzeit sehr weit fortgeschritten. Damit ist eine Kompatibilität mit aktuellen Systemen nicht garantiert und muss im Vorfeld untersucht werden.

7 Fazit

Das Ziel dieser Arbeit war eine Evaluation von Frameworks zur Verarbeitung und Speicherung unsicherer Daten. Anhand dieser Evaluation sollte die Nutzbarkeit im Produktivbetrieb beider Frameworks bewertet werden.

Dafür wurden die beiden Frameworks Trio und MayBMS gewählt und analysiert. Die Evaluation wurde anhand diverser Experimente durchgeführt. In den Experimenten hat sich herausgestellt, dass Trio vor allem in den Punkten Performance und Benutzbarkeit Defizite hat. Diese Defizite hat MayBMS nicht. Daher kann eine Empfehlung für MayBMS für Produktivsysteme ausgesprochen werden. Diese Empfehlung sollte aber trotzdem selbst beurteilt werden, da eventuell Kompatibilitätsprobleme bestehen.

7.1 Zusammenfassung

Als erstes wurden in dieser Arbeit grundlegende Informationen zu Datenbanken vermittelt. Es wurden Begrifflichkeiten definiert und die grundlegenden Operatoren der relationalen Algebra wurden vorgestellt.

Anschließend wurde der Fokus auf unsichere Daten gelegt. Dabei wurde vor allem das Konzept der Unsicherheit definiert. Außerdem wurden weitere grundlegende Konzepte wie die Semantik der möglichen Welten und die OWA und CWA erläutert.

Im nächsten Abschnitt wurde das erste Framework, Trio, vorgestellt. Es wurden die grundlegenden Speicherungsmechanismen und der Umfang der Funktionalitäten und mitgelieferten Pakete dargelegt und auf die Query-Language TriQL eingegangen.

Nachdem Trio vorgestellt wurde, wurde das zweite Framework, MayBMS, präsentiert. Auch hier wurden zu Beginn die Speicherungskonzepte, Schritt für Schritt auf Codd-Tabellen aufbauend erklärt. Zusätzlich gab es auch hier einen kurzen Exkurs in die Query-Language.

Als letztes wurde anhand von Experimenten eine Evaluation durchgeführt. Es wurden Hypothesen aufgestellt, welche dann durch Messungen bestätigt oder widerlegt wurden Anhand dieser Experimente konnte dann eine Bewertung beider Frameworks und ein Vergleich zwischen ihnen statt finden.

7.2 Ausblick

Die Popularität des Feldes der unsicheren Daten scheint in den letzten Jahren gesunken zu sein. Die letzten Frameworks, die entwickelt wurden, sind veraltet und neue lassen auf sich warten. Trotzdem könnte man die Evaluation noch für andere Frameworks, wie beispielsweise das ORION Framework [30] durchgeführt werden.

Außerdem könnte der Datenbestand erweitert werden, unter dessen Verwendung die Frameworks evaluiert werden. Die in Abschnitt 6.1 genannten Beispiele sind dabei als Referenz zu sehen. Hiermit könnte das Verhalten der Frameworks unter Verwendung komplexerer Datensätze evaluiert werden.

Für Anwendungsfälle, in denen es mit unsicheren Daten zu arbeiten gilt, können auch interessante Konzepte für relationale Datenbanken gefunden werden. Diese sind dann natürlich nicht in der Lage, mögliche Welten zu repräsentieren. Der nächste logische Schritt wäre dann, die Ergebnissicherheit der Frameworks mit aktuellen Forschungen zu vergleichen, die relationale Datenbanken für die Speicherung und Verarbeitung unsicherer Daten verwenden. Hierzu gibt es mittlerweile diverse Modelle, die unter anderem über Data-Cleaning unsichere Daten vereinfachen und damit repräsentieren können.

Literaturverzeichnis

[1] Abiteboul, Serge ; Hull, Richard ; Vianu, Victor: *Foundations of databases*. Bd. 8. Addison-Wesley Reading, 1995

[2] Abiteboul, Serge ; Kanellakis, Paris ; Grahne, Gosta: On the Representation and Querying of Sets of Possible Worlds. In: *Theoretical Computer Science* Bd. 78, 01 1987, S. 34–48

[3] Antova, Lyublena ; Koch, Christoph ; Olteanu, Dan: 10^ (10^6) Worlds and Beyond: Efficient Representation and Processing of Incomplete Information. In: *Data Engineering (ICDE)* (2007), S. 7

[4] Antova, Lyublena ; Koch, Christoph ; Olteanu, Dan: MayBMS: Managing incomplete information with probabilistic world-set decompositions. In: *2007 IEEE 23rd International Conference on Data Engineering* IEEE (Veranst.), 2007, S. 1479–1480

[5] Antova, Lyublena ; Koch, Christoph ; Olteanu, Dan: Query language support for incomplete information in the MayBMS system. In: *VLDB*, 2007

[6] Beaulieu, Alan: *Learning SQL*. Beijing Sebastopol : O'Reilly, 2009. – ISBN 978-0-596-52083-0

[7] Benjelloun, Omar ; Sarma, Anish D. ; Halevy, Alon ; Widom, Jennifer: ULDBs: Databases with uncertainty and lineage / Stanford. 2005. – Forschungsbericht

[8] Buneman, Peter ; Khanna, Sanjeev ; Wang-Chiew, Tan: Why and where: A characterization of data provenance. In: *International conference on database theory* Springer (Veranst.), 2001, S. 316–330

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

[10] Codd, Edgar F.: A relational model of data for large shared data banks. In: *Communications of the ACM* 13 (1970), Nr. 6, S. 377–387

[11] Codd, Edgar F.: Extending the database relational model to capture more meaning. In: *ACM Transactions on Database Systems (TODS)* 4 (1979), Nr. 4, S. 397–434

[12] Cui, Yingwei ; Widom, Jennifer: Practical lineage tracing in data warehouses. In: *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)* IEEE (Veranst.), S. 367–378

[13] Elsner, Tobias ; Erker, Thomas ; Lingnau, Anselm: *Linux kompakt*. tuxcademy, 2016

[14] Haggard, Gary: *Discrete mathematics for computer science*. Belmont, CA : Thomson Brooks/Cole, 2006. – ISBN 053449501X

[15] *IMDb data files available for download*. https://datasets.imdbws.coml. 2020. – Zugriffsdatum: 2020-05-12

[16] Imieliński, Tomasz ; Lipski Jr, Witold: Incomplete information in relational databases. In: *Readings in Artificial Intelligence and Databases*. Elsevier, 1989, S. 342–360

[17] *ISO/IEC 25010*. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. 2019. – Zugriffsdatum: 20-05-28

[18] Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models / International Organization for Standardization. Geneva, CH, März 2011. – Standard

[19] *Joy-IT® Ultrasonic Distance Sensor*. https://cdn-reichelt.de/documents/datenblatt/A300/SEN-US01-DATASHEET.pdfl. 2020. – Zugriffsdatum: 2020-06-02

[20] Kripke, Saul A.: Naming and necessity. In: *Semantics of natural language*. Springer, 1972, S. 253–355

[21] Krumke, Sven O. ; Noltemeier, Hartmut: *Graphentheoretische Konzepte und Algorithmen*. Springer-Verlag, 2009

[22] Levene, M: *A guided tour of relational databases and beyond*. London New York : Springer, 1999. – ISBN 978-1-85233-008-8

[23] *MayBMS - A Probabilistic Database Management System.* http://maybms.sourceforge.net. – Zugriffsdatum: 2020-03-12

[24] *MayBMS: A Probabilistic Database System - User Manual.* http://maybms.sourceforge.net/manual/index.html. – Zugriffsdatum: 2020-03-12

[25] *NoSQL - Your Ultimate Guide to the Non-Relational Universe!* https://nosql-database.org. 2019. – Zugriffsdatum: 2019-12-29

[26] Obe, Regina: *PostgreSQL : up and running : a practical guide to the advanced open source database.* Sebastopol, CA : O'Reilly Media, 2017. – ISBN 978-1-491-96341-8

[27] *Was ist eine Datenbank?* https://www.oracle.com/de/database/what-is-database.html. 2019. – Zugriffsdatum: 2019-12-08

[28] *PostgreSQL 12.3 Documentation.* https://www.postgresql.org/docs/current/index.html. 2019. – Zugriffsdatum: 2020-06-04

[29] *Apt.* https://wiki.postgresql.org/wiki/Apt. 2020. – Zugriffsdatum: 2020-06-04

[30] Prabhakar, Sunil: ORION: Managing Uncertain (Sensor) Data.

[31] *Python 2.7.12 Documentation.* https://docs.python.org/release/2.7.12/. 2016. – Zugriffsdatum: 2020-06-04

[32] Reiter, Raymond: *On Closed World Data Bases.* S. 55–76. In: Gallaire, Hervé (Hrsg.) ; Minker, Jack (Hrsg.): *Logic and Data Bases.* Boston, MA : Springer US, 1978. – URL https://doi.org/10.1007/978-1-4684-3384-5_3. – ISBN 978-1-4684-3384-5

[33] Sarma, Anish D. ; Benjelloun, Omar ; Halevy, Alon ; Widom, Jennifer: Working models for uncertain data. In: *22nd International Conference on Data Engineering (ICDE'06)* IEEE (Veranst.), 2006, S. 7–7

[34] Suciu, Dan: *Probabilistic databases.* San Rafael, Calif. (1537 Fourth Street, San Rafael, CA 94901 USA : Morgan & Claypool, 2011. – ISBN 9781608456802

[35] *TPC-H.* http://www.tpc.org/tpch/default.asp. 2019. – Zugriffsdatum: 20-05-28

[36] *Stanford Trio Project - Source Code.* http://infolab.stanford.edu/trio/code/index.html. 2019. – Zugriffsdatum: 2020-02-10

[37] *TriQL – The Trio Query Language*. https://cs.stanford.edu/people/widom/triql.html. 2008. – Zugriffsdatum: 20-05-28

[38] *VMWare Workstation Player 15*. https://www.vmware.com/de/products/workstation-player/workstation-player-evaluation.html. 2020. – Zugriffsdatum: 2020-06-02

[39] Walmsley, Priscilla: *XQuery*. Farnham, Calif : O'Reilly, 2007. – ISBN 0-596-00634-9

[40] Witt, B.C.: *Datenschutz kompakt und verständlich: Eine praxisorientierte Einführung*. Vieweg+Teubner Verlag, 2010 (Edition kes). – URL https://books.google.de/books?id=_NoYxjz6lF8C. – ISBN 9783834896537

[41] Zimanyi, Esteban: Incomplete and uncertain information in relational databases. In: *Université Libre de Bruxelles, Brussels, Belgium* (1992)

Abkürzungsverzeichnis

ASJP - Aggregation, Selection, Join, Projection

CWA - Closed World Assumption

DB - Datenbank

DBMS - Database Management System

LDB - Lineage Database

OWA - Open World Assumption

SJP - Selection, Join, Projection

SQL - Structured Query Language

ULDB - Uncertainty Lineage Database

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Experimentelle Evaluation von Frameworks zur Speicherung und Verarbeitung unsicherer Daten

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	[redacted]
Ort	Datum	Unterschrift im Original