Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

# Nutzung von Provenance-Daten zur Analyse personenbezogener Daten gemäß der DSGVO-Richtlinien

Alex Frank

| | |
|---|---|
| **Studiengang:** | Softwaretechnik |
| **Prüfer/in:** | PD Dr. rer. nat. habil. Holger Schwarz |
| **Betreuer/in:** | Dipl.-Inf. Michael Behringer,<br>Ralf Diestelkämper, M. Sc. |
| **Beginn am:** | 20. November 2019 |
| **Beendet am:** | 15. Juli 2020 |

## Kurzfassung

Durch die voranschreitende Digitalisierung der Gesellschaft sind Unternehmen mehr denn je zuvor in der Lage personenbezogene Daten im Internet zu erfassen, ohne dass Verbraucher dies mitbekommen. Es reicht schon das Besuchen einer Webseite mit angemeldeten Kundenkonto aus, um seine Daten preiszugeben. Personenbezogene Daten sind dabei mannigfaltig und reichen von Name und Anschrift bis zur IP-Adresse des Endgerätes. Diese Daten können für eine Vielzahl von personenbezogenen Analysen verwendet werden, wie etwa das Kaufverhalten eines Verbrauchers. Die Ergebnisse solcher Anfragen können für Zwecke, wie Werbung und Neukundenakquise genutzt werden.

Zum Schutz der persönlichen Daten der Verbraucher ist seit dem 25. Mai 2018 die Datenschutz-Grundverordnung in Kraft getreten. Diese Verordnung regelt den Datenschutz der EU-Bürger und gibt ihnen eine Vielzahl an Rechte im Bezug auf ihre personenbezogenen Daten, wie etwa das Recht auf Vergessenwerden. EU-Bürger können jederzeit von diesen Rechten Gebrauch machen. Dies führt zu einigen Herausforderungen bei Unternehmen, die nun zu jederzeit nachweisen müssen, für welchen Zweck die personenbezogenen Daten verarbeitet werden und sie müssen sicherstellen, dass Verbraucher in der Lage sind jederzeit die Einwilligung für die Verarbeitung ihrer personenbezogenen Daten zu entziehen. Bei Nichteinhaltung der Einwilligung drohen Strafen in der Höhe von 4% des Jahresumsatzes.

Personenbezogene Analysen und ihre Ergebnisse können dabei unterschiedlich komplex sein. Dies hängt auch maßgeblich von der Größe der Daten ab, die in die Terabytes gehen können. Diese Daten lassen sich unter dem Begriff *Big Data* zusammenfassen. Um die Berechnung dieser Big Data effizient zu ermöglichen, wurden verteilte Systeme für datengetriebene Berechnungen entwickelt, die in der Lage sind diese Daten in einem Rechnerverbund effizient zu berechnen. Bei einem Entzug einer Einwilligung müssen Ergebnisse von personenbezogenen Analysen neu berechnet werden, da sonst ein Rechtsverstoß vorliegt.

In dieser Arbeit wird ein Verfahren entwickelt, um Neuberechnungen unter Einhaltung der Datenschutz-Grundverordnung effizienter durchzuführen. Dazu wird zunächst ein Datenmodell vorgestellt, welches es ermöglicht personenbezogenen Analysen Datenschutz-Grundverordnung konform zu berechnen. Dieses Datenmodell erlaubt die Zuweisung von Einwilligungen für die Verarbeitung der Daten. Zusätzlich dazu wird eine Indexstruktur vorgestellt, die die effiziente Identifikation von geänderten Daten ermöglicht, so dass nur ein kleiner Anteil der Daten neu berechnet werden muss. Dadurch sollen Anfragen potentiell effizienter durchgeführt werden. Die Indexstruktur und das Datenmodell werden dabei in Apache Spark implementiert und evaluiert. Die Evaluation hat gezeigt, dass die Neuberechnung mittels der Indexstruktur für manche Anfragen schneller sein kann.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Verzeichnis der Algorithmen

# Abkürzungsverzeichnis

**API** Application Programming Interface. 42

**DBLP** Digital Bibliography and Library Project. 22

**DS-GVO** Datenschutz-Grundverordnung. 16

**DSL** Domain-specific language. 27

**EC2** Amazon Elastic Compute Cloud. 41

**ER-Diagramm** Entity-Relationship-Diagramm. 33

**GMRW** Generalized Map And Reduce Workflow. 40

**HDFS** Hadoop Distributed File System. 41

**HTTP** Hypertext Transfer Protocol. 49

**JVM** Java Virtual Machine. 70

**RAMP** Reduce And Map Provenance. 40

**RDD** Resilient Distributed Dataset. 26

**RIDs** Record IDs. 48

**SAMbA** Spark provenAnce MAnagement on RDDs. 46

**SQL** Structured Query Language. 16

**SSD** Solid State Drive. 44

**TPC** Transaction Processing Performance Council. 17

**TPM** Tree-Pattern-Matching. 44

**VSDB** Verteilte Systeme für datengetriebene Berechnungen. 15

**W3C** World Wide Web Consortium. 37

# 1 Einleitung

## 1.1 Problemstellung und Zieldefinition

Das Erfassen von personenbezogenen Daten ist im digitalen Zeitalter keine Seltenheit mehr. Vor allem im Internet ist die Preisgabe der persönlichen Daten schnell und intransparent für den Verbraucher geschehen. Dabei reicht meistens schon das Anschauen und Anklicken von Produkten, während man z. B. auf einer Webseite eingeloggt ist, aus, um die Vorlieben und Interessen zu ermitteln und sie einem Verbraucher zuzuordnen.

Für viele Unternehmen sind diese personenbezogenen Daten ein wichtiger Bestandteil des Geschäftsmodells. Daten wie z. B. Name, Adresse, E-Mail, IP-Adresse, Primärschlüssel in einer Datenbank und generell alle Daten, die direkt oder indirekt einer Person zugeordnet werden können, gelten als personenbezogene Daten [Uni16]. Diese Daten können für verschiedene analytische Abfragen verwendet werden, um daraus Profit für das Unternehmen zu generieren. Sie reichen dabei von geschäftsmäßiger Feststellung der Bonität bis zur Bindung und Akquisition von Kunden und Marktforschung. Auch ist das Einsparen von Marketingkosten durch solche Analysen möglich. Werbung muss nicht mehr auf verschiedenen Plattformen gestreut werden, sondern wird an die Interessen des Kunden angepasst [Uns10, S. 3 f.]. Ein weiteres Beispiel solch einer analytischen Abfrage auf personenbezogenen Daten ist z. B. die Interessen eines Kunden an Produktkategorien anhand seiner Bestellungen. Dadurch kann diesem Kunden gezielt Werbung angezeigt oder je nach Umsatz ein besonderer Kundenstatus zugesprochen werden.

Personenbezogene Daten werden dabei auf verschiedene Wege gesammelt. Neben den elektronischen Handelsplattformen wie z. B. Amazon[1] und sozialen Medien wie z. B. Facebook[2], sammeln elektronische Assistenten wie z. B. Siri[3] oder Alexa[4], Daten über ihre Nutzer [BWD+18]. Des weiteren existieren eine Vielzahl weiterer elektronischer Geräte wie z. B. Smart-Watches, Smartphones, Sensoren in Autos und Küchengeräte, die in der Lage sind Daten zu produzieren. Durch diese Vielzahl an Geräten und Sensoren wächst das auszuwertende Datenvolumen täglich massiv an, die Daten sind dabei keineswegs homogen, sondern unterscheiden sich in ihrem Inhalt und Struktur.

Um die Verarbeitung dieser Daten und die Durchführung von analytischen Abfragen effizient zu ermöglichen, wurden neue Konzepte und Techniken entwickelt. Verteilte Systeme für datengetriebene Berechnungen (VSDB) wie z. B. Apache Spark[5], Apache Hadoop[6] und Apache Flink[7] sind in der

[1] https://www.amazon.de/
[2] https://de-de.facebook.com/
[3] https://www.apple.com/de/siri/
[4] https://www.amazon.de/b?ie=UTF8&node=12775495031
[5] https://spark.apache.org/
[6] https://hadoop.apache.org/
[7] https://flink.apache.org/

Lage Petabyte große und unstrukturierte Daten zu verarbeiten. Sie bieten dabei eine Vielzahl an Möglichkeiten an, um Daten mittels der Abfragesprache Structured Query Language (SQL) oder Algorithmen aus dem Bereich des maschinellen Lernens zu analysieren und auszuwerten. Dazu werden die Analysen in einem Rechnerverbund und im Hautspeicher durchgeführt. Die Ergebnisse der Analysen werden dabei im Hauptspeicher zwischengespeichert und ermöglichen somit die schnelle Abfrage.

Personenbezogene Daten spielen eine zentrale Rolle in der Analyse von Verbrauchern. Die Europäische Union hat dies erkannt und den bestehenden Datenschutz der EU-Bürger aktualisiert. Seit dem 25. Mai 2018 ist die neue Datenschutz-Grundverordnung in Kraft getreten und ermöglicht Verbrauchern mehr Kontrolle und Transparenz über die Verarbeitung ihrer personenbezogenen Daten [BMWoD; Uni16]. Sie regelt den Datenschutz der EU-Bürger in den 27 EU-Ländern neu und löst, die bis dahin in Kraft gewesene, Europäische Datenschutzrichtlinie von 1995 ab. Durch Rechte wie z. B. das Recht auf Vergessenwerden, das Recht auf Korrektur und das Auskunftsrecht kann der Verbraucher die Verwendung seiner Daten kontrollieren, einschränken oder freigeben, sowie sich über den Verwendungszweck informieren. Der vollständige Entzug der Einwilligung für nicht essentielle Verarbeitungen ist dem Verbraucher ebenso möglich. Essentielle Verarbeitungen sind alle Verarbeitungen für die keine gesonderte Einwilligung benötigt wird, da sie für einen korrekten Geschäftsablauf notwendig sind. Dazu zählt z. B. die Verarbeitung der Lieferadresse eines Kunden, um die Auslieferung der Ware an die korrekte Lieferadresse zu veranlassen. Dazu wird keine Einwilligung benötigt, sofern die Lieferadresse nur zum Zwecke der Lieferung verwendet wird. Das Unternehmen muss dafür Sorge tragen, dass die Daten nach Abschluss der Lieferung entfernt, nicht zweckentfremdet und ohne Einwilligung weiterverarbeitet werden. Analytische Abfragen, für die keine Einwilligung seitens der Verbraucher besteht, werden nach Datenschutz-Grundverordnung (DS-GVO) als Rechtsverstoß mit bis zu 4% des Jahresumsatzes oder bis zu 20 Millionen Euro geahndet [Uni16].

Damit Unternehmen rechtmäßig handeln, haben sie dafür Sorge zu tragen, dass analytische Anfragen nur die personenbezogene Daten berücksichtigen, die für den spezifizierten Zweck vom Verbraucher freigegeben wurden. Bei einem Einwilligungsentzug müssen die Ergebnisse dementsprechend aktualisiert werden. Dabei wird in der Regel die analytische Abfrage erneut auf den Eingabedaten durchgeführt, aber ohne die Daten für die die Einwilligung entzogen wurde. Dies führt zu einer komplette Neuberechnung auf Daten bei denen sich nur ein kleiner Teil geändert hat. Eine Neuberechnung bzw. eine Aktualisierung des Ergebnisses anhand der Daten, die implizit von einer Änderungen bzw. einem Einwilligungsentzug betroffen sind kann effizienter durchgeführt werden. Dazu müssen nur die Teile der Daten ausgewählt werden, die von einer Änderung betroffen sind. Durch diese Art der Neuberechnung auf partiellen Daten, können Analysen effizienter durchgeführt werden.

Das Ziel dieser Arbeit ist die effiziente Neuberechnung auf partiellen Daten unter Beachtung der gegebenen Einwilligung zur weiteren Verarbeitung bzw. analytischen Abfragen. Dazu wird im ersten Schritt ein Konzept entworfen, welches es ermöglicht, die Daten mit Einwilligungen zu annotieren. Diese Einwilligungen werden bei analytischen Abfragen beachtet und entsprechend nur die Daten verarbeitet, bei denen eine Einwilligung gegeben wurde. Im zweiten Schritt wird ein Prototyp auf dem VSDB-System Apache Spark entwickelt, der eine Neuberechnung auf partiellen Daten ermöglicht. Dazu werden Provenance-Informationen verwendet, die die Assoziationen zwischen der Ausgabe von analytischen der Abfragen und den Eingabedaten ermöglichen. Der Abschluss dieser Arbeit bildet die Evaluation der beiden genannten Schritte.

## 1.2 Begleitendes Beispiel

Anhand eines begleitendes Beispiels wird die Funktionsweise des Prototyps Schritt für Schritt in dem nachfolgenden Kapitel erklärt. Das Beispiel orientiert sich dabei an dem TPC-H Benchmark[8] mit einigen Änderungen am Schema, um das Beispiel einfach zu halten.

### 1.2.1 TPC-H Benchmark

Das TPC-H Benchmark ist eine Entwicklung vom Transaction Processing Performance Council (TPC)[9] und beinhaltet insgesamt acht verschiedene Tabellen sowie 22 SQL-Anfragen. Das Benchmark soll bei der Auswahl eines Entscheidungsunterstützungssystem unterstützen. Solche Systeme ermitteln und bereiten Informationen auf und unterstützen somit ihrerseits Entscheidungsträger bei ihren strategischen Zielen [Cou18]. Aus diesem Grund befinden sich die Tabellen in einer wirtschaftsorientierten Domäne, wie dem Onlinehandel, der über eine Vielzahl an Kunden, Produkt- und Bestelldaten verfügt, aus denen entscheidende Informationen extrahiert werden können. Die vollständigen Beziehungen und alle Attribute der acht Tabellen des TPC-H Benchmarks sind in der Abbildung 1.1 zu sehen. Die Abbildung stammt aus dem TPC-H Benchmark [Cou18].

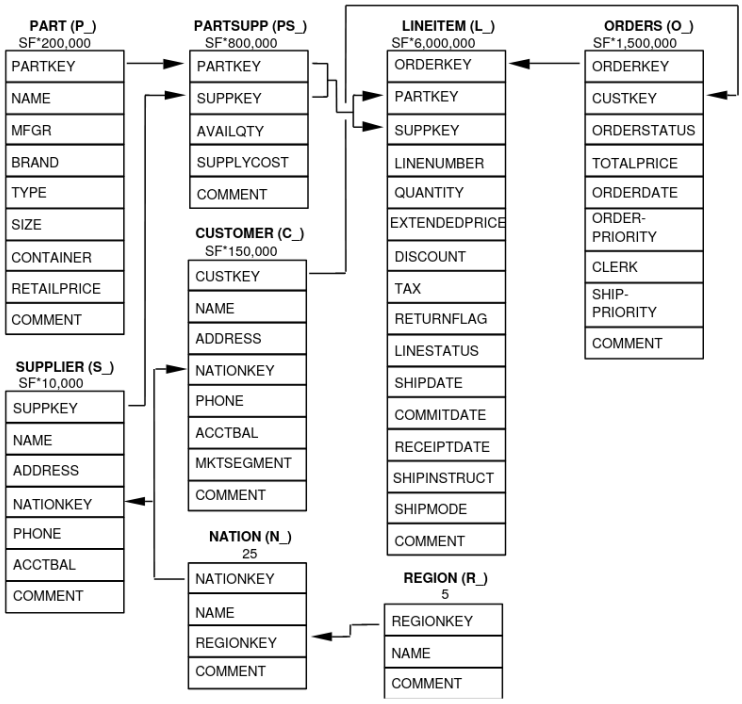



**Abbildung 1.1:** Schemata des TPC-H Benchmarks

[8] http://www.tpc.org/tpch/default5.asp
[9] http://www.tpc.org/

Die einzelnen Tabellen und ihre Attribute sind in der folgenden Auflistung genauer beschrieben.

**Customer** Tabelle mit Informationen über Kunden wie Name, Adresse und Telefonnummer

**Supplier** Tabelle mit Informationen über Lieferanten wie Name, Adresse und Telefonnummer

**Region** Tabelle mit Informationen über Regionen wie Europa, Amerika und Asien

**Nation** Tabelle mit Informationen über verschiedene Ländern wie Deutschland, Vereinigte Staaten von Amerika und China. Jedes Land ist immer einer Region in der **Region**-Tabelle zugeordnet

**Orders** Tabelle mit Bestellungen. Jeder Kunde kann mehrere Bestellungen durchführen. Dabei besteht jede Bestellung aus mehreren **Lineitem** und den Gesamtpreis.

**Lineitem** Tabelle mit Posten. Jeder Posten stellt dabei ein Produkt und Lieferinformationen in einer Bestellung dar. Ein Posten besteht aus einer Vielzahl Attributen wie Lieferstatus, Mehrwertsteuer und Rabatt.

**Part** Tabelle mit Produktinformationen. Diese Tabelle enthält verschiedene Attribute, die ein Produkt betreffen wie z. B. Name, Marke, Verkaufspreis

**Partsupp** Tabelle mit Produkten, die ein Lieferant liefert. Diese Tabelle ist eine *n:m* Zwischentabelle zwischen **Supplier** und **Part**.

### 1.2.2 Aufbau des Beispiels

Das Beispiel dieser Arbeit beschränkt sich auf drei Tabellen des Benchmarks, nämlich auf *Customer*, *Orders* und *Lineitem*, da diese personenbezogene Daten beinhalten und entsprechend personenbezogene Analysen ermöglichen. Aus Gründen der Einfachheit werden nicht alle Attribute der Tabellen verwenden, des weiteren werden einige Attribute hinzugefügt, um das Ergebnis der nachfolgende Analyse besser darzustellen.

Die *Customer*-Tabelle beinhaltet personenbezogene Daten wie Name, Adresse und Telefonnummer. Ein Kunde kann nun, wie es in einem Onlineshop üblich ist, mehrere Bestellungen tätigen. Diese Bestellungen werden in der *Orders*-Tabelle erfasst. Diese Tabelle beinhaltet außer dem Bestelldatum und dem eignen Primärschlüssel, noch einen Fremdschlüssel der immer auf einen Kunden verweist. Eine Bestellung hingegen kann aus mehreren Posten bestehen. Alle Posten werden in die Tabelle *Lineitem* eingetragen. Ein Posten repräsentiert dabei ein Produkt in einer Bestellung und besteht aus einer Bezeichnung, einer Produktkategorie und einem Preis, sowie einem Primärschlüssel und einem Fremdschlüssel, der einer Bestellung zugeordnet ist.

Für das Beispiel werden insgesamt drei Kunden *Bob*, *Alice* und *David* angelegt. Alle drei Kunden haben bereits Bestellungen getätigt. Bob hat am 2. Februar 2020 drei Bestellungen, Alice am 2. März 2020 und David am 12. Dezember 2019 jeweils eine Bestellung durchgeführt. Bob hat vier Artikel bestellt: ein Haushaltsartikel mit einem Automobilartikel, ein Modeartikel und noch ein separater Haushaltsartikel. Alice hat einen Elektronikartikel und David einen Modeartikel bestellt.

**Tabelle 1.1:** Instanz einer Customer-Tabelle

| c_custkey | c_name | c_address | c_phone |
|---|---|---|---|
| 1 | Bob | 9 Hermina Crossing | 555-1234 |
| 2 | Alice | 6 Katie Road | 555-1337 |
| 3 | David | 47 Anzinger Road | 555-0042 |

**Tabelle 1.2:** Instanz einer Orders-Tabelle

| o_orderkey | o_custkey | o_orderdate |
|---|---|---|
| 1 | 1 | 02-02-2020 |
| 2 | 1 | 02-02-2020 |
| 3 | 1 | 02-02-2020 |
| 4 | 2 | 02-03-2020 |
| 5 | 3 | 12-12-2019 |

### 1.2.3 Beispiel einer personenbezogene Analyse

Auf diesen Daten können nun verschiedene personenbezogene Analysen durchgeführt werden. Für dieses Beispiel haben wir eine personenbezogene Analyse ausgewählt, die das Kaufverhalten der Kunden beschreibt. Das Kaufverhalten wird dabei an der Interesse an einer Kategorie gemessen. Das Interesse wird hierbei gleichgesetzt mit dem ausgegebenen Geldbetrag pro Kategorie. Die daraus resultierenden Daten können z. B. für die gezielte Werbung von Artikeln in der entsprechenden Kategorie für den Kunden verwendet werden. Auch für die gezielte Kundenbindung kann diese Analyse hilfreich sein. Ein Kunde, bei dem ein erhöhtes Kaufverhalten für eine Kategorie festgestellt wurde, kann z. B. durch Rabattgutscheine oder einem exklusiven Kundenstatus zu weiteren Bestellungen animiert werden.

Abbildung 1.2 zeigt die analytische Abfrage als Operatorenbaum. Die Knoten stellen die Ergebnisse bzw. Teilergebnisse der relationalen Operationen dar, während die Kanten die einzelnen relationalen Operationen darstellen. Der Baum wird dabei von unten nach oben gelesen. Der oberste Knoten ist das Endergebnis. Die Analyse summiert alle Einkäufe eines Kunden auf und gruppiert sie nach der Produktkategorie. Dabei werden nur Bestellungen in Betracht gezogen, die im Jahre 2020 getätigt wurden.

Dazu werden im ersten Schritt alle Bestellungen ausgewählt, die im Jahr 2020 getätigt wurden. Anschließend werden die gefilterten Bestellungen ihrem jeweiligen Kunden mittels dem *custkey*-Schlüssel zugewiesen. Das Teilergebnis wiederum, wird mit der Posten-Tabelle verbunden. Dies geschieht durch den *orderkey*-Schlüssel. Nachdem alle notwendigen Tabellen miteinander verbunden wurden, beginnt die eigentliche Analyse. Dazu werden alle Posten anhand ihrer Kategorie und den Kundennamen gruppiert. Diese Gruppierung ermöglicht das anschließende Aufsummieren der einzelnen Preise pro Kategorie und Kunde. Die einzelnen Attribute des Kunden wie *Name*, *Adresse* und *Telefonnummer* werden ebenso in der Analyse ausgewählt. Dadurch kann eine Kontaktaufnahme mit dem Kunden erfolgen, um dem Kunden z. B. einen Gutschein bzw. Rabatte anzubieten oder gezielte Werbung anhand der Kategorie zu versenden.

**Tabelle 1.3:** Instanz einer Lineitem-Tabelle

| l_lineitemkey | l_orderkey | l_name | l_category | l_price |
|---|---|---|---|---|
| 1 | 1 | Waschmaschine | Haushalt | 500,00 |
| 2 | 1 | Scheibenwischer | Automobil | 50,00 |
| 3 | 2 | Trockner | Haushalt | 350,00 |
| 4 | 3 | Jeans | Mode | 30,00 |
| 5 | 4 | iPhone | Elektronik | 700,00 |
| 6 | 5 | Lederjacke | Mode | 120,00 |

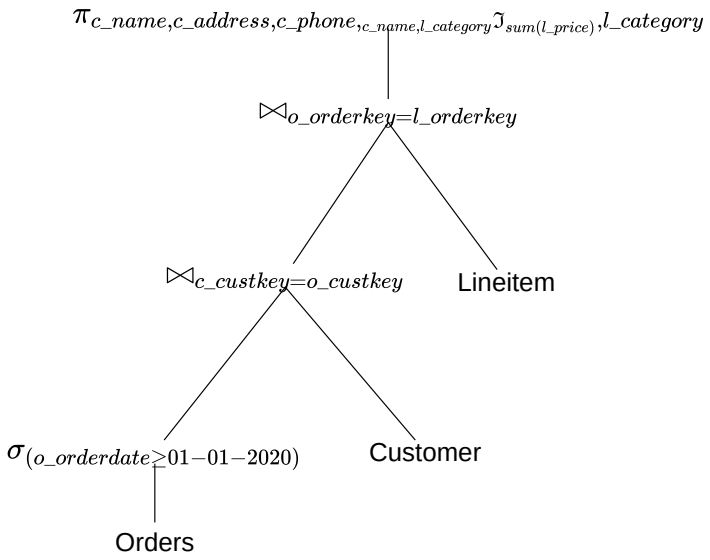



**Abbildung 1.2:** Analytische Abfrage als Operatorenbaum

Das daraus resultierende Ergebnis der Analyse ist in der Tabelle 1.4 dargestellt. In der Abfrage sind nur Alice und Bob vorhanden, da sie ihre Bestellungen im Jahr 2020 ausgeführt haben. David hat zuletzt im Jahr 2019 bestellt und ist somit nicht im Ergebnis vorhanden.

**Tabelle 1.4:** Instanz der Analyse

| c_name | c_address | c_phone | sum(l_price) | l_category |
|---|---|---|---|---|
| Bob | 9 Hermina Crossing | 555-1234 | 850,00 | Haushalt |
| Bob | 9 Hermina Crossing | 555-1234 | 50,00 | Automobil |
| Bob | 9 Hermina Crossing | 555-1234 | 30,00 | Mode |
| Alice | 6 Katie Road | 555-1337 | 700,00 | Elektronik |

### 1.2.4 Änderung der Einwilligung

Jeder der Kunden hat dank der DS-GVO das Recht, solche Analysen ihrer personenbezogenen Daten zu verhindern bzw. einzuschränken. In diesem Beispiel wird davon ausgegangen, dass der Entzug der Einwilligung feingranular auf der Ebene der Bestellungen erfolgen kann, d.h. ein Kunde hätte die Möglichkeit Einwilligungen für jede seiner getätigten Bestellung entweder zu geben oder zu verweigern. In diesem Beispiel wird nur der Einwilligungsentzug in Betracht genommen.

Um das zu demonstrieren, entzieht Bob seine Einwilligung bezüglich der personenbezogenen Verarbeitung für die Bestellung mit der Bestellnummer $o\_orderkey = 1$. Dies hat zur Folge das Analyse 1.2, die die Bestellung $o\_orderkey = 1$ verwendet, neu berechnet werden muss. Das Ergebnis der erneut durchgeführten Analyse ist in Tabelle 5.1 dargestellt. Der Entzug der Einwilligung hat Auswirkung auf zwei Bestellungen. Die Bestellung mit $o\_orderkey = 1$ darf nicht mehr für Analysen verwendet werden, dies wirkt sich jedoch auf die komplette Ergebnissumme für die Haushaltskategorie der Analyse aus, da in die Summe, die Bestellung $o\_orderkey = 2$ einfließt. Diese ist vom Entzug nicht betroffen und kann deshalb weiterhin verwendet werden. Konkret bedeutet das von der Summe für die Haushaltskategorie 500,00 Euro abgezogen werden müssen und die Zeile mit der Kategorie *Automobil* für Bob nicht mehr im Ergebnis vorkommen darf.

**Tabelle 1.5:** Bereinigte Instanz der Analyse

| c_name | c_address | c_phone | sum(l_price) | l_category |
|---|---|---|---|---|
| Bob | 9 Hermina Crossing | 555-1234 | 350,00 | Haushalt |
| Bob | 9 Hermina Crossing | 555-1234 | 30,00 | Mode |
| Alice | 6 Katie Road | 555-1337 | 700,00 | Elektronik |

In diesem Beispiel ist das Neuberechnen ohne viel rechnerischen Aufwand möglich, da nur drei Kunden und fünf Bestellungen in den Tabellen existieren. Unternehmen wie z. B. Alibaba[10] generieren jedoch innerhalb einer Sekunde 500.000 Bestellungen und verfügen über weitaus mehr Kunden [Hua19]. Eine erneute Durchführung der personenbezogenen Abfragen bei Änderungen der Einwilligungen führt zu einem massiven rechnerischen Aufwand bei potentiell wenigen Änderungen. Diese Neuberechnung muss trotz der rechnerischen Schwierigkeiten durchgeführt werden. Durch die, in dieser Arbeit vorgestellten, Konzepte, soll solch eine erneute Berechnung effizienter und konform zur DS-GVO gestaltet werden.

## 1.3 Struktur der Arbeit und Forschungsbeiträge

Diese Arbeit unterteilt sich in mehrere Kapitel. Der folgende Abschnitt beschreibt die Strukturierung und gibt einen kurzen Einblick in die jeweilige Thematik der Kapitel und den jeweiligen Beitrag zur Forschung. Forschungsbeiträge werden dabei mittels **B** gekennzeichnet.

[10] https://german.alibaba.com/

**Kapitel 2 - Grundlagen** beschreibt die fundamentalen Grundkenntnisse und Grundbegriffe, die notwendig zum Verständnis dieser Arbeit sind. Es wird in die Themengebiete Datenschutz-Grundverordnung, Big Data, Verteilte Systeme für datengetriebene Berechnungen und Provenance eingeführt. Die verwendeten mathematischen Notationen werden in diesem Kapitel definiert.

**(B.1) Kapitel 3 - Verwandte Arbeiten** gibt einen Überblick über den derzeitigen Stand der Forschung in Bezug auf Provenance-Modelle, Provenance-Lösungen, Indizes für Provenance-Daten in Provenance, Partielle Neuberechnung und das Aktualisieren von Views. Der Abschnitt Provenance-Lösungen vergleicht verschiedene Lösungen und wählt anhand ausgesuchter Kriterien eine Lösung für die Implementierung aus.

**(B.2) Kapitel 4 - DS-GVO konformes Datenmodell und Ausführung** beschreibt ein Konzept zur Abbildung von Einwilligungen auf Daten, so dass personenbezogene Analysen konform zur DS-GVO durchgeführt werden können.

**(B.3) Kapitel 5 - DS-GVO konforme Analyse mittels geeigneter Indexstruktur** beschreibt ein Konzept einer Indexstruktur für eine effiziente Identifikation von Eingabedaten bei Änderung von Einwilligungen und die damit verbundene effizientere Berechnung auf Teildaten.

**(B.4) Kapitel 6 - Implementierung** beschreibt die Implementierung des Datenmodells und der Indexstruktur.

**(B.5) Kapitel 7 - Evaluation** beschreibt die Evaluation des Datenmodells und der Indexstruktur auf den Datensätzen TPC-H und Digital Bibliography and Library Project (DBLP).

**Kapitel 8 - Zusammenfassung und Ausblick** fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick in die Zukunft.

# 2 Grundlagen

Diese Kapitel legt den Grundstein dieser Arbeit und gibt einen Überblick über die Themen DS-GVO, Big Data, VSDB, Provenance, sowie die Definition der verwendeten Notationen.

## 2.1 Datenschutz-Grundverordnung

Die DS-GVO ist eine europäische Richtlinie für die Einhaltung des Datenschutzes. Sie ist seit dem 25. Mai 2018 in Kraft und erweitert die EU-Datenschutzrichtlinie aus dem Jahre 1995. Diese erneuerte Richtlinie soll Lösungen für die Datenschutzfragen einer immer weiter voranschreitenden Informationsgesellschaft liefern, den EU-Bürgern mehr Kontrolle und Transparenz über die Verarbeitung seiner Daten an die Hand geben und den europäischen Datenschutz vereinheitlichen. Diese Vereinheitlichung ist vor allem getrieben durch die einzelnen unterschiedlichen Datenschutzrichtlinien in den verschiedenen EU-Ländern [BMWoD; Uni16].

Diese neue Richtlinie ist verbindlich für alle europäischen Mitgliedsstaaten und alle Unternehmen, die mit EU-Bürgern Geschäfte tätigen wollen. Dadurch ist die DS-GVO grenzübergreifend gültig und verpflichtet somit alle Unternehmen zur Einhaltung der DS-GVO.

Den EU-Bürgern werden durch die DS-GVO mehrere Rechte eingeräumt, so dass sie über mehr Kontrolle und über mehr Transparenz bezüglich ihrer personenbezogene Daten verfügen. Personenbezogene Daten sind Daten, die einer natürlichen Person zugeordnet werden können. Dabei können die Daten direkt einer Person zugeordnet werden, wie etwa der Name und die Anschrift aber auch indirekt, wie etwa IP-Adressen oder Rechnungsnummern. Die einzelnen Rechte sind im folgenden Abschnitt aufgelistet [Uni16].

**Recht auf Auskunft über persönliche Daten** Die betroffene Person hat das Recht zu jeder Zeit zu erfahren, ob ihre Daten verarbeitet werden und falls ja, zu welchem Zweck

**Recht auf Berichtigung** Falsche oder unvollständige Angaben über eine betroffene Person müssen, bei Meldung, vom Unternehmen berichtigt werden

**Recht auf Löschen und Vergessen** Das Unternehmen ist verpflichtet personenbezogene Daten nach Verarbeitung oder nach Aufforderung der betroffenen Person zu löschen

**Recht auf Einschränkung der Verarbeitung** Die betroffene Person kann das Unternehmen auffordern, die Verarbeitung seiner Daten einzuschränken

**Recht auf Datenübertragbarkeit** Personenbezogene Daten können von der betroffenen Person angefordert und an einen anderen Verantwortlichen übertragen werden

Diese Rechte können eingeschränkt werden, wenn z. B. die nationale Sicherheit gefährdet ist oder Daten für die Strafverfolgung benötigt werden. Diese Beschränkungen müssen dabei aber stets die Grundrechte und Grundfreiheiten der Bürger achten [Uni16].

Unternehmen stehen vor einigen Herausforderungen, um die DS-GVO umzusetzen [BMJoD; BMWoD; Uni16]. Diese werden in dem folgenden Abschnitt genauer aufgelistet.

**Marktortprinzip** Die DS-GVO ist für Unternehmen, die mit EU-Bürgern handeln möchten verbindlich einzuhalten, unabhängig vom Standort des Unternehmen.

**Zweckbindung** Erhobene Daten dürfen nur für den Zweck verwendet, für den sie erhoben wurden.

**Datenminimierung** Es dürfen nur die Daten erhoben werden, die für den jeweiligen Zweck relevant sind.

**Einwilligungen** Strikte Anforderungen an Einwilligungsformulare müssen umgesetzt werden. Für jede benötige Datenerhebung ist eine separate Einwilligung vom Nutzer erforderlich. Pauschaleinwilligungen sind nicht erlaubt.

**Informationspflicht** Bei der Erhebung von personenbezogenen Daten muss das Unternehmen verschiedene Informationen, darunter den Zweck der Verarbeitung, vor der Zustimmung der Datenverarbeitung klar und deutlich an die betroffene Person kommunizieren.

**Transparenz** Durch die Verwendung einer klaren und einfachen Sprache muss das Unternehmen sicherstellen, dass Informationen, die die Verarbeitung der Daten betreffen, dem Verbraucher zugänglich gemacht werden. Dabei müssen auch Kinder in der Lage sein, den Zweck der Erhebung nachzuvollziehen.

**Privacy By Design** Der Datenschutz ist von Beginn an in einer Anwendung gewährleistet.

**Privacy By Default** Um die Verordnung mit der Benutzbarkeit zu verbinden, müssen neue Programme oder Konten, die personenbezogene Daten erheben, immer standardmäßig auf die datenschutzfreundlichste Einstellung gesetzt sein

Um diese Rechte durchzusetzen ist das Strafmaß verhältnismäßig hoch. Bei einem Datenschutzvergehen ist eine Geldbuße von bis zu 20 Millionen Euro oder bis zu 4 Prozent des gesamten Jahresumsatz fällig [Uni16]. Große Unternehmen müssen dementsprechend sehr hohe Strafen zahlen, da sich der Betrag nach dem Umsatz und nicht nach dem tatsächlichen Gewinn richtet.

Durch die DS-GVO haben natürliche Personen in der Europäischen Union jederzeit die Möglichkeit ihre Einwilligung zur Verarbeitung ihrer personenbezogenen Daten zu ändern. Des weiteren wird durch die Zweckbindung jeweils eine separate Einwilligung pro personenbezogene Analyse benötigt. Dadurch kann der Verbraucher feingranular kontrollieren, für welchen Zweck seine Daten verwendet werden dürfen. Damit diese Daten DS-GVO konform verarbeitet werden können, müssen sie mit den jeweiligen Einwilligungen für den jeweiligen Zweck verknüpft bzw. annotiert werden. In dieser Arbeit wird ein Konzept vorgestellt, welches es ermöglicht personenbezogene Daten mit den jeweiligen Einwilligungen des Verbraucher zu versehen und sie an einen Zweck d. h. eine personenbezogene Analyse zu koppeln, so dass diese DS-GVO konform verarbeitet werden können.

## 2.2 Big Data

Durch die voranschreitende Digitalisierung produziert die Gesellschaft zunehmend mehr Daten [CMY14]. Datengrößen von mehreren hundert Petabytes sind bereits keine Seltenheit mehr. Solche Daten fallen unter den Begriff der *Big Data*. Big Data werden verschiedene Eigenschaften zugesprochen, jedoch existiert kein Konsens diesbezüglich, da sie sich je nach Anwendungsgebiet unterscheiden. Firmen wie z. B. Oracle beschreiben Big Data als zusätzliche Quelle aus unstrukturierten Daten zu den bereits vorhandenen strukturierten Daten, die für die Entscheidungsfindung verwendet werden. Microsoft hingegen charakterisiert Big Data als Daten, die ein erhöhtes Rechenpotential benötigen und aus denen mittels maschinellen Lernen und künstlicher Intelligenz Informationen gewonnen werden können. Weitere Firmen wie etwa Intel beschreiben Daten als Big Data, wenn Organisationen durchschnittlich ca. 300 Terabyte an Daten in der Woche erzeugen [WB13].

Auf Grund dieser Diskrepanz der Eigenschaften wurden Ansätze unternommen, um Big Data mittels des *V-Modells* zu beschreiben. Dieses Modell bietet jedoch ebenso keine vollständige Charakterisierung von Big Data [CMY14; EN15].

Nichtsdestotrotz werden in der nachfolgenden Auflistung vier Eigenschaften im Stile des V-Modells aufgelistet, die versuchen Big Data entsprechend zu charakterisieren. Die Auflistung beruht dabei hauptsächlich auf der Arbeit von Elmasri und Navathe [EN15]. Ähnliche Eigenschaften werden ebenso in den Arbeiten NIST [NIS15] und Chen et al. [CMY14] beschrieben. Die Eigenschaften im *V-Modell* beginnen dabei immer mit dem Buchstaben *V*.

**Volume** Daten, die unter dem Begriff Big Data fallen, sind in der Regel sehr groß. Dazu zählen Daten, die automatisch von einem System generiert werden. Diese Daten reichen dabei von Trackingdaten aus Smartphones bis hin zu Sensordaten aus einem Produktionsprozess. Aber auch Daten aus sozialen Netzwerken wie Twitter oder Facebook zählen zu Big Data, da diese Plattformen über Millionen von Mitgliedern verfügen. Sie erzeugen dabei eine Vielzahl an Nachrichten, Bild- und Videodaten und Dokumente.

**Velocity** Durch die automatische Generierung der Daten ist die Geschwindigkeit, in der diese Daten erzeugt werden, entsprechend hoch. Die Analyse ist dementsprechend schwierig, vor allem dann, wenn die Ergebnisse von Analysen möglichst aktuell sein sollen.

**Variety** Eine weitere Eigenschaft von Big Data ist die Vielfalt der Daten. Diese unterscheiden sich nicht nur in ihrem Inhalt, sondern auch hinsichtlich der Struktur. Teilstrukturierte Daten wie etwa *JSON* oder *XML* haben keine formale Struktur, wie sie etwa bei strukturierten Daten im relationalen Datenmodell vorhanden ist. Daneben gibt es auch Daten, die vollständig ohne definierte Struktur auskommen, wie etwa Dokumente, Bild-, Audio- und Videodaten.

**Veracity** Die Genauigkeit der Daten variiert auf Grund der verschiedenen Quellen im Big Data erheblich. Big Data Daten können unvollständig, zeitverzögert ankommen oder korrupt sein. Dies kann sich negativ auf die Ergebnisse von Analysen auswirken.

Für personenbezogene Abfragen sind Big Data von relevanter Bedeutung. Durch die verschiedenen Kanäle aus denen strukturierte und unstrukturierte Daten zusammenkommen, wie etwa soziale Medien, Smartphone-Tracking und Informationen aus der Bestellhistorie, kann ein detailliertes Profil über Mitglieder und Verbraucher erstellt werden. Dieses Profil kann dann für verschiedene Zwecke, wie z. B. der personalisierten Werbung verwendet werden.

## 2.3 Verteilte Systeme für datengetriebene Berechnungen

Um die Analyse von Big Data effizient zu ermöglichen, wurden verschiedene Ansätze und Systeme entwickelt. Apache Hadoop ist ein solches System, welches die parallele und verteile Analyse von Big Data ermöglicht. Dazu werden die Arbeitsschritte, mittels dem sogenannte *MapReduce*-Programmiermodell, so aufgeteilt, das sie parallel ausführbar sind. Der erste Schritt ist der *Map*-Schritt, der eine Umformung der Eingabedaten in Schlüssel-Wert-Paare ausführt. Der nächste und zugleich letzte Schritt ist der *Reduce*-Schritt, der die Schlüssel-Werte-Paare mittels des Schlüssels auf ein Ergebnis zusammenführt. Die Map- und Reduce-Schritte sind untereinander unabhängig und erlauben deshalb eine Aufteilung der Aufgaben auf verschiedene Rechnerknoten [DG04]. Durch die Verteilung der Schritte auf mehrere Knoten zeichnet sich das Modell durch eine erhöhte Fehlertoleranz aus. Systeme wie Apache Spark[1] und Apache Flink[2] bauen das Programmiermodell aus und erlauben Analysen direkt im Arbeitsspeicher. Im nächsten Abschnitt wird hauptsächlich Apache Spark näher erläutert, da dieses System in dieser Arbeit verwendet wird.

Apache Spark ist Teil der Apache Foundation[3] und somit kostenlos und als Open-Source Software verfügbar. Nutzer von Spark können ihre Analysen in den Programmiersprachen Java, Scala, Python oder R implementieren [SDC+16; ZXW+16]. Apache Spark kann für verschiedene Einsatzzwecke im Rahmen der Datenberechnung verwendet werden [ZXW+16]

Das Hauptmerkmal von Apache Spark ist die abstrakte Darstellung von Daten anhand des Resilient Distributed Dataset (RDD) [ZXW+16]. Durch RDD bietet Spark eine einheitliche und generische Schnittstelle für die parallele Verarbeitung von Daten an [SDC+16]. Verschiedene Operationen werden über Schnittstellen angeboten, darunter befinden sich SQL, maschinelles Lernen und Graphberechnungen [ZXW+16]. Apache Spark unterscheidet zwischen *Transformationen* und *Aktionen*. Eine Transformation transformiert ein RDD in ein anderes RDD anhand von Transformationsoperationen wie z. B. *filter*. Aktionen hingegen führen eine Berechnung auf einem RDD aus und liefern entsprechend ein Ergebnis anstatt eines RDD zurück. Alle Transformationen werden als direkter und azyklischer Graph abgebildet [SDC+16]. Die Ausführung des Graphen ist verzögert und findet erst dann statt, wenn eine Aktion wie z. B. *count* aufgerufen wird. Bei einem Fehler, wie etwa dem Ausfallen eines Knoten, werden die Transformationen wiederholt, die vor dem Fehler aufgetreten sind. Die Datensätze sind dadurch fehlertolerant und ermöglichen die schnelle Erholung von Fehlern. Apache Spark bietet für SQL-Operationen sogenannte *DataFrames* an. *DataFrame* ist eine Klasse in Apache Spark und repräsentiert eine Relation, die über ein Schema, Attribute mit Datentypen und Daten in Tupeln verfügt. Sie ist eine Unterklasse von RDD und erbt somit ihre Vorteile im Bezug auf Fehlertoleranz und verteilte Berechnung. Typische relationale Operationen

[1] https://spark.apache.org/
[2] https://flink.apache.org/
[3] https://www.apache.org/

ähnlichen den Operationen wie *Join*, *Projektion* und *Selektion* werden von der DataFrame-Klasse angeboten. Dies ermöglicht eine SQL-ähnliche Abfrage der Daten [ZXW+16]. Dabei können Anfragen in einer SQL ähnlichen Sprache oder mittels der DataFrame Domain-specific language (DSL) implementiert werden [ApaoDd].

Durch die abstrakte Darstellung von Daten erlaubt Spark die Verwendung von verschiedenen Speichersystem wie z. B. MySQL[4] oder Apache Cassandra[5]. Dadurch kann der Nutzer flexibel das Speichersystem auswählen, welches für seinen Anwendungsfall am besten geeignet ist [ZXW+16].

Neben der generischen Repräsentation der Daten als RDD ist die Verwaltung der Berechnung auf Clustern eine weitere Funktion von Spark. Spark verfügt über einen Cluster-Manager, um die Ressourcen auf den verschiedenen Rechnerknoten zu nutzen und zu verwalten. Dabei ist es möglich den Cluster-Manager von Spark oder Cluster-Manager von Drittanbieter zu verwenden, wie z. B. Amazon EC2[6] [ApaoDa; SDC+16]. Dadurch kann Spark z. B. die Rechenleistung einer Cloud-Computing-Umgebung verwenden. Dies funktioniert für den Nutzer komplett transparent. Neben dem Cluster-Manager besteht Spark noch aus den folgenden Komponente.

**Driver-Programm** Ein Nutzerprogramm, das die Funktionalität von Spark verwendet, um z. B. Analysen durchzuführen

**SparkContext** Teil des Driver-Programms, welches als Verbindung zwischen dem Nutzerprogramm und dem Cluster-Manager dient

**Worker** Liefert die Computerressourcen für die Berechnungen

**Executor** Befindet sich auf einem Worker und führt eine Berechnung aus

**Task** Die kleinste Arbeitseinheit, die von einem Executer ausgeführt wird.

**Job** Ein Sammlung von Berechnungsoperationen, die das Ergebnis für das Driver-Programm berechnen. Aus Jobs wird der direkte und azyklische Graph erzeugt.

Abbildung 2.1 zeigt die einzelnen Beziehungen zwischen den Komponente nach Apache Software Foundation [ApaoDa]. Das Driver-Programm verbindet sich, mittels dem SparkContext, zum Cluster-Manager und führt das Nutzerprogramm aus. Der Cluster-Manager teilt den Graphen des Nutzerprogramms in Tasks auf und verteilt sie auf die Worker, auf denen sich Executors befinden. Je nachdem, ob eine Aktion oder Transformation im Nutzerprogramm durchgeführt wird, werden Ergebnisse der Worker entweder direkt an das Driver-Programm oder an den Cluster-Manager über das Netzwerk gesendet.

Abbildung 2.2 zeigt die vollständige Architektur nach Salloum et al. [SDC+16]. Die oberste Schicht zeigt alle Schnittstellen die Spark anbietet. Dazu zählt die Berechnung mittels Spark SQL, Streaming, maschinelles Lernen und eine Schnittstelle für die Berechnung von Graphen. Die *Spark-Core*-Schicht zeigt alle Kernelemente aus denen Spark besteht, wie etwa die RDD-Abstraktion der Daten. Aus

[4] https://www.mysql.com/
[5] https://cassandra.apache.org/
[6] https://aws.amazon.com/de/ec2/

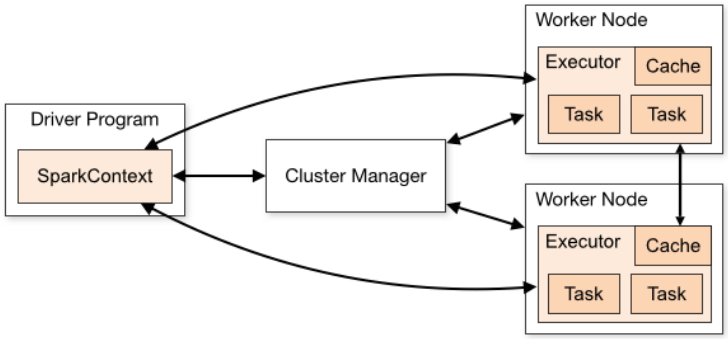



**Abbildung 2.1:** Cluster-Komponenten von Spark

diesen Elementen werden die darüber liegenden Schnittstellen implementiert. Die *Cluster-Manager*-Schicht zeigt alle unterstützten Cluster-Manager, darunter ist der Spark-interne Cluster-Manager vorhanden. Der letzte Abschnitt der Architektur zeigt alle unterstützten Speichersysteme, wie etwa die NoSQL-Datenbank Apache Cassandra.

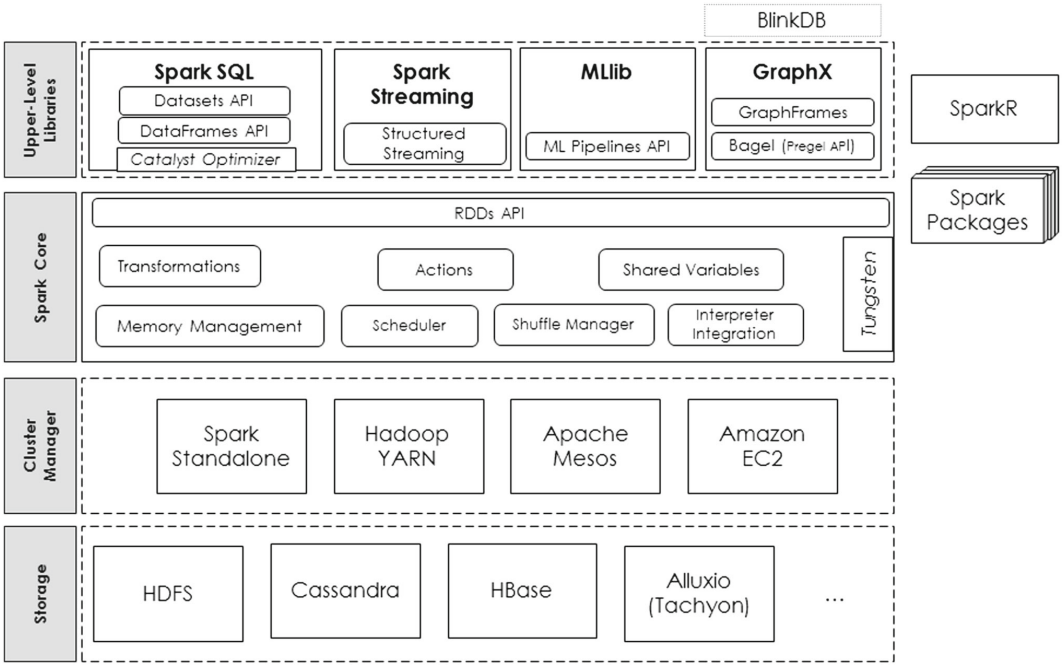



**Abbildung 2.2:** Architektur und Komponenten von Spark

Durch die Vereinheitlichung von verschiedenen Programmierparadigmen, der abstrakte Darstellung von Daten und der Verteilung der Berechnung auf verschiedene Rechnerknoten ist Spark ein beliebtes System für die Berechnung von Big Data und eigenen sich somit für die Analyse von

strukturierten und unstrukturierten Daten, wie etwa personenbezogenen Daten. Apache Spark und andere VSDB-Systeme, wie Apache Flink und Apache Hadoop, werden in großen Unternehmen wie IBM[7] und Huawei[8] bereits für Analysen aktiv eingesetzt [ZXW+16].

## 2.4 Provenance

Als Provenance werden Metadaten bezeichnet, die den Entstehungsprozess eines digitalen oder physischen Objektes beschreiben. Unter anderem ermöglicht Provenance die Assoziationen zwischen Eingabedaten und Ergebnissen. Dazu werden Eingabedaten und die daraus resultierenden Ergebnisse mit Provenance-Daten annotiert. Dadurch ist es mögliche fehlerhafte Ergebnisse bis zu ihrem Ursprung zu verfolgen und den ganzen Transformationsprozess der Daten zu beschreiben. Die folgenden Abschnitten stützen sich auf die Arbeit von Herschel et al. [HDB17].

### 2.4.1 Typen

Die Typen der Metadaten bzw. Provenance teilen sich in eine Hierarchie mit insgesamt vier Ebenen auf. Der Übergang zwischen den Ebenen ist fließend und der Informationsgehalt der Metadaten wird pro Ebene feiner d. h. der Informationsgehalt der Metadaten nimmt immer mehr zu und ermöglicht deshalb eine genauere Beschreibung des Entstehungsprozess [HDB17]. Die folgende Auflistung beschreibt alle vier Provenance-Typen aufsteigend in ihrer hierarchischen Reihenfolge.

**Provenance Metadaten** Das ist die unterste Ebene der Hierarchie. Sie umfasst alle möglichen Metadaten, die während einem Entstehungsprozess erzeugt werden und folgen dementsprechend keinem Modell und ermöglichen keine Annahmen über den Entstehungsprozess.

**Information System Provenance** Diese Ebene ist bereits restriktiver und beschreibt einen Entstehungsprozess mit Eingabe- und Ausgabedaten, sowie alle involvierten Parameter.

**Workflow Provenance** In dieser Ebene werden die Metadaten als gerichteter Graph mit Knoten und Kanten dargestellt. Knoten stellen dabei eine Funktion innerhalb eines Prozesses dar, während Kanten den Ablauf des Prozess darstellen.

**Data Provenance** Dies ist die oberste Ebene und weist den feinsten Informationsgehalt auf. Mittels den Metadaten aus dieser Ebene können einzelne Datenelemente über den ganzen Prozessablauf erfasst werden.

### 2.4.2 Data Provenance

Die Data Provenance unterscheidet sich in insgesamt fünf verschiedenen Typen, die jeweils eine Fragestellung beantworten. Die einzelnen Typen und ihre Fragen werden in den nachfolgenden Abschnitten erklärt.

[7] https://www.ibm.com
[8] https://www.huawei.com

**Why-Provenance** Mittels der Why-Provenance kann die Frage *"Welche Eingabedaten sind beteiligt am Ergebnis?"* beantwortet werden. Dazu werden die beteiligten Tupel als Zeugen bezeichnet, da sie Teil des Ergebnisses sind und somit dessen Existenz bezeugen [CCT09; HDB17].

**How-Provenance** Die How-Provenance beantwortet die Frage *"Wie wirken Eingabedaten in das Ergebnis mit ein?"*. How-Provenance unterscheidet sich in dieser Hinsicht zu Why-Provenance, da nicht nur die Existenz eines Ergebnis anhand der Eingabetupel bezeugt werden kann, sondern die Art der Abfrage, die zum Ergebnis geführt hat. Die Definition erfolgt durch Polynome bzw. dem Semiring-Framework [CCT09; GKT07; HDB17].

**Where-Provenance** Where-Provenance beantwortet die Frage *"Woher wurden die Eingabedaten für das Ergebnis kopiert?"*. Dieser Typ von Provenance beschäftigt sich mit der Position der Eingabetupel. Im Bezug auf relationale Datenbanken wäre eine mögliche Position, das Attribute also die Spalte einer Tabelle [CCT09; HDB17].

**Why-Not-Provenance** Bei fehlenden Ergebnissen beantwortet Why-Not-Provenance, die Frage *"Welche Eingabedaten fehlen für das korrekte Ergebnis?"*. Für diese Frage können insgesamt drei verschiedene Lösungen existieren. (i) Es fehlen Eingabedaten, um das korrekte Resultat zu erzeugen (instance-based). (ii) Die Operationen in der Abfrage weichen von der originalen Abfrage ab (query-based). (iii) Vorschläge bezüglich der Anpassung der Anfrage (refinement-based) [HDB17].

**Structural-Provenance** Mittels der Structural-Provenance können Änderungen in geschachtelten Daten wie etwa JSON oder XML verfolgt werden. Dieser Typ beantwortet demnach die Frage nach der Struktur bzw. nach Strukturänderungen von geschachtelten Daten [DH19; DH20].

### 2.4.3 Workflow Provenance

Die, in dem Workflow Provenance gewonnen, Metadaten ermöglichen die Beschreibung eines Entstehungsprozesses mittels eines gerichteten Graphen. Knoten stellen dabei ein beliebiges Arbeitsmodul dar durch den die Daten wandern, während Kanten die Abhängigkeiten zwischen den Arbeitsmodulen darstellen. Ein Arbeitsmodul stellt eine Funktion da, durch die Daten transformiert werden, wie etwa eine Filter-Funktion. Workflow Provenance lässt sich in die Dimensionen *Granularität*, *Anwendungsdomäne* und *Form* einteilen [HDB17].

#### Granularität

Der Informationsgehalt von Workflow Provenance kann in die folgenden zwei Granularitätsstufen unterteilt werden. Die daraus resultierenden Daten unterscheiden sich in ihrem Informationsgehalt.

**Grob** Die grobe Granularitätsstufe macht keine Annahmen über das Verhalten der Arbeitsmodul d. h. es werden keine Informationen über den Entstehungsprozess der einzelnen Datenelemente in den Arbeitsmodulen gemacht. Dementsprechend hängt ein Arbeitsmodul immer von allen Eingabedaten im Graphen ab.

**Fein** Die feine Granularitätsstufe baut auf der groben Granularitätsstufe auf und ermöglicht, das genau Verfolgen der einzelnen Datenelemente im kompletten Entstehungsprozess. Um dies zu ermöglichen muss das Verhalten der jeweiligen Knoten d. h. Arbeitsmodule bekannt sein. Die daraus resultierenden Metadaten sind so fein bzw. so genau wie beim Data Provenance Typen.

#### Form

Die Workflow Provenance unterscheidet zwischen drei verschiedenen Formen, die jeweils für einen unterschiedlichen Einsatzzweck angewendet werden können [HDB17]. In der folgenden Auflistung sind die drei Formen beschrieben.

**Prospektive Form** Die prospektive Form erfasst die Struktur des gerichteten Graphen, unabhängig von den Eingabedaten. Sie ermöglicht somit einen Überblick über die Knoten und Kanten. Die Übersicht ist statisch, da kein Durchlauf mit Daten benötigt wird.

**Retrospektive Form** Die retrospektive Form erfasst neben der Struktur ebenso alle Daten, die in den Arbeitsmodulen erzeugt wurden. Sie entscheidet sich somit von der prospektiven Form, die nur die statische Struktur des Graphen erfasst.

**Evolutionäre Form** Die evolutionäre Form erfasst alle Änderungen zwischen verschiedenen Eingabedaten und gibt Aufschluss, welche Änderungen im Graphen gemacht wurde oder wie die Eingabedaten sich unterscheiden.

#### Anwendungsdomäne

Workflow Provenance lässt sich in verschiedenen Domänen einsetzen. Herschel et al. [HDB17] haben in ihrer Arbeit insgesamt 4 Hauptdomänen ausgemacht, in denen Workflow Provenance eingesetzt wird. Die Anwendungsdomänen für Workflow Provenance sind wie folgt.

**Wissenschaft** In der Wissenschaft wird Workflow Provenance verwendet, um hauptsächlich grobgranulare Provenance in der retrospektiven Form zu generieren, die für die Untersuchung von z. B. Medizin- oder Umweltdaten verwendet werden können.

**Wirtschaft** In der Wirtschaft wird Workflow Provenance verwendet, um z. B. die Einhaltungen des Datenschutzes während einer Analyse nachzuweisen. Dabei ist die Granularität meistens auf Dateiebene und es handelt sich dabei um die retrospektive Form

**Datenanalyse** In der Datenanalyse wird eine feingranulare Workflow Provenance in der retrospektiven Form benötigt, um die daraus gewonnen Provenance Informationen z. B. für das Debugging von Analysen zu verwenden.

**Entwicklung** In der Entwicklung wird Provenance eingesetzt, um z. B. die Änderungen von Variablen während dem Lebenszyklus eines Programmes zu erfassen. Die Granularität und Form variiert dabei, je nach Anwendungsfall wie die feingranulare Erfassung von Variablen oder den grobgranularen Ablauf von Funktionsaufrufen.

Die Provenance-Informationen können für die Zuordnung von Eingabedaten zu Ergebnisdaten über einen Entstehungsprozess verwendet werden. Dadurch ermöglicht z. B. die Workflow Provenance die Verfolgung des Entstehungsprozess von Ergebnissen aus personenbezogenen Analysen. Die Form dabei ist die retrospektive Form, da für den Nachweis alle Provenance-Daten aus allen durchlaufenen Arbeitsmodulen benötigt werden. Die Granularität muss fein sein, dadurch ist gewährleistet das exakt bestimmt werden kann, welches Datenelement zu welchem Ergebnis beigetragen hat. Somit kann z. B. eine Einwilligungsänderung bis zum Analyseergebnis verfolgt werden. Mittels diesen Provenance-Informationen können effiziente Neuberechnung durchgeführt werden.

## 2.5 Relationales Datenmodell

Das relationale Datenmodell ist ein weitverbreitetes Datenmodell, das die Speicherung und Abfragen in Relationen ermöglicht. Relationen weisen eine eindeutige Struktur, wie Tabellen, der Daten auf und ermöglichen eine Abfrage der Daten durch die relationale Algebra. Das Modell ist durch die Verwendung von Relationen einfach, jedoch mächtig und ist in verschiedenen Datenbanksystemen wie etwa MySQL, PostgreSQL[9] und IBM Db2[10] implementiert. Dementsprechend ist das relationale Datenmodell ein wichtiger Bestandteil zur Speicherung und Analyse von Daten geworden [EN15].

Dieses Grundlagenkapitel legt den Grundstein für das Verständnis des relationalen Datenmodells und der relationalen Algebra und baut auf der Arbeit von Elmasri und Navathe [EN15] auf. Der erste Abschnitt beschreibt das relationale Schema, aus denen konkrete relationale Instanzen erstellt werden können. Die Notationen für relationalen Instanzen wird im darauffolgenden Abschnitt beschrieben. Der letzte Abschnitt beschreibt die Notation der relationale Algebra, gefolgt von der Definition von personenbezogenen Analysen für diese Arbeit.

### 2.5.1 Relationales Schema

Ein relationales Schema dient als Vorlage für die Erstellung von relationalen Instanzen. Mittels des Schemas wird die Struktur einer Instanz genau definiert. Sie gibt den Namen, die Attribute sowie den Datentyp der Attribute der Instanz vor. Nach Elmasri und Navathe [EN15] ist eine relationales Schema wie folgt definiert.

**Definition 2.5.1**

- *Ein Schema $\mathcal{S}$ ist eine geordnete Menge von n Attributen $\mathcal{A}$ definiert als $\mathcal{S}(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$ mit $n > 0$.*
- *Jedes Attribut $\mathcal{A}$ besteht aus einem eindeutigen Namen und einem Datentypen (Zeichenfolgen, numerische Typen und Datumstypen)*
- *Der Grad eines Schemas ist die Anzahl der Attributen in $\mathcal{S}$ : $n = |\mathcal{S}|$.*

[9] https://www.postgresql.org/
[10] https://www.ibm.com/de-de/analytics/db2

Abbildung 2.3 zeigt das Schema der Customer-Relation als Entity-Relationship-Diagramm (ER-Diagramm). *Customer* ist der Name des Schemas. Die einzelnen Attribute sind unter dem Namen der Relation aufgelistet. Bei der Instanziierung des Schemas wird die Ordnung der Attribute beibehalten. Der jeweilige Datentyp eines Attributes befindet sich rechts neben dem Namen des Attributes und ermöglicht eine Restriktion von erlaubten Werten.

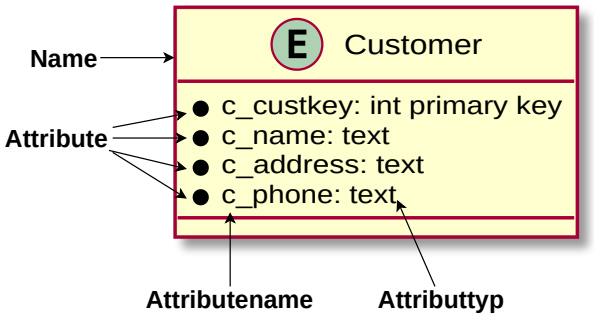



**Abbildung 2.3:** Beschreibung eines Schemas anhand der Customer-Relation

### 2.5.2 Relationale Instanz

Aus einem relationalen Schema $\mathcal{S}$ kann nun eine Relation abgeleitet werden. Nach Elmasri und Navathe [EN15] ist eine Relation wie folgt definiert.

**Definition 2.5.2**

- *Eine Relation $\mathcal{R}$ ist eine Instanz eines Schemas $\mathcal{S}$ und besteht aus einer Menge von n Tupeln definiert als $\mathcal{R} = \{t_1, t_2, \ldots, t_n\}$. Als eine Instanz vom Schema $\mathcal{S}$ hat die Relation $\mathcal{R}$ einen Namen.*
- *Ein Tupel $t \in \mathcal{R}$ ist eine geordnete Liste von i Werten definiert als $t =< v_1, v_2, \ldots, v_i >$. Die Ordnung, Anzahl und der Datentyp der Werte ist abhängig von den Attributen des Schemas $\mathcal{S}$. Der spezielle Datentyp $NULL$ wird für die Abwesenheit eines Wertes benutzt.*
- *Der Zugriff auf einen Wert $v_i$ im Tupel $t \in \mathcal{R}$ ist als $v_i = t[i]$ mit $0 \leq i \leq n$ definiert. Ein alternativer Zugriff auf einen Wert $v_i$ im Tupel $t \in \mathcal{R}$ ist als $v_i = t.\mathcal{A}_i$, wobei $\mathcal{A}$ ein Attribut aus dem Schema der Relation $\mathcal{R}$ ist.*

Abbildung 2.4 zeigt das Beispiel der Customer-Relation mit allen Elementen. *Customer* ist der Name der Relation, anhand dem die Relation z. B. in SQL-Abfragen referenziert werden kann. *c_custkey*, *c_name*, *c_address* und *c_phone* sind die Attribute der Customer-Relation. Sie sind spaltenweise aufgelistet. Alle Attribute bis auf *c_custkey* sind vom *TEXT*-Datentyp. *c_custkey* ist der eindeutige Primärschlüssel der Relation und hat entsprechend den *INT*-Datentyp. Unterhalb der Attribute sind zeilenweise die Tupel zu finden.

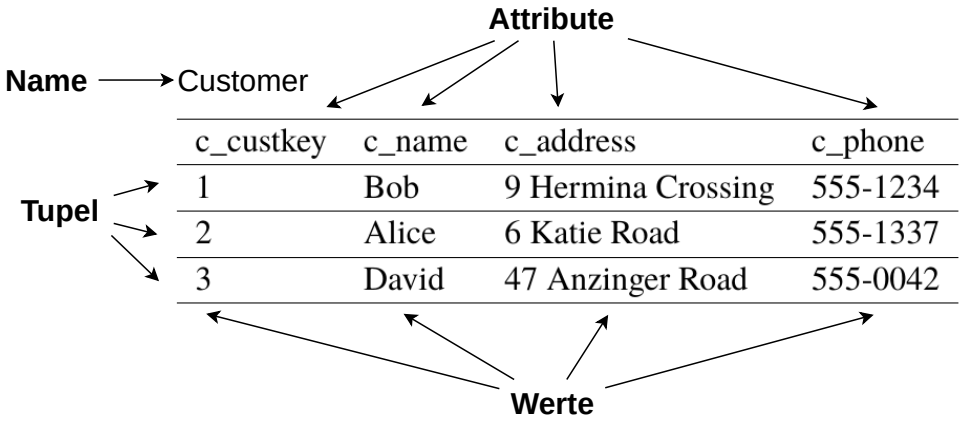



**Abbildung 2.4:** Beschreibung der Customer-Relation

### 2.5.3 Relationale Algebra

Die relationale Algebra definiert Operationen für die Manipulation der relationalen Instanzen. Dabei werden mengentypischen Operationen wie Vereinigung, Schnittmenge, Differenz und das kartesische Produkt für relationalen Instanzen definiert. Neben diesen Operationen definiert die relationale Algebra weitere Operationen für relationalen Instanzen, wie die Selektion, Projektion und Joins [EN15]. Diese Operationen können zu Anfragen kombiniert werden. Nach Elmasri und Navathe [EN15] ist eine relationale Anfrage wie folgt definiert.

**Definition 2.5.3 (Anfrage Q)**
*Sei eine relationale Anfrage definiert als $Q(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n)$ mit $n > 0$. $\mathcal{R}_n$ ist hierbei eine Relation, anhand derer die Anfrage $Q$ ausgeführt wird und $Q = \{t_1, t_2, \ldots, t_n | COND(t_1, t_2, \ldots, t_n)\}$ mit $t_n \in \mathcal{R}_n$. $COND(t_1, t_2, \ldots, t_n)$ ist hierbei ein bedingter Ausdruck, der entweder zu TRUE oder FALSE ausgewertet wird. Dabei stehen Operatoren wie AND, OR und NOT, sowie Vergleichsoperatoren zur Verfügung, anhand derer der Ausdruck auf den Relationen definiert wird.*

**Definition 2.5.4 (Personenbezogene Anfrage Q)**
*Eine personenbezogene Anfrage ist eine relationale Anfrage, die auf personenbezogenen Daten arbeitet. Personenbezogene Daten können direkt oder indirekt einer natürlich Person zugeordnet werden und sind in der DS-GVO und im Kapitel "Datenschutz-Grundverordnung"definiert. Sei nun eine Berechnung und das Ergebnis einer personenbezogenen Analyse definiert als $\mathcal{V} = Q(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n)$. $\mathcal{V}$ ist die DS-GVO konforme Ergebnisrelation der Analyse von $Q$ auf den Eingaberelationen $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$, bei denen die Bedingung $COND(t_1, t_2, \ldots, t_n)$ zu TRUE führt und bei denen nur Eingabetupel $t \in \mathcal{R}$ verwendet werden, für die eine Einwilligung gegeben wurde.*

### 2.5.4 Änderungen

Änderungen in den Eingabedaten von personenbezogenen Analysen führen zu inkonsistenten Analyseergebnissen. In dieser Arbeit werden nur Änderungen in Bezug auf den Entzug der Einwilligung in Betracht gezogen. Ein Einwilligungsentzug in den Eingabedaten sorgt dafür, dass das erzeugte Analyseergebnis nicht mehr rechtskonform ist. Eine erneute Berechnung ist notwendig, damit kein Rechtsverstoß entsteht. Änderungen werden wie folgt definiert.

**Definition 2.5.5 (Änderungen)**

- *Änderungen werden definiert als Änderungen der Tupelwerte in den Eingaberelationen. Sei* $t_0 \in \mathcal{R}$ *ein Tupel in einer Eingaberelation* $\mathcal{R}$ *zum Zeitpunkt* $p_0$. $p_0$ *ist der Zeitpunkt in der das Analyseergebnis* $\mathcal{V} = Q(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n)$ *initial berechnet wurde.* $t_0' \in \mathcal{R}$ *ist nun das Tupel* $t_0$ *zu Zeitpunkt* $p_1$ *mit* $p_1 > p_0$. *Eine Änderung von* $t_0$ *hat nun dann stattgefunden wenn* $\exists i \in \{0, 1, \ldots, n\} : t_0[i] \neq t_0'[i]$, *wobei n der Grad des Schemas* $\mathcal{R}$ *ist.*
- *Bei einer Änderungen in einen von den Eingaberelationen* $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$ *muss* $\mathcal{V} = Q(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n)$ *erneut berechnet werden, damit das Ergebnis konsistent mit den Änderungen ist.*
- *Sei C nun die Menge aller Tupel aus einer der Eingaberelationen* $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$, *bei denen eine Änderung zu einem Zeitpunkt* $p_1$ *stattgefunden hat, so dass gilt* $C \subseteq \mathcal{R}$ *und* $C \neq \emptyset$.

# 3 Verwandte Arbeiten

Dieser Abschnitt befasst sich mit Arbeiten, die mit dieser Arbeit verwandt sind. Es wird der Stand der Technik erläutert und welche Relevanz sie für diese Arbeit haben. Dazu werden im ersten Abschnitt verschiedenen Ansätze für die Modellierung von Provenance gezeigt, gefolgt von einem Vergleich bereits bestehender Provenance-Lösungen, sowie Indizes für Provenance-Daten. Der letzte Abschnitt befasst sich mit der View-Update Problematik.

## 3.1 Provenance-Modelle

Verschiedene Modellierungslösungen, um Provenance zu modellieren und auch grafisch darzustellen wurden bereits entwickelt. In diesem Abschnitt werden die Provenance Modelle *W3C Prov* und ein für DS-GVO ausgelegtes Provenance Modell behandelt.

### 3.1.1 W3C PROV

*W3C PROV* ist eine Standardisierung für die Modellierung von Provenance-Abläufen und Provenance-Daten, welche vom World Wide Web Consortium (W3C) vorangetrieben wird [GMB+13]. Die Autoren Gil et al. [GMB+13] beschreiben PROV hierbei, als eine Spezifikation für die Beschreibung von Provenance-Daten, mittels der Objekte und ihre Aktivitäten beschrieben werden können.

Das Modell unterscheidet zwischen Entitäten und Aktivitäten. Als Entitäten werden Objekte beschrieben, die in einer physische, konzeptuellen, digitalen oder jedweden weiteren Form existieren können wie z. B. eine Person oder eine Rechnung. Aktivitäten erzeugen neue Entitäten, z. B. *Schreiben einer Rechnung*, wobei *Schreiben* die eigentliche Aktivität ist und als Ergebnis die Entität *Rechnung* erzeugt [GMB+13].

Gil et al. [GMB+13] beschreiben neben den Entitäten und Aktivitäten noch folgende Konzepte.

**Agent** Agenten nehmen Teil in einer Aktivität, lösen diese aber nicht aus z. B. das Erstellen einer Rechnung mittels eines Computers. Der Computer dient hierbei als Agent in der Aktivität, da er zwar an der Aktivität teilnimmt, aber nicht der Auslöser ist. Es ist aber auch möglich, dass ein Agent auch gleichzeitig eine Entität darstellt.

**Rolle** Durch Rollen wird beschrieben, in welcher Weise ein Agent in einer Aktivität teilnimmt

**Änderungen** Durch Änderungen werden Revisionen und Ableitungen einer Entität dokumentiert, z. B. neue Version eines Textdokuments

**Zeit** Die Erfassung von Zeitpunkten ist ein wesentlicher Bestandteil von Provenance. In *W3C PROV* werden Zeitpunkte von Aktivitäten festgehalten z. B. wann ein Dokument erstellt wurde

**Alternativen und Spezialisierung** Spezialisierungen in *W3C PROV* sind Erweiterungen von Entitäten mit zusätzlichen Attributen. Dabei teilt sich eine spezialisierte Entität, die selben fest vorgegeben Attributen von ihrer Basisentität, fügt aber weitere eigene Attribute hinzu. Dadurch können spezielle Entitäten für Provenance erstellt werden. Als Alternativen werden Entitäten bezeichnet, die einen anderen Änderungsgrad bzw. eine andere Revision aufweisen

Grafisch werden *W3C PROV* Spezifikationen als Graphen modelliert. Die Knoten stellen dabei die Entitäten, Agenten und Aktivitäten dar, während Kanten die Beziehung unter ihnen beschreiben, wobei *W3C PROV* vorgibt welche Beziehungen erlaubt sind.

Die Abbildung 3.1 zeigt ein Diagramm für die Erstellung einer Rechnung nach einem Verkaufsabschluss. Die *Rechnung*-Entität wird von der *Verkaufen*-Aktivität erzeugt. Der *Verkäufer*-Agent ist für den Verkauf verantwortlich und stellt die Rechnung aus.

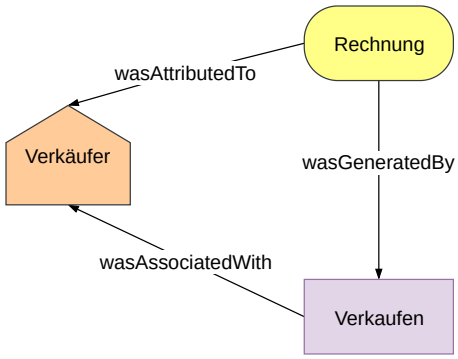



**Abbildung 3.1:** Beispiel eines W3C PROV Diagramms

Des weiteren besteht *W3C PROV* aus mehreren Unterprojekten bzw. Dokumenten [GM13], wie z. B. einer genauen Beschreibung des Datenmodells[1] und eines XML-Schemas[2] für die Darstellung der Provenance-Daten in einem XML-Format.

*W3C PROV* ist ein sehr generisches Modell für die Erstellung von Provenance-Abläufen und kann somit nach entsprechender Anpassung für jede Domäne eingesetzt werden. Eine genaue Spezifikation für DS-GVO Provenance gibt *W3C PROV* somit nicht vor und entsprechend müssen die Entitäten, Aktivitäten sowie Agenten und Rollen erst definiert werden, bevor ein Ablauf in Rahmen eines DS-GVO Ablaufs erstellt werden kann. Beispielsweise können Aktivitäten als DS-GVO Ereignisse modelliert werden (z. B. Entziehen von Rechten). Änderungen von Entitäten und die zeitliche Erfassung von Aktivitäten ist in *W3C PROV* bereits gegeben und ist in DS-GVO ein wichtiger Bestandteil, da ein Unternehmen in der Pflicht ist solche Informationen abzuspeichern.

Des weiteren beschreibt *W3C PROV* nur die Entitäten und Aktivitäten, die in der Vergangenheit passiert sind. Genaue Angaben bzw. Workflows, die nötig sind, um z. B. eine neue Entität im Modell zu erzeugen sind in *W3C PROV* explizit nicht vorhergesehen. Es gibt jedoch die Möglichkeit anhand einer generischen *prov:Plan* Entität einen Ablaufplan zu beschreiben [GMB+13].

[1] https://www.w3.org/TR/2013/REC-prov-dm-20130430/
[2] https://www.w3.org/TR/2013/NOTE-prov-xml-20130430/

### 3.1.2 GDPRov

Das Sammeln und Archivieren von Provenance-Daten ist für den Nachweis der Erfüllung der DS-GVO-Richtlinien ein wesentlicher Bestandteil. *GDPRov* [PL17] ist ein Modell, welches die Vorgänge im Rahmen der DS-GVO erfasst und Aktivitäten beschreibt, die ein Kunde bzw. ein Nutzer ausführen kann wie z. B. das Entziehen der Einwilligung für die Datenverarbeitung. Dabei baut *GDPRov* auf *W3C PROV* und *P-Plan*[3] auf. *W3C PROV* wird für die Darstellung der Provenance-Daten verwendet. P-Plan wird verwendet, um den Ablauf zu modellieren. *GDPRov* erweitert die *W3C PROV* Entitäten um eigene Entitäten, die die DS-GVO Terminologie reflektieren und gibt vor wie diese in Bezug zueinander stehen bzw. stehen können. Wie bereits in Abschnitt 3.1.1 bereits beschrieben, ist der *W3C PROV* Standard für die Darstellung von bereits geschehenen Aktivitäten und den daraus resultierenden Entitäten entwickelt worden. *W3C PROV* bietet zwar eine Möglichkeit für die Repräsentation von Plänen mittels der Entität *prov:Plan*, jedoch nicht die genauen Schritt-für-Schritt Ablaufmodellierung. Aus diesem Grund haben Pandit und Lewis [PL17], die Ontologie *P-Plan* ausgewählt, die es ermöglicht Ablaufpläne und ihre Zusammenhang mit Provenance-Daten zu beschreiben.

Pandit und Lewis [PL17] unterscheiden zwischen den eigentlichen Provenance-Daten und den verschiedenen Prozessen. Daten sind immer einer Person zugeordnet und haben verschiedene Stufen der Anonymisierung. Welche Daten gespeichert werden hängt, von der Domäne ab in welcher der Nutzer bzw. sich das Unternehmen befindet. Das können z. B. Daten über Hobbies oder Vorlieben sein. Dabei können die Daten komplett unanonymisiert, pseudoanonymisiert, pseudoanonymisiert mit der Möglichkeit zur vollständigen Rekonstruktion der ursprünglichen Daten und anonymisiert vorliegen.

Um an diese Daten zu gelangen und sie rechtskonform verarbeiten zu können, muss der Nutzer eine Einverständniserklärung abgeben. In dieser Erklärung wird dem Nutzer oder Kunden erklärt für welchen Zweck seine Daten erhoben werden. *GDPRov* modelliert das Einverständnis als *ConsentAgreement* und beschreibt alle Optionen für die der Nutzer sein Einverständnis gegeben hat. Das Formular, das die Einverständniserklärung vom Nutzer einholt wird in *GDPRov ConsentAgreementTemplate* bezeichnet. Bei Veränderungen des Formulars wird eine neue Instanz angelegt, somit ist gegeben dass das Einverständnis mit dem Originalformular übereinstimmt und das Unternehmen nachweisen kann, für welches Formular es eine Einverständnis bekommen hat [PL17].

Prozesse werden als P-Plan abgebildet. P-Plan ist ein Modell zur Modellierung von wissenschaftlichen Abläufen und besteht dabei aus einer Anzahl von Schritten, die zu einem Ziel bzw. einem Plan führen. P-Plan erweitert dazu PROV-O von *W3C PROV* Pandit und Lewis [PL17] haben folgende Schritte ausgearbeitet:

**ConsentStep** Statusbeschreibung des Einverständnisses

**DataCollectionStep** Sammeln von Daten

**DataDeletionStep** Löschen von Daten

**DataSharingStep** Das Teilen von Daten mit Drittpartnern

**DataStorageStep** Speichern von Daten

[3] http://vocab.linkeddata.es/p-plan/

**DataTransformationStep** Transformation der Daten z. B. in ein anderes Format

**DataAnonymisationStep** Ist eine Subklasse von DataTransformationStep mit der Eigenschaften, das Daten anhand der Anonymisierungsstufe anonymisiert werden

**DataArchivalStep** Langfristige Archivierung der Daten. Der Schritt ist eine Unterklasse von DataTransformationStep und DataStorageStep, da die Daten für die Archivierung in ein Format transformiert werden

Anhand dieser Schritte können entsprechend Prozesse bzw. P-Plan Pläne modelliert werden. Jeder Kunde hat zu jeder Zeit die Möglichkeit seine Daten einzusehen, dies kann z. B. als *DataAccessProcess* abgebildet werden, der durch die Schritte *DataCollectionStep* und *DataTransformationStep* geht. Diese Prozesse können dann innerhalb der *prov:Plan*-Entitäten eingebettet werden und sind damit Teil der Provenance. Änderungen von Prozessen werden dadurch ebenso korrekt abgebildet, da *W3C PROV* dies bereits durch *Revisionen* ermöglicht [PL17].

Durch die Spezialisierung auf die DS-GVO-Domäne des *W3C PROV* Standards ist *GDPRov* ein vielversprechender theoretischer Startpunkt für die Implementierung einer Lösung, die sich mit der Analyse von personenbezogenen Daten mit Hilfe von Provenance-Daten beschäftigt.Pandit und Lewis [PL17] haben für die praktische Evaluation ihres Modells *SPARQL*[4] eingesetzt.

Ontologien wie W3C-Prov und vorallem GDPRov brechen den Rechtstext der DS-GVO in Entitäten und Beziehungen auf. Dadurch können Workflows in Bezug auf Provenance und der DS-GVO abgebildet werden. Sie befassen sich vorrangig mit der Beschreibung der DS-GVO als Abläufe. Das Ziel dieser Arbeit ist jedoch die DS-GVO konforme Datenverarbeitung von personenbezogenen Daten und unterscheidet sich in dieser Hinsicht von der Modellierung von Abläufen.

## 3.2 Provenance-Lösungen

Im folgende Abschnitt werden verschiedene Provenance-Lösungen vorgestellt. Dabei hat jede Lösung verschiedene Ansätze und Verwendungszwecke für das Erfassen von Provenance-Daten. Anhand dieser Ansätze werden Anforderungen an die existierenden Lösungen aufgestellt, die für die Implementierung der DSVGO-Lösung in dieser Arbeit unerlässlich sind.

### 3.2.1 RAMP

Ikeda et al. [IPW11] stellen ihrer Arbeit die Provenance-Lösung *Reduce And Map Provenance (RAMP)* vor. Das System befasst sich mit der Erfassung von Provenance-Daten in Generalized Map And Reduce Workflows (GMRWs) in *Apache Hadoop*[5]. *MapReduce* [DG04] ist ein Programmiermodell, das die parallele Verarbeitung von sehr großen Daten ermöglicht. Dazu werden *Map* und *Reduce* Funktionen auf einem Cluster von Computern ausgeführt, das erlaubt eine hochskalierbare Ausführung von MapReduce-Jobs. In Apache Hadoop werden *MapReduce*-Jobs als *Transformationen* bezeichnet. Um den Nutzer bei der Fehleranalyse zu unterstützen, erfasst RAMP alle Map und Reduce Vorgänge als Graphen, wobei die Knoten die Transformationen darstellen und

[4] https://www.w3.org/TR/rdf-sparql-query/
[5] https://hadoop.apache.org/

die Kanten den Datenfluss. Dies geschieht für den Nutzer transparent, sodass keine Konfigurationen oder API-Aufrufe erfolgen müssen. RAMP umhüllt dazu die Komponenten von Apache Hadoop mit eigenen Komponenten, die die Ein- und Ausgabedaten, sowie die Transformationen erfassen und sie mit Metadaten anreichern. Diese sogenannten Wrapper-Komponenten sind folgende [IPW11].

**Record Reader** Eingabedaten werden in Tupel der Form (key,value) von RAMP transformiert. Der key-Wert wird von RAMP automatisch gesetzt und dient zur Verfolgung der Daten und wird für den Aufbau des Graphen benötigt.

**Mapper** Eine benutzerdefinierte Funktion, die beschreibt wie Daten von einem Format in ein anderes Format transformiert werden

**Combiner** Optionale Zusammenführung der Map Ergebnisse anhand eines Schlüssels

**Reducer** Eine benutzerdefinierte Funktion, die das Ergebnis vom Mapper als Eingabe hat und eine Berechnung auf dieser Eingabe ausführt (z. B. das Filtern von Daten)

**Record Writer** Ergebnis des Reducers wird als Ausgabedatei in einem spezifizierten Format geschrieben

Die erzeugten Metadaten von den Wrapper-Komponenten werden in Apache Hadoop eigenen Dateisystem (Hadoop Distributed File System (HDFS)) abgelegt und können für die Nachverfolgung der MapReduce-Jobs verwendet werden. Die Metadaten liegen dabei in Tupelform in Tabellen vor und können durch Join-Operationen anhand der hinzugefügten ID erfasst werden. Die Arbeit von Ikeda et al. [IPW11] beschreibt jeweils einen Algorithmus für das Backward- und Forward-Tracing, diese müssen jedoch vom Nutzer selbst, in Apache Hadoop eigener Sprache Apache Pig[6] oder Apache Hive[7] implementiert werden. Dies kann je nach Menge der Daten und Transformationen komplex werden.

Für die Evaluation wurden ein *Wordcount* MapReduce-Job mit jeweils 100, 300 und 500 GB an Daten und ein *Terasort*[8] MapReduce-Job mit jeweils 93, 279 und 466 GB an Daten ausgeführt. Für die auszuführende Umgebung haben sich die Autoren von Ikeda et al. [IPW11] für Amazon Elastic Compute Cloud (EC2) [9] entschieden mit 51 Computing Instanzen, die jeweils über 2 Kernen sowie 7.5 GB Arbeitsspeicher verfügen. Das Ergebnis der Evaluation war ein erhöhter Aufwand in Berechnungszeit sowohl Festplattenspeicher. Dabei war die Höhe des Aufwandes je nach MapReduce-Job unterschiedlich hoch. Für Wordcount wurde ein Anstieg um ca. 76% in Berechnungszeit gemessen und ein signifikanter Anstieg des Festplattenspeichers, da hier mehr Provenance-Daten gespeichert als Ergebnisse berechnet wurden. Für Terasort wurde ein Anstieg der Berechnungszeit um ca. 20%, sowie ein Anstieg des Festplattenspeichers um ca. 21% gemessen.

[6] https://pig.apache.org/
[7] https://hive.apache.org/
[8] https://hadoop.apache.org/docs/r3.2.0/api/org/apache/hadoop/examples/terasort/package-summary.html
[9] https://aws.amazon.com/de/ec2/

### 3.2.2 Newt

Newt [LDY13] ist eine Lösung für das Erfassen von Provenance-Daten. Im Gegensatz zu anderen Lösungen ist Newt generisch implementiert, sodass die Lösung unabhängig vom verwendetem VSDB-System eingesetzt werden kann. Dazu bieten die Autoren von Newt eine Application Programming Interface (API) zum Erfassen von Provenance-Daten an. Diese Daten werden von Agenten erfasst und in eine SQL-Tabelle geschrieben, die sich in einem Cluster befindet. Jeder Agent bekommt dabei eine eigene Tabelle, die der Agent mit Provenance-Daten befüllt. Ein zentraler Controller ist hierbei für die Fehlertoleranz zuständig und erstellt bei Fehlern die Agenten-Tabellen neu. Jeder Agent meldet sich beim Start am Controller an. Nachdem alle Provenance-Daten von den Agenten erfasst wurden, importiert Newt diese Daten in einen indizierte SQL-Tabelle, dadurch kann ein Nutzer Daten zurückverfolgen. Newt bewerkstelligt, dies durch das Verbinden der einzelnen Agent-Tabellen durch Join-Operationen. Logothetis et al. [LDY13] haben für die Demonstration der Funktionsweise jeweils Apache Hadoop und Hyracks[10] verwendet.

Die Evaluation von Newt bestand aus dem Erfassen des Mehraufwandes bzgl. Berechnungsdauer und Speicherverbrauch. Logothetis et al. [LDY13] haben dafür die Systeme Apache Hadoop und Hyracks auf einen Cluster mit 17 Rechnern, die jeweils über 2.4 GHz an Leistung verfügen sowie jeweils 4 GB an Arbeitsspeicher, wobei in der Arbeit [LDY13] nur die Ergebnisse von Apache Hadoop veröffentlicht wurden, da Hyracks ähnliche Ergebnisse liefert.

Getestet wurde Newt mit realen Workflows, wie einem Genom Assembler und zwei Analyseprogrammen (Empfehlungsgeber, Artikelähnlichkeit) von *Apache Mahout*[11]. Für den Genom Assembler wurde ein Mehraufwand von 20% gemessen und jeweils 36% und 31% für die Analyseprogramme von Apache Mahout.

### 3.2.3 Titian

Titian [IES+18; IST+15] ist eine Provenance-Lösung, die in das VSDB-System Apache Spark integriert ist. Software-Agenten innerhalb von Titian erfassen den gerichteten azyklischen Graphen von Spark, annotieren Eingabedaten mit einer ID und verknüpfen die Ergebnistupel mit Eingabetupel [IST+15]. Neben den Erfassen und Erstellen von Provenance-Daten ermöglicht Titian, durch seine tiefe Integration in Spark, eine direkte Interaktion mit den Daten und Zwischenergebnissen über die *Spark-Shell*. Dazu wurden die Spark-internen RDD-Klassen, um Provenance-Funktionalitäten von Interlandi et al. [IES+18; IST+15] erweitert. Diese Funktionen ermöglichen es dem Nutzer innerhalb des Graphen vor und zurückzuspringen. Ergebnisse von Zwischenberechnungen und Provenance-Assoziationen werden dabei nicht als Dateien auf Festplatten oder in Datenbanken gespeichert, sondern sind direkt innerhalb des In-Memory Layer von Spark, dem sogenannten *BlockManager*, verfügbar und deshalb direkt zugreifbar [IST+15]. Die Provenance-Daten liegen in Tupelform vor. Für die Analyse können diese Daten auf einen externen Speicher wie z. B. HDFS persistiert werden.

[10] http://hyracks.weebly.com/
[11] https://mahout.apache.org/

Die Autoren vergleichen Titian mit Newt [LDY13] und RAMP [IPW11]. Dazu wurden vier Dateien der Größe 500 MB, 5 GB, 50 GB, 500 GB generiert. Anhand jeweils einem *Wordcount* und *Grep* (Suche nach Zeichenketten) Spark-Jobs wurden dabei die Berechnungszeit, Speicherverbrauch und Tracing-Laufzeit gemessen. Die Testumgebung bestand dabei aus einem Cluster von 17 Computer mit jeweils vier Kernen zu je 3.4 GHz, 32 GB Arbeitsspeicher und insgesamt 1 TB an Festplattenspeicher. Das Ergebnis der Evaluation ist das Titian für Wordcount und Grep eine Berechnungsaufwand von jeweils max. 1.29x (500 MB) und 1.27x (500 GB) aufweist, RAMP einen Berechnungsaufwand von jeweils max. 1.40x (500MB) und 3.2x (50GB). Newt weist einen fünfzehnfachen Berechnungsaufwand für den Grep-Job auf und für den Wordcount konnten keine Ergebnis für den 500 GB Datensatz gemessen werden, da mit der Testumgebung der Autoren Newt nicht in der Lage war Daten über 80 GB zu verarbeiten [IST+15]. Leistungseinbrüche lassen sich teilweise, im Fall von Newt, auf die Datenbank zurückführen, aber auch an generellen Leistungseinbußen bei der Verarbeitung von sehr großen Daten in Spark selbst.

### 3.2.4 BigDebug

Eine weitere Lösung, die auf *Apache Spark* und *Titian* aufbaut ist BigDebug [GIY+16]. Der Fokus der Autoren ist das interaktive Debugging von Spark. BigDebug bietet dazu eigene Debug-Funktionen an. Diese Funktionen können innerhalb der Spark-Shell aufgerufen werden. Dabei stoppt BigDebug den Ablauf einer Berechnung nicht, da dies zu einem großen Aufwand führt. Stattdessen werden Debug-Breakpoints simuliert d. h., dass die Berechnung im Hintergrund weiterläuft. Beim Halten auf einem Breakpoint erhält der Nutzer, die Möglichkeit anhand der RDD-Klassen, wie bei Titian, vor und zurückzuspringen und sich Zwischenergebnisse anzuschauen. Die Position eines Breakpoint kann der Nutzer im Code setzen, dabei kann er optional eine Guard-Funktion definieren, wann ein simulierter Breakpoint halten soll. Funktionen wie *Step-Over* und *Resume* sind in BigDebug vorhanden. Auch kann der Nutzer Codeänderungen, z. B. eine Fehlerbehebung, direkt in die Ausführung hinzufügen und vom Breakpoint die Berechnung weiter ausführen lassen. Das Erfassen von Provenance-Daten für das Debugging funktioniert transparent und interaktiv über die Spark-Shell.

Für die Evaluation wurde BigDebug auf einer Umgebung mit sechzehn Maschinen mit vier Kernen, die jeweils über 3.4 GHz an Leistung verfügen, sowie jeweils 32 GB Arbeitsspeicher und insgesamt 1 TB an Festplattenspeicher [GIY+16]. Getestet wurde mit PigMix L1[12], Wordcount und Grep. Für PigMix L1 wurden Daten der Größe 1 GB, 10 GB, 50 GB, 100 GB, 150 GB und 200 GB verwendet, für Grep wurden Daten der Größe 20-90 GB verwendet, wobei die Dateigröße immer um 10 GB ansteigt bis maximal 90 GB und für Wordcount wurden Daten der Größe 0.5 GB bis 1 TB verwendet, wobei der Anstieg der Dateigröße logarithmisch berechnet wurde. Das Ergebnis der Evaluation von BigDebug [GIY+16] hat gezeigt, dass die Ausführung von PigMix L1 max. 1.38x, Grep max. 1.76x und Wordcount max. 2.5x länger braucht. Das Pausieren und Weiterführen von Breakpoints hatte fast keinen Mehraufwand [GIY+16].

[12] https://cwiki.apache.org/confluence/display/PIG/PigMix

### 3.2.5 Pebble

Pebble [DH19; DH20] ist eine Provenance-Lösung in Apache Spark, die auf das Erfassen von Provenancen-Daten von strukturierten Daten spezialisiert ist. Diese neue Art von Provenance nennen die Autoren Diestelkämper und Herschel [DH19] *Structural Provenance*. Informationen über Änderungen in der Reihenfolge, sowie Schachtelung können für das Debugging von großen Daten hilfreich sein. Dazu erweitert Pebble die bestehenden RDD-Klassen von Spark um Funktionen, die die Änderungen von Daten erfassen. Diese Daten können dann mittels eines speziell dafür entwickelten Tree-Pattern-Matching (TPM)-Algorithmus angezeigt werden. Ein Tree-Pattern beschreibt hierbei den strukturellen Zusammenhang von Daten in einer Baumstruktur. Knoten beschreiben die Daten im Schema, während Kanten die Beziehungen der Knoten untereinander beschreiben. Die Kanten schränken das Schema entsprechend ein, da das Schema nur korrekt ist wenn alle Vater-Kind-Beziehungen erfüllt werden. Dazu können noch zusätzlich Beschränkungen auf Knotendaten auferlegt werden, sowie Beschränkungen auf die Anzahl an Vater-Kind-Beziehungen [DH19]. Der TPM-Algorithmus filtert den Baum nach einem Muster und entfernt zusätzlich alle Ergebnisse, die entweder die Kardinalität nicht erfüllen oder nicht mit den Knotendaten übereinstimmen. Die Filterfunktion kann dabei von Nutzer selbst angegeben werden. Des weiteren erweitert Pebble, die Funktionalität von Titian [IST+15], um das Backtracking auf strukturierten Daten zu ermöglichen. Durch eine Anbindung an *Jupyter Notebooks*[13] kann der Nutzer die Ergebnisse vom TPM-Algorithmus grafisch einsehen.

Diestelkämper und Herschel [DH20] haben Pebble in Bezug auf Laufzeit- und Speicherverbrauch, sowie Abfragegeschwindigkeit evaluiert. Des weiteren wurde ein Vergleich zwischen der Provenance-Lösung *Titian* durchgeführt. Für die Evaluation wurden ein Spark-Cluster mit drei Worker-Knoten bereitgestellt. Jeder Knoten verfügt über 8 Kernen mit jeweils 256 GB an Arbeitsspeicher und Solid State Drive (SSD) Speicherplatz. Insgesamt wurden fünf Testdurchläufe mit dieser Einstellung durchgeführt, dabei wurden verschachtelte Daten von Twitter[14] (ein Kurznachrichtendienst) und DBLP[15] (eine Online-Bibliothek) verwendet. Die Anfangsgröße des Datensatz war 100 GB und wurde bei jedem Testlauf um 100 GB erhöht. Der erhöhte Laufzeitaufwand für Twitter-Daten betrug ca. 20% - 70%, wobei der Aufwand konstant für jede Datengröße geblieben ist. Für DBLP betrug der erhöhte Aufwand 13% - 30%, wobei hier der Aufwand ebenso konstant für jede Datengröße blieb. Eine Erhöhung von ca. 800 MB für die Twitter-Daten und ca. 12 GB für den DBLP-Datensatz wurde festgestellt.

Für die Ausführung der Abfragen hat Pebble für die Twitter-Daten ca. 10 und ca. 45 Minuten benötigt. Für DBLP wurde eine Ausführungszeit von ca. 11 und 45 Minuten gemessen. In Pebble können Abfrage mittel einer *Eager*-Einstellung beschleunigt werden. Dazu werden Zwischendaten für die Zurückverfolgung gespeichert und somit sind schnellere Abfragezeiten möglich [DH20].

Ein Vergleich zwischen *Titian* und *Pebble* hat einen Aufwand von respektiv 5.89% und 6.98% ergeben. Ein vollständiger Vergleich ist jedoch nicht möglich, da *Titian* keine Structural Provenance unterstützt.

[13] https://jupyter.org/
[14] https://twitter.com/?lang=de
[15] https://dblp.uni-trier.de/

### 3.2.6 Lipstick

Lipstick [ADD+11] ist eine Provenance-Lösung für das Erfassen von *MapReduce* Workflows mittels *Pig Latin* in Apache Hadoop. Lipstick baut auf Modulen auf, die den Zustand, sowie Pig Latin Queries in einem 5-Tupel-Datenformat beschreiben. Module sind mit anderen Modulen über zusätzliche Knoten verbunden, die die Eingabe und Ausgabe eines Moduls widerspiegeln. Aus dieser Verkettung von Modulen und Zusatzknoten wird ein Graph modelliert, der eine grobe und feine Provenance aufweist. Grobe Provenance beschreibt den Ablauf einer Sequenz der Module als Workflow, sowie die dazugehörenden Ein- und Ausgabedaten. Feine Provenance baut auf der groben Provenance auf und fügt weitere Informationen wie den Status hinzu. Durch diese Unterscheidung der Provenance kann der Nutzer aus dem Graphen heraus- oder hineinzoomen. Durch das Zoomen können mehr oder weniger Informationen bezüglich des Workflows angezeigt oder ausgeblendet werden.

Amsterdamer et al. [ADD+11] haben für die Evaluation von Lipstick ein eigenes Framework mit Namen *WorkflowGen* entwickelt. Das Framework beinhaltet zwei verschiedene Workflows. Ein Workflow, die den Ablauf eines Autokaufs innerhalb einer Auktion abbildet und ein Workflow für arktische Wetterstationen mit Wetterdaten von 1961 bis 2000 [ADD+11]. Die Evaluation wurde teils auf einen Cluster mit 27 Kernen und teils auf einem MacBook Pro mit einem i7 Prozessor ausgeführt. Dabei wurde die die Berechnungszeit der Workflows mit und ohne Provenance verglichen. Für den Auktion-Workflow wurden 4 Autohäusern mit insgesamt 20000 Autos (5000 pro Haus) angelegt. Die benötigte Zeit ohne Provenance betrug 3.8 Sekunden und mit Provenance 11.9 Sekunden (für jeweils 100 Gebote pro Autohaus). Für die Evaluation mittels der Wetterstationen wurden drei Workflows getestet mit unterschiedlichen Topologien. Der Mehraufwand betrug dabei jeweils 16.5%, 20% und 35%.

### 3.2.7 PROVision

PROVision [ZAI19] ist eine Provenance-Lösung, die für das Debugging von wissenschaftlichen Workflows entwickelt wurde. Der Fokus dieser Lösung ist das Rekonstruieren bzw. Berechnen von Provenance-Daten auf Anfrage, anhand von Eingabedaten, Zwischenergebnissen und einer Workflowbeschreibung. Getrieben ist der Fokus dadurch, dass das Erfassen von Provenance-Daten einen Aufwand im System erzeugt. Diese Aufwand ist nicht immer notwendig bzw. ist nur dann von Nutzen, wenn Fehler im Workflow entstehen, z. B. wenn sich der Input ändert. Dazu speichert PROVision keine Provenance-Daten, wie z. B. in einer Input-Output-Tabelle. Für das Tracing wird ein sogenannter *Workflow-Module-Deskriptor* gespeichert. Dieser Deskriptor beschreibt alle nötigen Informationen für das Berechnen der Provenance-Daten wie z. B. Format der Daten, Transformationen und verwendete Parameter. Anhand der Beschreibung kann ein Plan erzeugt werden, der die Ergebnisse des Workflows, annotiert mit Provenance-Daten, berechnet [ZAI19]. Neben der Erzeugung dieses Planes kann PROVision, fehlende Programmparameter für Drittprogramme finden. Um dies zu ermöglichen werden verschiedene Parameter getestet, bis der berechnete Output mit dem Original-Output übereinstimmt.

Die Evaluation [ZAI19] der PROVision Lösung wurde anhand drei Jobs getestet, die auf einem Computer mit 24 Kernen, die jeweils über 2.20 GHz an Leistung verfügen, sowie insgesamt 64 GB an Arbeitsspeicher ausgeführt wurden. Der erste Job vergleicht eine Gensequenz mit einer Reihe von Referenzgenen. Job 2 findet gleiche Entitäten in zwei verschiedenen Datenquellen. Der dritte und

letzte Job ist das Finden und Entfernen von Duplikaten. Gemessen wurde die Ausführungszeit und der zusätzliche Platzverbrauch. Der Platzverbrauch für Job 1 beträgt ca 20% für eine Input-Datei der Größe 3.5 GB, für Job 2 ist der Platzverbrauch um 20% größer für eine Input-Datei der Größe 615 KB, der dritte Job hat eine Platzverbrauch von ca 7-17% für eine Input-Datei der Größe 4.6 MB. Entsprechend beträgt der Aufwand für die Rückberechnung der Workflows anhand des Deskriptor für den ersten Job ca. 30%, für Job 2 beträgt der Berechnungsaufwand ca. 50% und der dritte Job benötigt für die Berechnung ca. 480% mehr. Diese Berechnung finden auf einen vollständigen Satz von Daten statt ohne die Verwendung von Zwischenergebnissen.

### 3.2.8 SAMbA

Ein weitere Lösung, die auf *Apache Spark* aufbaut ist Spark provenAnce MAnagement on RDDs (SAMbA) [GSM+18]. Der Fokus von SAMbA ist das Erfassen von Provenance-Daten von wissenschaftlichen Workflows innerhalb von Apache Spark. Durch die Erweiterung der RDD-Klassen von Spark ähnelt das Konzept von SAMbA, den Konzepten wie z. B. Titian [IST+15] oder BigDebug [GIY+16]. SAMbA speichert die Provenance-Daten konform zu dem W3C-Prov-Standard ab und somit können diese Provenance-Daten von anderen W3C konformen Provenance-Lösungen geladen werden.

Das System besteht aus insgesamt vier Komponenten. Die erste Komponenten ist der *Retrospective And Domain Provenance Manager*, die vorgibt wie lesende Daten ausgewertet werden. Dazu implementiert ein Nutzer das RDD-Schema. Anhand des Schemas werden die Daten aufgeteilt in eine *Data Collection* und in einen Umschlag (Envelope) umhüllt. Dieser Umschlag liefert Provenance-Daten wie z. B. Datenabhängigkeiten, Transformationen und IDs für das Tracing.

Die zweite Komponente ist der *Prospective Provenance Manager*. Diese Komponente erfasst die Transformationen der Data Collection und fügt eine *Transformation ID* dem Umschlag hinzu. Anhand dieser ID kann die Transformation von einer Daten Collection verfolgt werden. Innerhalb dieser Systeme kann der Nutzer Provenance-Queries mittels der entsprechenden Query-Sprache ausführen.

Die dritte Komponente ist der *Provenance Data Server* und speichert alle Provenance-Daten innerhalb einer *Apache Cassandra*[16] Instanz ab. Mittels eines Datenkonverters ist es möglich, die Daten in *PostgreSQL*[17]-Tabellen umzuwandeln [GSM+18].

Die vierte Komponente ist ein In-Memory Dateisystem, in dem SAMbA Zwischenergebnissen von Blackbox-Transformationen ablegt werden. Als Blackbox werden hierbei Programme bezeichnet, die nicht über RDD kommunizieren, sondern über Dateien innerhalb des Dateisystems.

Die Evaluation von SAMbA besteht aus der sechsfachen Ausführung von *SciPhy*. SciPhy ist ein wissenschaftlicher Workflow in der Bioinformatik und wird für das Auffinden von optimalen Ausgangswirkstoff für Arzneimittel verwendet. Dazu wurden in der Evaluation vier Blackbox-Programme aufgerufen und die Zeit zur Erfassung von Provenance-Daten gemessen. Die Blackbox verarbeiten 193 Dateien und erstellen daraus 4531 Rohdaten. Die Größe und Format der verwendeten und generierten Dateien ist in der Evaluation [GSM+18] nicht angegeben. Die Programme werden

[16] http://cassandra.apache.org/
[17] https://www.postgresql.org/

auf einen Cluster mit 84 Kernen ausgeführt, wobei vier Kerne für das Speichern der Provenance-Daten auf Apache Cassandra reserviert sind. Zum Vergleich der Leistung wurde SAMbA mit einer unmodifizierten Apache Spark Version und einer weiteren Workflow-Lösung *SciCumulus* verglichen. Das Ergebnis der Evaluation ist das *SciCumulus* ca. 40% und die unmodifizierte Apache Spark Version ca. 1% langsamer ist als SAMbA.

### 3.2.9 Anforderungen an Provenance-Lösungen

Für die Implementierung der Konzepte in dieser Arbeit wird eine Provenance-Lösung benötigt. Eine Vielzahl von Provenance-Lösungen wurde bereits in den vorhergehenden Abschnitten beschrieben. In dieser Arbeit werden fünf Anforderungen aufgestellt, anhand derer eine Provenance-Lösung aus den beschrieben Provenance-Lösung ausgewählt wird. Die folgende Auflistung beschreibt die Anforderungen.

**Integration** Um das Nutzererlebnis konsistent zu halten, muss die Lösung das Berechnen, Speichern sowie das Anzeigen von Provenance-Daten auf einem VSDB-System ermöglichen. In dieser Arbeit wird das VSDB-System Apache Spark verwendet.

**Workflow-Typ** Für diese Arbeit wird der Workflow-Typ benötigt, um Ergebnis bis zu ihren Eingaben zurückzuverfolgen.

**Granularität** Der Workflow-Typ muss eine feine Granularität aufweisen, sodass Daten bis auf Tupelebene erfasst werden.

**Tracing** Die Provenance-Lösung muss eine Tracing-Funktionalität aufweisen, mit deren Ergebnisse bis zu ihren Eingaben verfolgt werden können. Die Implementierung der Tracing-Funktionalität ist nicht Teil dieser Arbeit.

**Verwendungszweck** Der Verwendungszweck, der Provenance-Lösung sollte soweit wie möglich generisch sein und nicht auf eine bestimmten Verwendungszweck hinzielen.

Die Tabelle 3.1 listet alle Provenance-Lösungen, die hier beschrieben wurden auf und zeigt anhand eines Hakens an, welche Anforderungen von der Provenance-Lösung erfüllt wurden.

**Tabelle 3.1:** Gegenüberstellung der Provenance-Lösungen

| Lösung | Integration | Workflow-Typ | Granularität | Tracing | Verwendungszweck |
|---|---|---|---|---|---|
| Newt [LDY13] | | | ✓ | | |
| RAMP [IPW11] | ✓ | ✓ | | | |
| Titian [IST+15] | ✓ | | ✓ | ✓ | |
| BigDebug [GIY+16] | ✓ | | ✓ | ✓ | |
| Pebble [DH19; DH20] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Lipstick[ADD+11] | ✓ | ✓ | ✓ | | |
| PROVision [ZAI19] | | ✓ | ✓ | ✓ | |
| SAMbA [GSM+18] | | ✓ | ✓ | | ✓ |

Die Analyse der ausgewählten Provenance-Lösungen hat ergeben, dass die Provenance-Lösung *Pebble* alle Anforderungen erfüllt. Pebble wurde in Apache Spark implementiert und ist ein feingranularer Workflow-Typ. Für das Tracing bietet Pebble eine Backtracking-Funktion an, mit denen Ergebnis bis zu ihren Eingabe zurückverfolgt werden können. Des weiteren ist der Verwendungszweck von Pebble generisch genug, sodass Pebble für die Implementierung der Konzepte in dieser Arbeit angewendet werden kann.

## 3.3 Indizes für Provenance-Daten

Der Einsatzzweck von VSDB ist die verteilte Berechnung von sehr großen Daten. Je nach Komplexität der Berechnung auf diesen Daten kann die Laufzeit sich über mehrere Stunden erstrecken. Dies betrifft ebenso die Erstellung von Provenance-Daten und die Abfrage. Indizes können die Abfrage durch die Speicherung der Provenance-Daten in einem speziellen Datenformat deutlich verbessern. In der Regel werden diese Indizes sortiert abgelegt und befindet sich im Hauptspeicher des Systems. Dadurch können Daten innerhalb des Indizes mittels optimierter Suchverfahren wie z. B. *Binary-Search* in $O(log(n))$ gefunden werden.

Psallidas und Wu [PW18] beschreiben *Smoke*, eine In-Memory-Datenbankengine, die Abfragezeiten von Provenance-Daten verkürzt. Die Autoren von Smoke setzen sich dabei Ziele, um diese Optimierung zu erreichen.

- Schreibeffiziente Indexstrukturen und entsprechend eine neue physische Algebra
- Informationen über Queries, die in der Zukunft auf den Provenance-Datensätzen angewandt werden sollen, werden in die Erstellung der Provenance-Daten miteinbezogen. Dadurch werden nur wirklich relevanten Provenance-Daten erzeugt
- Provenance-Daten werden, in einer für zukünftigen Queries zugeschnitten, Datenstruktur geschrieben
- Angelegt Datenstrukturen werden, wenn möglich, wiederverwendet

Smoke indiziert sogenannte Record IDs (RIDs) anstatt komplette Tupel. RIDs sind Verknüpfungen zwischen der Eingabe und Ausgabe und umgekehrt. Die RIDs werden, typisch für einen Index, sortiert abgelegt. Smoke unterstützt dabei 1-N und 1-1 Beziehungen. 1-N Beziehung werden als mehrdimensionale Indizes, während 1-1 Beziehung als einfache, eindimensionale Indizes abgebildet [PW18].

Durch diese Indizierung ermöglicht Smoke, die schnelle Abfrage von Provenance-Queries. Um dies zu ermögliche werden zusätzliche Informationen für den Index benötigt. Dies führt zu einem zusätzlichen Aufwand, der Anfragen verlangsamt. In der Evaluation wurde ein Verlangsamung der Ausgangsabfrage um 0.7x - 1.2x gemessen. Dieser initialer Aufwand ermöglicht im Gegenzug eine deutliche schneller Abfrage von Lineage-Abfrage im Millisekunden Bereich [PW18].

Durch die Indizierung von Ein- und Ausgabe mittels RIDs und die Implementierung einer physischen Algebra bietet Smoke einen interessanten Ansatz für die Beschleunigung von Provenance-Abfragen. Einige dieser Ansätze, wie z. B. die Indizierung von Eingabe- und Ausgabedaten, finden sich in dieser Arbeit wieder. Das Ziel dieser Arbeit ist aber nicht die Beschleunigung der ausgewählten

Provenance-Lösung, sondern die Neuberechnung von Analysen durch die effiziente Identifizierung von Eingabe- und Ausgabedaten, dies wird entsprechend durch eine Indizierung von Ein- und Ausgabedaten ähnlich wie bei [PW18] erreicht.

## 3.4 Partielle Neuberechnung

Die Analyse von Big Data kann ein langer Prozess sein, der über mehrere Stunden und Tage gehen kann. Bei Änderungen in den Daten muss eventuell ein langer Berechnungsprozess erneut gestartet werden. Dies ist vor allem dann problematisch, wenn sich die Eingabedaten häufig ändern. Bei geringen Änderungen kann es sinnvoller sein, eine Neuberechnung auf den geänderten Teildaten durchzuführen. Wann jedoch eine komplette oder partielle Neuberechnung durchgeführt werden soll ist von Fall zu Fall unterschiedlich und muss je nach Datensatz und dem Ausmaß der Änderungen entschieden werden.

Missier und Cala [MC19] beschreiben *ReComp*, einen Metaprozess, der die Neuberechnung von Ergebnissen vermeidet, indem die Auswirkung von Änderungen der Eingabedaten auf das Ergebnis evaluiert wird. Dazu werden die Eingabedaten beobachtet und bei Änderungen, die Auswirkung berechnet. Falls die Auswirkung eine Neuberechnung rechtfertigt, wird das Ergebnis mit den neuen Daten berechnet, falls die Auswirkung nur geringfügig sind wird das bestehende Ergebnis beibehalten. Wie die Auswirkung berechnet wird, hängt vom Kontext der Domäne ab. Für Domänen bei dem selbst kleinste Änderungen in den Eingabedaten zwangsweise zu einem neuem Ergebnis führen, also die Auswirkung sehr hoch sind, kann *ReComp* die Neuberechnung zwar automatisieren, aber keine Laufzeit einsparen. Implementiert ist *ReComp* als ein verteiltes System, das die entsprechenden Prozesse mittels Hypertext Transfer Protocol (HTTP) aufruft. Dazu muss der Nutzer Schnittstellen implementieren, die die Unterschiede der alten und neuen Daten berechnet, die Auswirkung der Unterschiede evaluiert und falls nötig den Prozess für die Berechnung startet. Diese drei Schnittstellen laufen im *ReComp-Loop-Service*. Dieser Service ist die zentrale Koordinierungsschnittstelle für die Neuberechnung von Prozessen bzw. Daten. Bei einem komplexen Prozess mit mehreren Zwischenergebnissen können mehrere solcher Schnittstellen für die Unterschieds- und Auswirkungsberechnung angegeben werden. Der Loop-Service kann anhand von gesammelten Provenance-Daten entsprechend die Zwischenprozesse aufrufen und die Auswirkung berechnen lassen.

Durch den generischen Ansatz müssen Black-Box-Programme so angepasst werden, das diese über HTTP-Schnittstellen aufgerufen werden können. Des weiteren müssen diese Programme Provenance-Daten über ihre Eingabedaten und Ausgabedaten sammeln, da anhand dieser Provenance-Informationen der Loop-Service, die Berechnung der Daten durchführt und die nächsten Prozesse aufruft [MC19].

Um die Funktionalität von *ReComp* zu überprüfen haben Missier und Cala [MC19] zwei Fallstudien durchgeführt. Die erste Fallstudie befasst sich mit einem Simulationswerkzeug für die Flutmodellierung und ist eine ausführbare Datei und ein Black-Box-Prozess. Ein Wrapper wurde geschrieben, der in der Lage ist die Daten von *ReComp* herunterzuladen und Provenance-Daten in *ReComp* hochzuladen. Insgesamt konnten 30% bis 60% Neuberechnungen eingespart werden. Die zweite Fallstudie handelt von einem Gray-Box-Prozess, der die Pathogenität der genetischen Variationen der Patienten klassifiziert. Bei diesem Prozess handelt es sich um einen Workflow, welcher mehrere Datenquellen und Zwischenprozesse für die Klassifizierung verwendet. Für diesen Prozess musste

kein Wrapper implementiert werden, da der Prozess selbst modifiziert werden konnte, um eine direkte Kommunikation mit *ReComp* zu ermöglichen. Das Ergebnis ist eine 90% Einsparung von Neuberechnungen.

## 3.5 Aktualisieren von Views

Als *View* wird eine logische Relation in einem Datenbanksystem bezeichnet [CL11; Wie91]. Sie dienen ebenso als Sichten auf Daten, die durch Queries definiert werden, mit dem Vorteil, dass sie wie Datenbanktabellen agieren auf denen weitere Queries angewendet werden können. Somit ist auch die Einschränkung und Vereinfachung vom Datenmodell möglich, da sie nur die Daten beinhalten, die durch die Queries vorgegeben werden.

View können wie normale Datenbanktabellen aktualisiert werden. Die Problematik hierbei ist die Vermeidung von Seiteneffekten bei Aktualisierung der entsprechenden Eingabedaten. Eine Aktualisierung ist dann seiteneffektfrei, wenn sie nur die Eingabedaten aktualisiert, die von der Aktualisierung betroffen sind [BKT02]. Dieses Problem ist als *View-Update-Problem* bekannt und wurde bereits in verschiedenen wissenschaftlichen Arbeiten behandelt [BKT02; BS81; CL11; FG12; KSV06; UC92]

Buneman et al. [BKT02] beschreiben in ihrer Arbeit die Komplexität des View-Update Problems für zwei verschiedene Anwendungsfälle. Der erste Fall befasst sich mit dem Löschen eines Ergebnis aus der View und wie sich das Löschen auf die Eingabedaten auswirkt. Dabei soll entsprechend des View-Update Problems entschieden werden, ob eine Löschoperation seiteneffektfrei durchgeführt werden kann, d. h. das Löschen von Eingabedaten hat nur Auswirkung auf das zu löschende Ergebnis. Dies bezeichnen Buneman et al. [BKT02] als *Delete Minimization*. Das Ergebnis der Arbeit ist, dass es NP-schwer zu entscheiden ist, ob *Projektion* und *Join* Queries seiteneffektfrei sind. Dies gilt auch für Join-Queries die anstatt eines Projektion Ausdrucks ein *Union* beinhalten. Projektion und Union Queries, sowie Join Queries sind in der Komplexitätsklasse P und sind somit in Polynomialzeit lösbar.

Der zweite Fall der Arbeit befasst sich mit dem Anreichern der View durch Annotationen. Dies bezeichnen Buneman et al. [BKT02] als das *Annotation Placement Problem* und weist eine Verbindung zu *Provenance* (vgl. 2.4) auf. Annotationen sind in Provenance-Systemen wichtig, da sie die Rückverfolgung vom Ergebnis zur Eingabe ermöglichen. In diesem Fall wurde untersucht, in welcher Komplexitätsklasse sich die Entscheidung befindet, ob es eine seiteneffektfrei Propagation der Annotationen von der Eingabe zu dem annotierenden Ergebnis gibt. Queries, die mit einer Projektion und einem Join ausgeführt werden sind NP-hart zu entscheiden. Im Gegensatz zur Delete Minimization sind Join-Union und Projektion-Union Queries in der P-Klasse und somit in Polynomialzeit lösbar.

Die vorliegende Arbeit befasst sich hauptsächlich mit dem Aktualisieren von Ergebnissen bei Änderungen von Eingabedaten unter der Verwendung von Provenance und unter Beachtung von gegebenen Einwilligungen. Neuberechnungen von Ergebnissen, die durch mehrere Eingabedaten erzeugt werden, werden durch die Provenance-Lösung auf ein absolutes Minimum reduziert, so dass diese Änderungen bzw. Aktualisierungen effektiv seiteneffektfrei sind. Annotation werden durch die Provenance-Lösung entsprechend zu den Ergebnissen propagiert. Das Hinzufügen oder Löschen von Annotation an Ergebnisse ist nicht Teil dieser Arbeit.

# 4 DS-GVO konformes Datenmodell und Ausführung

Dieses Kapitel befasst sich mit einem Konzept für die Datenmodellierung und Verarbeitung unter Berücksichtigung der DS-GVO. Des weiteren wird eine Indexstruktur entwickelt, die eine effiziente Neuberechnung ermöglichen soll.

## 4.1 DS-GVO konformes Datenmodell

Die DS-GVO erlaubt Verbrauchern ihre Einwilligung zu personenbezogenen Analysen jederzeit zu widerrufen. Um zu gewährleisten, das Analyseergebnisse von personenbezogenen Abfragen konform zur DS-GVO sind, müssen die gegebenen bzw. entzogenen Einwilligungen in der Berechnung berücksichtigt werden. Dazu wird ein Datenmodell benötigt, welches es ermöglicht Eingabedaten mit Einwilligungen zu erweitern. In dieser Arbeit wird das relationale Datenmodell aus Abschnitt 2.5 so erweitert, dass die DS-GVO konforme Datenverarbeitung ermöglicht wird. Des weiteren werden Einwilligungen nur für den Zweck der personenbezogenen Analyse in Betracht gezogen. Andere Zwecke, wie etwa die Weitergabe von personenbezogene Daten oder die Kontaktaufnahme mit potentiellen Kunden können jedoch ebenso mit dem Modell abgebildet werden.

Eine Einwilligung muss zwei Eigenschaften erfüllen. Die erste Eigenschaft ist, dass die Einwilligung einem Zweck zugeordnet ist. Der Zweck ist der Zweck der personenbezogenen Analyse. Die zweite Eigenschaft ist der Status einer Einwilligung und kann insgesamt zwei Werte annehmen. Ein Verbraucher kann einer Verarbeitung seiner personenbezogenen Daten zustimmen oder sie entziehen bzw. vorneherein ablehnen. Dies lässt sich durch einen booleschen Wert darstellen. Dabei wird eine Einwilligung als *TRUE* gewertet, wenn der Verbraucher der Verarbeitung zugestimmt hat. Falls der Verbraucher einer Verarbeitung nicht zugestimmt hat, wird die Einwilligung als *FALSE* gewertet. Anhand dieses Wertes kann entschieden werden, welche Daten für die Verarbeitung verwendet werden. Dies lässt sich wie folgt definieren.

**Definition 4.1.1 (Einwilligung)**
*Sei der Zweck einer personenbezogenen Analyse $Q$ definiert als Z. Eine Einwilligung E ist immer an einen Zweck Z gebunden und ist von Typ $E \in \{TRUE, FALSE\}$. Die Menge aller Einwilligungen sei definiert als $\mathcal{E} = \{E_1, E_2, \ldots, E_n\}$.*

Anhand dieser Einwilligungen lässt sich eine Relation definieren, die die Einwilligung auf eine Eingaberelation widerspiegelt. Dies lässt sich wie folgt definieren.

**Definition 4.1.2 (DS-GVO konformes Datenmodell)**
*Sei eine Relation $\mathcal{R}$ und Einwilligungen $\mathcal{E}' \subset \mathcal{E}$ gegeben. Eine Relation, die die Einwilligung für eine Eingaberelation beinhaltet wird definiert als $K = \mathcal{R} \times \mathcal{E}'$. Das Schema von K ist definiert als $K = (\mathcal{A}_{id}, \mathcal{A}_{E_1}, \ldots, \mathcal{A}_{E_n})$, wobei $\mathcal{A}_{id}$ den Fremdschlüssel als Referenz zu $\mathcal{R}$ und $\mathcal{A}_{E_n}$ die Einwilligungen in $\mathcal{E}'$ abbildet.*

*$\mathcal{R}$ ist genau dann konform zur DS-GVO, wenn gilt $\forall k \in K | k.E_i = TRUE$ mit $0 \leq i \leq n$ wobei $n = |\mathcal{E}'|$.*

Die Einwilligungen werden, dem relationalen Datenmodell entsprechend, in Relationen abgebildet. Einwilligungen werden dabei auf der Datenebene von Tupeln vergeben. Die Referenzierung der Einwilligung auf das jeweilige Tupel muss dabei durch ein eindeutiges Kennzeichen erfolgen. Dazu bieten Datenbanken und das relationale Datenmodell sogenannte Primärschlüssel an [EN15]. In dieser Arbeit wird davon ausgegangen, dass jede Eingaberelation über solch einen Primärschlüssel verfügt. Dieser Primärschlüssel ist notwendig, um eine eindeutige Referenzierung der Einwilligung auf das entsprechende Eingabetupel herzustellen.

Die Architektur der Relationen und die Beziehung der Einwilligung $K$ zur Eingaberelation $\mathcal{R}$ ist in Abbildung 4.1 dargestellt. $\mathcal{R}$ verfügt über einen eindeutigen Primärschlüssel. $K$ referenziert den Primärschlüssels der Eingaberelation $\mathcal{R}$ mittels des Fremdschlüssel *id*. Des weiteren werden *Consent*-Attribute benötigt, welche den booleschen Datentyp verwendet. Die Beziehung zwischen den Relationen ist dabei eine 1:1 Beziehung, dies ist durch die Querstriche in der Verbindungslinie dargestellt.

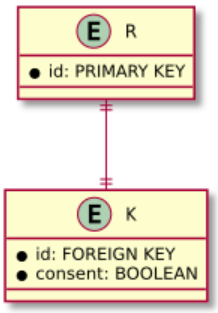
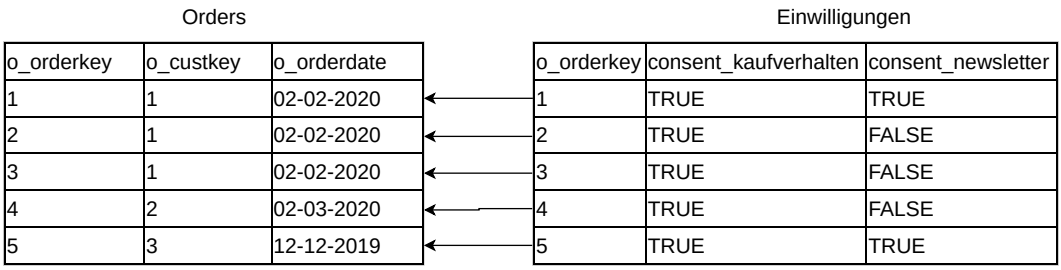



**Abbildung 4.1:** ER-Diagramm des Datenmodells

Die Abbildung 4.2 zeigt die *Orders*-Tabelle des Szenarios mit dem DS-GVO konformen Datenmodell. Diese enthält keine direkten Einwilligungen, da sich diese in der *Einwilligungen*-Relation befinden. Initial haben alle Kunden ihre Einwilligung für die Verarbeitung der Daten für Analyse 1.2 gegeben. Dies ist durch den *TRUE*-Wert im Attribut *consent_kaufverhalten* abgebildet. Das Attribut *o_orderkey* ist die eindeutige Assoziation zwischen den beiden Tabellen. Anhand dieses Attributes wird eine Referenzierung der Einwilligungen-Relation zur Orders-Relation hergestellt. Um die beliebige Erweiterbarkeit des Modell anhand des Beispiels zu demonstrieren, wird eine zusätzliche

Einwilligung für den Zweck *consent_newsletter* eingefügt. Dieser Zweck dient zur Erfassung aller Kunden, die einem Newsletter in den Empfang eingewilligt haben und kann verwendet werden um Kunden einen zugeschnittenen Newsletter anhand ihrer Bestellung zuzuschicken.




| o_orderkey | o_custkey | o_orderdate |
|---|---|---|
| 1 | 1 | 02-02-2020 |
| 2 | 1 | 02-02-2020 |
| 3 | 1 | 02-02-2020 |
| 4 | 2 | 02-03-2020 |
| 5 | 3 | 12-12-2019 |

| o_orderkey | consent_kaufverhalten | consent_newsletter |
|---|---|---|
| 1 | TRUE | TRUE |
| 2 | TRUE | FALSE |
| 3 | TRUE | FALSE |
| 4 | TRUE | FALSE |
| 5 | TRUE | TRUE |

**Abbildung 4.2:** Einwilligungen-Tabelle für Bestellungen

Das Modell ist beliebig um weitere Einwilligungen erweiterbar. Dazu müssen die benötigten Zwecke definiert und einer Einwilligung zugeordnet werden. Die Einwilligung kann dann in einer Einwilligungsrelation abgebildet werden. Dadurch ist das Modell in der Lage für eine Anfrage mehrere Einwilligungen abzubilden. Mittels diesem Modell lassen sich nun DS-GVO konforme personenbezogene Anfragen definieren, die im nächsten Abschnitt beschrieben werden.

## 4.2 DS-GVO konforme personenbezogene Anfragen

Um die Daten entsprechend DS-GVO konform zu verarbeiten, müssen personenbezogene Analyse in der Lage sein, Daten mittels des Datenmodells DS-GVO konform zu berechnen. Dazu muss die personenbezogene Anfrage, die entsprechende Einwilligung bzw. das DS-GVO konforme Datenmodell verwenden, damit das daraus resultierende Ergebnis ebenso konform zur DS-GVO ist.

Die personenbezogene Anfrage, die anhand von Einwilligungen ein DS-GVO konformes Ergebnis produziert lässt sich wie folgt definieren.

**Definition 4.2.1 (Personenbezogene Anfrage mit Einwilligung)**
*Sei eine personenbezogene Anfrage $\mathcal{Q}$ und eine Menge von $\mathcal{E}' \subset \mathcal{E}$. Eine personenbezogene Anfrage $\mathcal{Q}'$, die unter Berücksichtigung der gegeben Einwilligung $\mathcal{V}$ berechnet, sei definiert als* $\mathcal{Q}' =< \mathcal{Q}, \mathcal{E}' >$.

Anhand der Erweiterung der personenbezogenen Anfrage $\mathcal{Q}'$ mit Einwilligungen $\mathcal{E}'$ ist die Anfrage in der Lage, die Eingabedaten DS-GVO konform zu berechnen. Der Algorithmus zur Berechnung lässt sich demnach wie folgt definieren.

**Input:** Eingaberelation $\mathcal{R}$
**Input:** Personenbezogene Anfrage $Q'$
**Input:** Einwilligung E
**Result:** DSGVO konforme Daten $\mathcal{R}'$

```
1 R' ← ∅;
2 if E ∈ Q'.ℰ then
3 |   K ← R × E;
4 |   R' ← R ⋈_{R_{A_id} = K_{A_id}} K;
5 |   R' ← Q'(R');
6 else
7 |   R' ← Q'(R);
8 return R';
```

**Algorithmus 4.1:** DS-GVO konforme Berechnung

Der Algorithmus erwartet als Eingabe eine Eingaberelation, eine personenbezogene Anfrage sowie eine Einwilligung. Der Algorithmus prüft zunächst, ob die personenbezogene Anfrage, die Einwilligung für die Berechnung verwendet. Falls dies nicht der Fall ist, wird die Berechnung ohne die Einwilligung durchgeführt, andernfalls wird die Einwilligung auf die Eingaberelation angewandt. Das Ergebnis ist die Einwilligungsrelation K in Zeile 3. Mittels K werden dann in Zeile 4 nur die diejenigen Tupel aus $\mathcal{R}$ ausgewählt, für die eine Einwilligung gegeben wurde. Das Zwischenergebnis wird dann von der personenbezogenen Anfrage $Q'$ in Zeile 5 verwendet. Das Ergebnis ist somit konform zur DS-GVO.

Mittels dem Algorithmus und der Einwilligungsrelation aus Abbildung 4.2 lässt sich die personenbezogene Analyse 1.2 aus *Szenario* 1.2 für die Einwilligung $consent\_kaufverhalten$ berechnen. Das Ergebnis der personenbezogenen Analyse $Q'$ ist in Tabelle 4.1 dargestellt. In diesem Ergebnis wurde die Bestellung $o\_orderkey = 1$ nicht verwendet, da dafür keine Einwilligung gegeben wurde.

**Tabelle 4.1:** DS-GVO konformes Ergebnis

| c_name | c_address | c_phone | sum(l_price) | l_category |
|---|---|---|---|---|
| Bob | 9 Hermina Crossing | 555-1234 | 350,00 | Haushalt |
| Bob | 9 Hermina Crossing | 555-1234 | 30,00 | Mode |
| Alice | 6 Katie Road | 555-1337 | 700,00 | Elektronik |

Damit ein Kunde die Rechte nutzen kann, die durch die DS-GVO gewährt werden, wird in Abschnitt 4.3 eine Möglichkeit zum Einwilligungsentzug erläutert.

## 4.3 Einwilligungsentzug

Ein Verbraucher hat dank der DS-GVO das Recht seine Einwilligung jederzeit zu widerrufen. Die betroffenen Eingabedaten dürfen dann nicht mehr in personenbezogenen Analysen verwendet werden. Die Änderung einer Einwilligung für eine Eingaberelation $\mathcal{R}$ kann mittels des DS-GVO konformen Datenmodell durch die Änderung des Status des jeweiligen Einwilligungsattributes in der Einwilligungsrelation $K$ vollzogen werden. Einwilligungsentzüge sind dabei wichtiger, als die Zustimmung der Einwilligung. Dies ist dadurch begründet, da Ergebnis von personenbezogenen Analysen einen Rechtsverstoß darstellen, wenn entzogenen Einwilligungen in den Ergebnis auftauchen. Deshalb wird in dieser Arbeit hauptsächlich der Einwilligungsentzug mittels des Datenmodells abgebildet. Der Algorithmus für den Entzug lässt sich wie folgt definieren.

**Input:** Einwilligungsrelation $K$
**Input:** Einwilligung $E \in \mathcal{E}$
**Input:** Geänderte Einwilligung $id \in \mathcal{R}$
**Result:** Aktualisiert Einwilligungsrelation $K$
1 **if** $K \neq \emptyset \wedge id \in K \wedge E \in K$ **then**
2 $\quad t \leftarrow \sigma_{\mathcal{A}_{id}=id} K$;
3 $\quad t.\mathcal{A}_E = FALSE$;
4 return $K$

**Algorithmus 4.2:** Entzug einer Einwilligung

Der Algorithmus 4.2 erwartet als Eingabe eine Einwilligungsrelation K, eine Einwilligung E und den $id$ Primärschlüssel der Eingaberelation, der zugleich der Fremdschlüssel der Einwilligungsrelation K ist. Der Algorithmus prüft zunächst in Zeile 1, ob K über Elemente bzw. Tupel verfügt, ob die $id$ sich in K befindet und ob K die Einwilligung E abbildet. Falls dies der Fall ist, wird das entsprechende Tupel von K mittels der $id$ selektiert und das Einwilligungsattribut in Zeile 3 auf *FALSE* gesetzt. Der nächste Abschnitt zeigt die Funktionsweise des Algorithmus anhand des *Szenario* 1.2 .

Die Änderung einer Einwilligung bzw. der Entzug lässt sich in der *Orders*-Tabelle wie folgt beschreiben. Im *Szenario* 1.2 hat Bob seine Einwilligung für die Bestellung $o\_orderkey = 1$ entzogen. Die führt dazu, das die entsprechende Einwilligungsrelation mit dem Fremdschlüssel 1 aktualisiert werden muss. Dies geschieht durch den Algorithmus 4.2. Abbildung 4.3 stellt die aktualisierte Einwilligungstabelle für die *Orders*-Tabelle dar. Für das Tupel $o\_orderkey = 1$ wurde die Einwilligung entsprechend auf FALSE gesetzt.

Orders

| o_orderkey | o_custkey | o_orderdate |
|---|---|---|
| 1 | 1 | 02-02-2020 |
| 2 | 1 | 02-02-2020 |
| 3 | 1 | 02-02-2020 |
| 4 | 2 | 02-03-2020 |
| 5 | 3 | 12-12-2019 |

Einwilligungen

| o_orderkey | consent_kaufverhalten | consent_newsletter |
|---|---|---|
| 1 | ***FALSE*** | TRUE |
| 2 | TRUE | FALSE |
| 3 | TRUE | FALSE |
| 4 | TRUE | FALSE |
| 5 | TRUE | TRUE |

**Abbildung 4.3:** Aktualisierte Einwilligungen-Tabelle für Bestellungen

# 5 DS-GVO konforme Analyse mittels geeigneter Indexstruktur

In diesem Kapitel wird die DS-GVO konforme personenbezogene Analyse mittels einer Indexstruktur bzw. eines Index vorgestellt. Dieser Abschnitt besteht aus zwei Teilen, aus der *Initialen Berechnung* und der *indexgestützten Neuberechnung*. Zuerst wird die Indexerzeugung erläutert und anschließend die indexgestützte Berechnung. Abbildung 5.1 zeigt den vollständigen Ablauf.

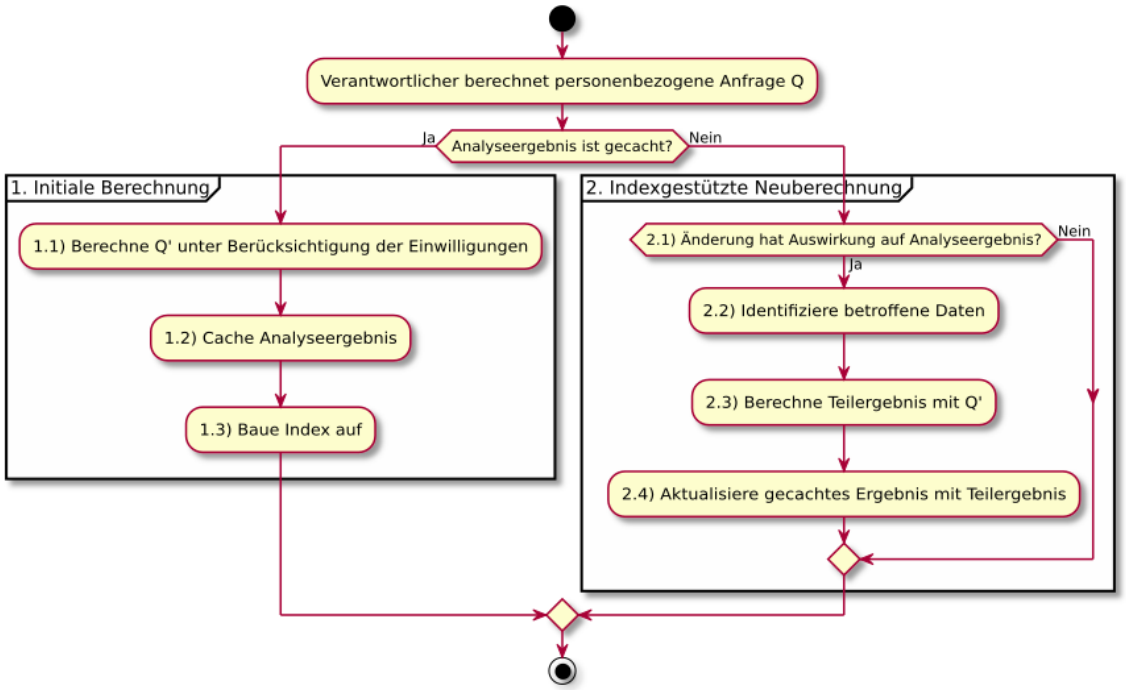



**Abbildung 5.1:** Ablauf der DS-GVO konformen Datenverarbeitung und indexgestützten Neuberechnung

Der Ablauf beginnt in dem ein Verantwortlicher eine personenbezogene Anfrage $Q$ durchführt. Als Verantwortlicher sind hierbei juristische Personen gemeint, die über personenbezogene Daten verfügen und daraus Ergebnisse berechnen. Das daraus resultierende Ergebnis kann entweder bereits existieren d. h. gecached sein oder wird erst durch die Ausführung von $Q$ erzeugt. Zunächst gehen wir davon aus, dass das Ergebnis noch nicht gecached wurde d. h. das Ergebnis muss initial berechnet werden (Schritt 1. Initiale Berechnung). Die Berechnung passiert dabei unter der Berücksichtigung der Einwilligungen mittels des Datenmodells aus Kapitel 4. Das DS-GVO konforme Ergebnis

wird daraufhin gecached. Das gecachte Ergebnis kann dabei als View entweder im Hauptspeicher gehalten oder als separate View in einer Datenbank gespeichert werden. Anhand dieses Ergebnisses wird nun der Index aufgebaut, der eine potentielle Neuberechnung ermöglichen soll.

Einwilligungsentzüge führen zu einer vollständigen Neuberechnung des Ergebnissen, da das gecachte Ergebnis einen Rechtsverstoß im Sinne der DS-GVO darstellt. Für komplexe Anfragen oder sehr große Daten ist eine vollständige Neuberechnung möglicherweise jedoch nicht notwendig, vor allem dann nicht, wenn Einwilligungsentzüge nicht häufig durchgeführt werden. Solche nachfolgende Berechnungen, auf Grund von Einwilligungsentzügen, können nun mittels des aufgebauten Index behandelt werden (Schritt 2: Indexgestützte Neuberechnung). Dazu wird zunächst geprüft, ob die Änderung eine Auswirkung auf das Ergebnis hat. Falls ja, werden die betroffenen Ergebnisse und die betroffene Eingabetupel identifiziert. Falls nicht, muss keine Neuberechnung durchgeführt werden. Das ist z. B. dann der Fall, wenn das betroffene Eingabetupel durch eine Filter-Operation nicht im Ergebnis verwendet wird. Nachdem alle betroffenen Daten identifiziert wurden, werden die Teildaten mittels der personenbezogenen Abfrage $Q'$ neu berechnet. Die Berechnung geschieht, dabei unter der Beachtung der Einwilligung wie in Kapitel 4 beschrieben. Nach der indexgestützten Neuberechnung kann das Ergebnis mit dem Teilergebnis aktualisiert werden. Diese Teilberechnung kann potentiell schneller sein als eine Berechnung über den vollständigen Datensatz.

Die nächsten Abschnitte erläutern die Schritte 1 und 2 aus der Abbildung 5.1 im Detail.

## 5.1 Initiale Berechnung

Der Schritt *1. Initiale Berechnung* aus Abbildung 5.1 beschreibt den Ablauf der erstmaligen Berechnung einer personenbezogenen Analyse und teilt sich in drei Teile auf. Die Nummern referenzieren hierbei die jeweiligen Schritte in Abbildung 5.1.

**1.1** Die initiale Berechnung muss DS-GVO konform sein, damit keine Rechtsverstöße auftreten. Die DS-GVO Konformität wird dabei durch die Verwendung des DS-GVO Modells aus Kapitel 4 ermöglicht.

**1.2** Das Ergebnis der Berechnung wird danach gecached d. h. es wird entweder im Hautspeicher gehalten oder auf die Festplatte geschrieben. Damit der vollständige Ablauf 5.1 bei einer Einwilligungsänderung nicht erneut durchgeführt werden muss, wird ein Index aufgebaut, der in nachfolgenden Berechnung verwendet werden kann.

**1.3** Anhand der Ergebnisdaten wird ein Index aufgebaut. Ziel des Index ist die effiziente Identifikation von Ergebnistupel und die damit verbundenen Ergebnistupel, die durch einen Einwilligungsentzug erneut berechnet werden müssen. Im nächsten Abschnitt 5.1.1 wird eine detaillierte Übersicht über den Aufbau des Index bzw. der Indexstruktur gegeben.

### 5.1.1 Aufbau des Index

Unter der Verwendung einer Provenance-Lösung (vgl. Abschnitt 3.2) werden Provenance-Daten als Grundlagen für den Aufbau des Index verwendet. Bei den Provenance-Daten handelt es sich, um Assoziationen zwischen dem Ergebnis einer personenbezogenen Analyse und den verwendeten Eingabedaten. Der Index soll die potentielle Neuberechnung dadurch ermöglichen, indem nur ein

Teil der Daten neu berechnet wird. Da eine personenbezogene Analyse in der Regel aus mehreren Eingaberelationen besteht wird für jede Relation ein separater Index benötigt. Die Assoziation zwischen den einzelnen Indizes, und somit zwischen den Eingabedaten, ist durch das Ergebnis implizit gegeben. Somit lässt sich der Index wie folgt definieren.

**Definition 5.1.1 (Index)**
*Sei Eingaberelation $\mathcal{R}$ gegeben, sowie eine Ergebnisrelation $\mathcal{V}$. Ein Index $\mathcal{I}_\mathcal{R}$ ist eine Relation mit zwei Attributen $\mathcal{A}_{in}$ und $\mathcal{A}_{out}$ definiert als $\mathcal{I}_\mathcal{R} = (\mathcal{A}_{in}, \mathcal{A}_{out})$. $\mathcal{A}_{in}$ und $\mathcal{A}_{out}$ sind Attribute für die bidirektionale Assoziation zwischen den Tupel in der Eingaberelation $\mathcal{R}$ und den Tupel in der Ergebnisrelation $\mathcal{V}$.*

Der Algorithmus 5.1 erzeugt den Index und erwartet eine Menge von Eingaberelationen und eine Ergebnisrelation. In Zeile 1 werden über alle Ergebnistupel der Ergebnisrelation iteriert. Diese Tupel werden für den Aufbau des Index benötigt. Anhand des Ergebnistupel und einer Eingaberelation $\mathcal{R}$ kann die Provenance-Lösung alle Eingabetupel aus $\mathcal{R}$ berechnen, die zu $v \in \mathcal{V}$ geführt haben. Die Provenance-Lösung bietet dazu eine $backtrack$-Funktion an [DH20]. Sie ermöglicht, das jedes Ergebnistupel $v \in \mathcal{V}$, über alle Zwischenoperationen, mit ihren jeweiligen Eingabetupeln $t \in \mathcal{R}$ assoziiert wird. Das Ergebnis der Backtracking ist ein Teilmenge der Eingaberelation.

**Input:** Ergebnisrelation $\mathcal{V}$
**Input:** Eingaberelationen $\mathcal{A} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$
**Result:** Index $\mathcal{I}$
1 **for** $v \in \mathcal{V}$ **do**
2  **for** $\mathcal{R} \in \mathcal{A}$ **do**
3  $\mathcal{I}^v_\mathcal{R} \leftarrow backtrack(v, \mathcal{R})$

**Algorithmus 5.1:** Aufbau des Index

Die Architektur des Index ist als ER-Diagramm in der Abbildung 5.2 dargestellt. Der Index lässt sich als Relation mit den Attributen *IN* und *OUT* darstellen. Diese Attribute dienen dabei als Assoziationen zwischen der Eingaberelation $\mathcal{R}$ und der Ergebnisrelation $\mathcal{V}$. Beide Relationen verfügen, dabei über jeweils einen eindeutigen Primärschlüssel, der für die Assoziation verwenden wird. Ein Index verweist dabei mittels zweier Fremdschlüssel auf die Tupel in einer Eingabe- und Ergebnisrelation.

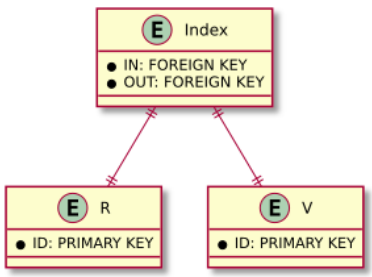



**Abbildung 5.2:** ER-Diagramm: Index

Für das *Szenario* 1.2 lässt sich somit jeweils ein Index pro Eingabetabelle erstellt. Um die Assoziation zwischen Eingaberelation und Ergebnisrelation zu ermöglichen, benötigt Provenance eine eindeutige Identifikation. Für die Eingaberelation wird hierbei der existierende Primärschlüssel verwendet. Generell benötigen Ergebnisse aus personenbezogenen Analysen keinen Primärschlüssel, da sie das Ergebnis aus Eingaberelationen sind und somit keinem vordefinierten Schema folgen. Dementsprechend wird das DS-GVO konforme Analyseergebnis 5.1 aus Schritt 1.1 mit einem eindeutigen Identifier $\mathcal{A}_{OUT}$ erweitert.

**Tabelle 5.1:** DS-GVO konforme Instanz der Analyse

| c_name | c_address | c_phone | sum(l_price) | l_category | **OUT** |
|---|---|---|---|---|---|
| Bob | 9 Hermina Crossing | 555-1234 | 850,00 | Haushalt | **A** |
| Bob | 9 Hermina Crossing | 555-1234 | 50,00 | Automobil | **B** |
| Bob | 9 Hermina Crossing | 555-1234 | 30,00 | Mode | **C** |
| Alice | 6 Katie Road | 555-1337 | 700,00 | Elektronik | **D** |

Die Indizes für die Eingabetabellen sind jeweils in den Abbildungen 5.3, 5.4 und 5.5 dargestellt. Aus Übersichtszwecken werden nur ausgewählte Attribute, der jeweiligen Tabellen dargestellt. Es ist zu beachten, das nicht alle Tupel aus den Eingaberelationen zu einem Ergebnis beitragen, da nicht alle Verbraucher, wie etwa David, in die Analyse miteinbezogen werden, da sie das Filterkriterium der Analyse nicht erfüllen.

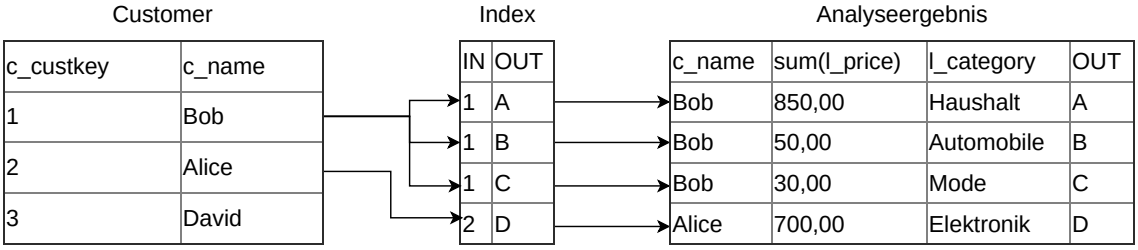



**Abbildung 5.3:** Index der Customer-Tabelle

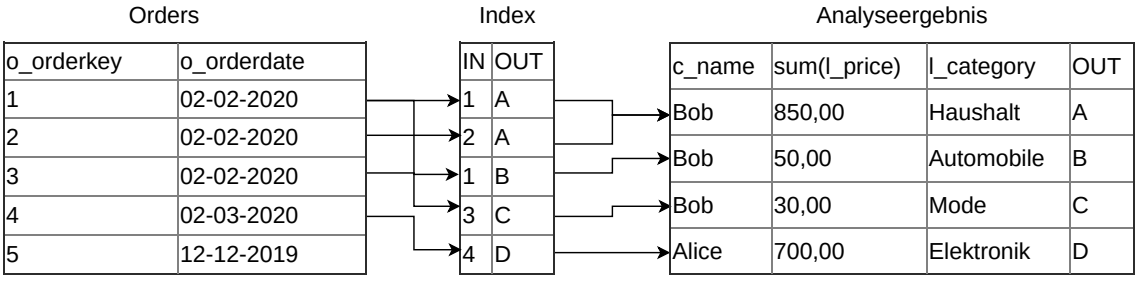



**Abbildung 5.4:** Index der Orders-Tabelle

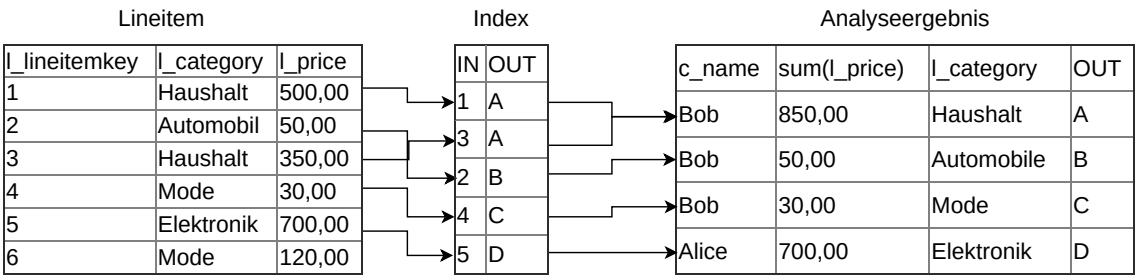



**Abbildung 5.5:** Index der Lineitem-Tabelle

Durch diese Indizes kann eine nachfolgende Neuberechnung bei Entzug von Einwilligungen potentiell schneller sein. Abschnitt 5.2 beschreibt die indexgestützte Neuberechnung.

## 5.2 Indexgestützte Neuberechnung

Nachdem der Index erstellt wurde, kann dieser genutzt werden, um Neuberechnung potentiell effizienter durchzuführen. Dazu wird aus Abbildung 5.1 der Ablauf *2. Indexgestützte Neuberechnung* in diesem Abschnitt erläutert. Dieser Ablauf besteht aus insgesamt vier Schritten, die die folgenden Funktionen erfüllen.

**2.1** Zunächst wird geprüft, ob eine Änderung zu einer Neuberechnung führt. Falls dies nicht der Fall ist, muss keine Neuberechnung durchgeführt werden, ansonsten wird in den Schritt 2.2 übergegangen.

**2.2** Für die indexgestützte Neuberechnung werden in diesem Schritt alle betroffenen Eingabetupel mittels des Index identifiziert.

**2.3** Anhand der identifizierten Daten werden mittels der personenbezogenen Analyse $Q'$ Teilergebnis berechnet.

**2.4** Die berechneten Teilergebnisse werden mit dem bereits vorhandenen Ergebnis zusammengeführt.

Durch die Verwendung des vorgestellten Index können Neuberechnungen auf Teildaten durchgeführt werden. Dadurch können Änderungen potentiell effizienter neu berechnet werden, als die Berechnung auf einem vollständigen Datensatz. Die Aufgabe des Index ist dabei die effiziente Identifikation der betroffenen Eingabedaten bzw. Eingabetupel. Der Schritt 2.2 lässt sich in insgesamt zwei Unterschritte einteilen. Der erste Schritt ist die Identifikation des betroffenen Ergebnis. Daraus lässt sich im zweiten Schritt ableiten, welche Eingabetupel zum Ergebnis beigetragen haben und somit für die Neuberechnung in Frage kommen. Diese zwei Schritte werden in den folgenden Abschnitten genauer erläutert.

### 5.2.1 Schritt I: Identifikation der Ergebnistupel

Im ersten Schritt müssen zunächst alle Ergebnisse identifiziert werden, die durch einen Einwilligungsentzug betroffen sind. Dies ist notwendig, da eine Entzug in den Eingabedaten zu einer potentiellen Rechtsverstoß im Ergebnis führt. Durch die Identifikation des Ergebnistupels können im nächsten Schritt alle betroffenen Eingabetupel identifiziert werden. Der Ablauf der Identifikation ist in Abbildung 5.6 dargestellt. Bei Änderungen in einer Eingaberelation bzw. einem Einwilligungsentzug muss zunächst geprüft werden, ob die Relation $\mathcal{R}$ auch in der jeweiligen Analyse $Q$ verwendet wird. Falls dies der Fall ist, müssen die geänderten Tupel $r \in \mathcal{R}$ extrahiert werden, ansonsten ist keine Neuberechnung notwendig, da die Eingaberelation $\mathcal{R}$ nicht in der Analyse $Q$ verwendet wird und somit zu keinem Ergebnis in $\mathcal{V}$ beiträgt. Anschließend muss geprüft werden, ob die geänderten Tupel $r \in \mathcal{R}$ zum Ergebnis beitragen. Falls dies der Fall ist können mittels des Index alle Ergebnistupel $v \in \mathcal{V}$ gefunden werden, ansonsten ist kein Neuberechnung notwendig, da die getroffenen Änderungen keine Auswirkung auf die Ergebnistupel $v \in \mathcal{V}$ haben. Die Identifikation der Ergebnistupel ist damit abgeschlossen.

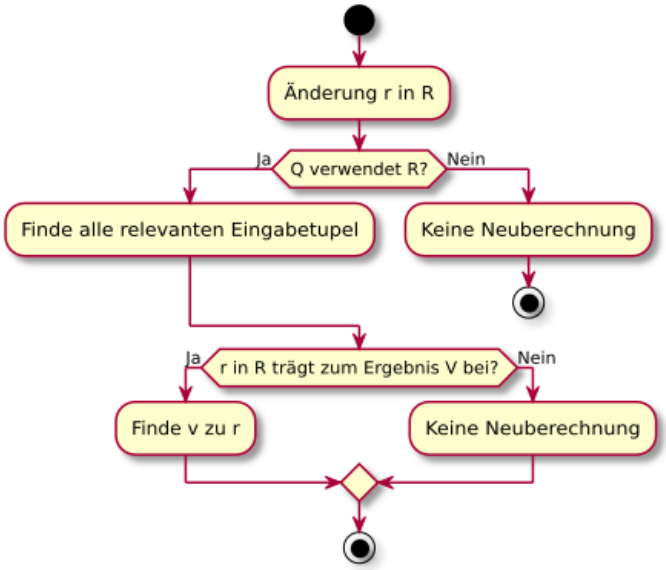



**Abbildung 5.6:** Identifikation von Ergebnistupeln

Der Algorithmus 5.2 berechnet alle betroffenen Ergebnistupel anhand des geänderten Einwilligungsentzug. Der Algorithmus erwartet als Eingabe eine personenbezogene Analyse $Q'$ und ein geändertes Tupel aus der Einwilligungsrelation K und alle Eingaberelationen die $Q'$ verwendet. Zunächst wird in Zeile 1 geprüft, ob die entzogene Einwilligung $k \in K$ in $Q'$ verwendet wird. Falls das nicht der Fall ist, muss $Q'$ nicht neu berechnet werden und es wird eine leere Menge zurückgeliefert. Ansonsten wird in Zeile 5 mittels einer $lookup$-Methode des Index, alle betroffenen Ergebnistupel $v \in \mathcal{V}'$ mit $\mathcal{V}' \subset \mathcal{V}$ für jede Eingaberelation $\mathcal{R}$ berechnet.

**Input:** Personenbezogene Anfrage $\mathcal{Q}'$
**Input:** Einwilligungsentzug $k \in K$
**Input:** Menge aller Eingaberelationen $\mathcal{A} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$
**Data:** Index $\mathcal{I}$
**Result:** $\mathcal{V}'$

```
1 if k.E ∉ Q'.E then
2     return ∅;
3 V' ← ∅;
4 for R ∈ A do
5     V' ← lookup(R, k.id) ;
6 return V'
```

**Algorithmus 5.2:** Identifikation vom Ergebnistupel

In *Szenario* 1.2 ist das Analyseergebnis aus den *Customer*, *Lineitem* und *Orders* zusammengesetzt und es findet eine Reduzierung auf den Kundennamen und die Produktkategorie statt. Das hat zur Folge, dass mehrere Eingabetupel aus einer Eingaberelation zum Gesamtergebnis beitragen. Eine Neuberechnung bei einem Einwilligungsentzug muss somit mehrere Eingaberelationen und mehrere Eingabetupel in Betracht ziehen. Bob hat im *Szenario* 1.2 seine Einwilligung für Bestellung $o\_orderkey = 1$ in der *Orders*-Tabelle entzogen. Damit die Neuberechnung für Bob nun korrekt durchgeführt werden kann, müssen zunächst die Ergebnisse identifiziert werden, die durch den Einwilligungsentzug betroffen sind. Die Identifikation ist in Abbildung 5.7 dargestellt. Die grau hervorgehobenen Tupel sind hierbei die betroffenen Eingabedaten und Analyseergebnisse. Das Ergebnis des Ablaufs sind die identifizierten Ergebnistupel *A* und *B*. Die gestrichelten Pfeile stellen die Assoziationen der betroffenen Eingabetupel über den Index zu den Ergebnistupel dar. Aus Übersichtsgründen wurden einige Attribute in der Abbildung weggelassen.

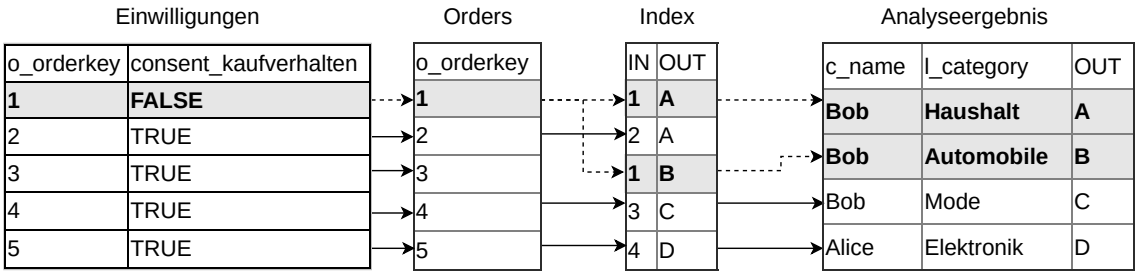



**Abbildung 5.7:** Identifikation der Ergebnistupel für die Orders-Tabelle

#### Schritt II: Identifikation der Eingabetupel

Nach der Identifikation der Ergebnistupel können alle betroffenen Ergebnistupel berechnet werden. Der Ablauf ist in der Abbildung 5.8 dargestellt und lässt sich wie folgt zusammenfassen. Falls keine Ergebnistupel im ersten Schritt gefunden wurden, ist eine Neuberechnung nicht notwendig, ansonsten können mittels des Index alle betroffenen Eingabetupel gefunden werden. Falls keine Eingabetupel existieren ist keine Berechnung notwendig, ansonsten werden alle betroffenen Eingabetupel zurückgeliefert. Diese Eingabetupel können für die Neuberechnung verwendet werden.

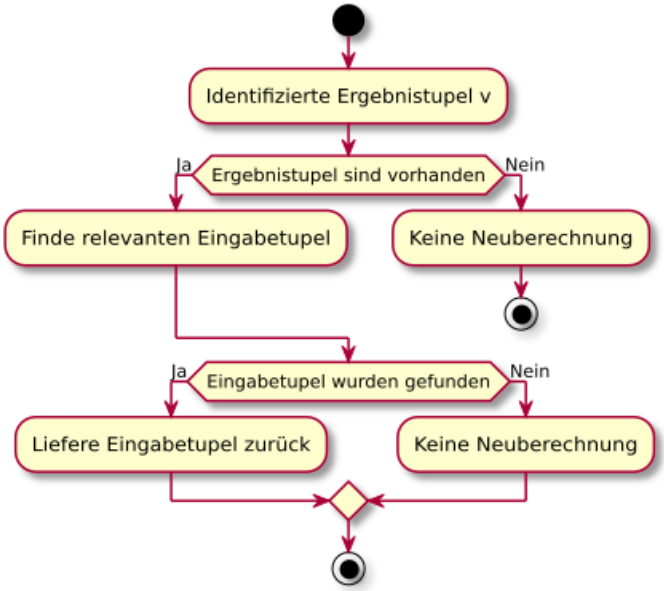



**Abbildung 5.8:** Identifikation von Eingabetupeln

Mittels des Algorithmus 5.3 werden alle betroffenen Eingabetupel identifiziert. Dazu erwartet der Algorithmus als Eingabe die Ergebnisrelation $\mathcal{V}' \subset \mathcal{V}$ und die Menge aller Eingaberelationen mit denen $\mathcal{V}$ berechnet wurde. Der Algorithmus prüft in Zeile 1 zunächst, ob Ergebnistupel in $\mathcal{V}'$ existieren. Falls dies nicht der Fall ist, können die Eingabetupel nicht identifiziert werden und es wird eine leere Menge zurückgeliefert. Ansonsten wird in Zeile 4 über alle Ergebnistupel und in Zeile 5 über alle Eingaberelationen iteriert. In Zeile 7 wird geprüft, ob das Ergebnistupel aus Zeile 4 im Index der Eingaberelation aus Zeile 5 vorhanden ist. Falls dies der Fall ist werden in Zeile 8 mittels des reverseLookup des Indexes, alle betroffenen Eingabetupel für ein Ergebnistupel berechnet. Das Ergebnis ist eine Menge von Eingaberelationen, die jeweils ein Teilmenge der Eingaberelationen darstellen.

**Input:** Eingaberelation in $\mathcal{V}' \subset \mathcal{V}$
**Input:** Menge aller Eingaberelationen $\mathcal{A} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$
**Data:** Index $\mathcal{I}$
**Result:** Betroffenen Eingabetupel $\mathcal{A}'$

```
1  if V' = ∅ then
2      return ∅;
3  A' ← ∅;
4  for v ∈ V' do
5      for R ∈ A do
6          R' ← ∅;
7          if v ∈ I_R then
8              R' ← reverseLookup(R, v);
9              A ← R';
10 return A';
```

**Algorithmus 5.3:** Identifikation vom Eingabetupel

Abbildung 5.9 zeigt den vollständigen Verlauf der Identifikation von Schritt 2.2 mittels dem *Szenario* 1.2 . Zunächst wird mittels des DS-GVO konformen Datenmodells ein Einwilligungsentzug vollzogen. Dieser hat Auswirkung auf die Bestellung mit $o\_orderkey = 1$. Ergebnisse *A,B* sind durch den Entzug betroffen und werden mittels des *Orders-Index* identifiziert. Dies wird durch die orangenen und gestrichelten Pfeile dargestellt. Durch die Identifikation der Ergebnisse ist es nun möglich, die jeweiligen Eingabetupel in den Eingaberelationen *Lineitem, Customer* und *Orders* zu identifizieren. Dies wird jeweils durch die farblich markierten und gestrichelte Pfeile ausgehenden vom *Orders-Index* dargestellt. Die roten Pfeile zeigen dabei die betroffenen Tupel in der *Lineitem*-Tabelle, die grünen Pfeile die betroffenen Tupel in der *Customer*-Tabelle und die blauen Pfeile die betroffenen Tupel in der *Orders*-Tabelle.

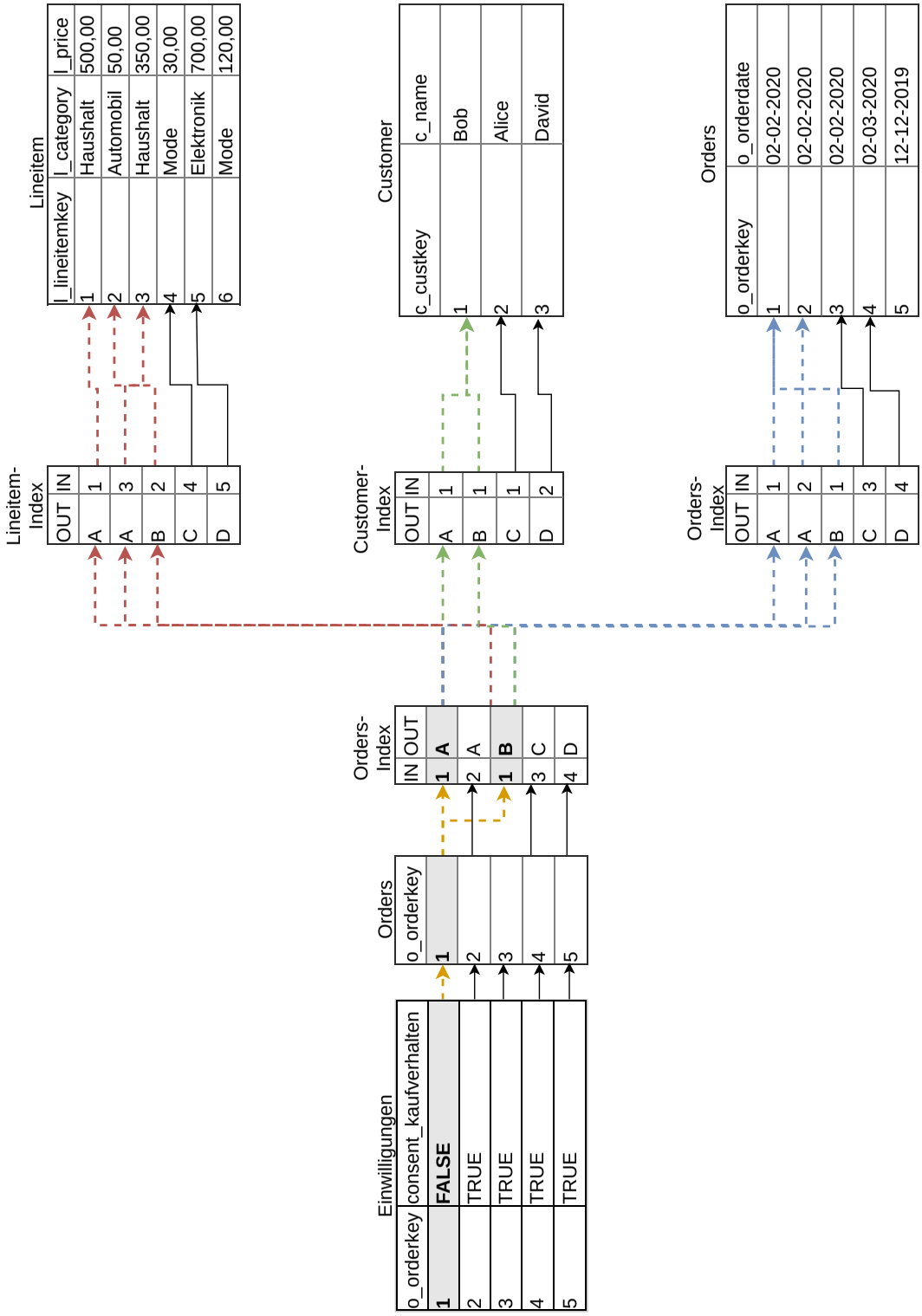



**Abbildung 5.9:** Identifikation der Eingabetupel am *Szenario* 1.2

Anhand der Identifikation der Eingabetupel kann nun eine Neuberechnung stattfinden. Der potentielle Vorteil durch die Identifikation ist eine effizientere Neuberechnung, da dadurch nur ein Teil der Daten für die Berechnung verwendet werden muss. Der nachfolgende Abschnitt 5.2.2 beschreibt dabei die Schritte 2.3 und 2.4.

### 5.2.2 Aktualisierung des Ergebnisses

Nachdem anhand des Schritts 2.2 die betroffenen Eingabetupel identifiziert wurden, kann eine Neuberechnung auf einen Teil der Eingabedaten durchgeführt werden. In diesen Abschnitt werden die Schritte 2.3 und 2.4 von Abbildung 5.1 beschrieben.

Der Algorithmus 5.4 berechnet anhand der identifizierten Eingabetupel das Teilergebnis und erwartet als Eingabe die Menge aller betroffenen Eingabetupel. Zunächst identifiziert der Algorithmus in Zeile 2 alle betroffenen Ergebnistupel anhand der geänderten Eingabetupel. Mittels den betroffenen Ergebnistupel werden alle betroffenen Eingabetupel in Zeile 4 identifiziert (Schritt 2.2). In Zeile 5 werden alle Eingabetupel entfernt, die als Eingabe für den Algorithmus verwendet wurden. Dies ist notwendig, damit das Ergebnis nicht mit den Eingabetupeln berechnet wird, für die eine Änderung bzw. eine Einwilligungsentzug stattgefunden hat. Mittels den betroffenen Eingabetupel kann die personenbezogene Anfrage in Zeile 6 erneut durchgeführt werden (Schritt 2.3). In Zeile 7 werden alle alten Ergebnisse herausgefiltert, damit in Zeile 8 das Teilergebnis mit dem gefilterten Ergebnis vereinigt werden kann (Schritt 2.4).

**Input:** Geänderte Eingabetupel $\mathcal{C}$
**Data:** Personenbezogene Anfrage $\mathcal{Q}'$
**Data:** Ergebnis $\mathcal{V}$
**Data:** Index $\mathcal{I}$
**Result:** Aktualisiertes Ergebnis $\mathcal{V}'$

1 **for** $c \in \mathcal{C}$ **do**
2 $\quad \mathcal{V}_C \leftarrow lookup(\mathcal{I}_C, c)$;
3 **for** $v \in \mathcal{V}_C$ **do**
4 $\quad \mathcal{R}' \leftarrow reverseLookup(\mathcal{I}, v)$;
5 $\quad \mathcal{R}' \leftarrow \mathcal{C} \bowtie_{id \neq id} \mathcal{R}'$;
6 $\mathcal{V}' \leftarrow \mathcal{Q}'(\mathcal{R}')$;
7 $\mathcal{V} \leftarrow \mathcal{V} \bowtie_{OUT \neq OUT} \mathcal{V}_C$;
8 $\mathcal{V}' \leftarrow \mathcal{V} \cup \mathcal{V}'$;
9 return $\mathcal{V}'$;

**Algorithmus 5.4:** Aktualisierung vom Ergebnis

Anhand des begleitenden Szenarios wird eine Neuberechnung auf einen Teil der Daten durchgeführt. Dabei wird die Bestellung von Bob mit $o\_orderkey = 1$ aus den Eingabedaten entfernt. Dies sorgt dafür, dass das bestehende Ergebnis erneut berechnet werden muss. In einer Neuberechnung ohne Index müssten alle Eingabetupel für die Berechnung verwendet werden. Die indexgestützte Neuberechnung ermöglicht eine potentiell effizienter Neuberechnung, in dem nur die betroffenen

Tupel verwendet werden. Im *Szenario* 1.2 bedeutet dies, das nur $\frac{1}{3}$ der Tupel von der *Customer*-Tabelle und $\frac{3}{6}$ der Tupel von der *Lineitem*-Tabelle verwendet werden. Aus der *Orders-Tabelle* werden $\frac{1}{5}$ Tupel verwendet, da das Tupel $o\_orderkey$ zwar vom Index Lookup identifiziert wird, jedoch in der Berechnung nicht mehr verwendet werden darf und somit aus der Menge betroffenen Eingabetupel der Tabelle *Orders* herausgefiltert wird. Insgesamt werden für die indexgestützte Neuberechnung $\frac{5}{14}$ aller Tupel benötigt. Dies entspricht einer Einsparung der Daten um ca. 65%. Diese Einsparung kann bei sehr großen Daten für eine potentiell effizientere Neuberechnung führen, vor allem wenn die Daten in einem Rechnerverbund verteilt und im Hauptspeicher berechnet werden.

Die Tabelle 5.2 zeigt das Teilergebnis der Berechnung. Das Ergebnis ist das Resultat der Anfrage 1.2 und verwendet für die Berechnung nur die Bestellung mit $o\_orderkey = 2$, da die Bestellung mit $o\_orderkey = 1$ über keine Einwilligung verfügt bzw. die Einwilligung entzogen wurden. Das Ergebnis kann mit diesem Teilergebnis aktualisiert werden und somit ist das Ergebnis wieder DS-GVO konform.

**Tabelle 5.2:** Teilergebnis der indexgestützten Berechnung

| c_name | c_address | c_phone | sum(l_price) | l_category |
|---|---|---|---|---|
| Bob | 9 Hermina Crossing | 555-1234 | 350,00 | Haushalt |

# 6 Implementierung

In diesem Kapitel wird die Implementierung des DS-GVO konformen Datenmodell aus Kapitel 4 und der indexgestützten Neuberechnung aus Kapitel 5 vorgestellt. Zunächst wird eine Übersicht über die Architektur der Implementierung und der verwendeten Software gegeben. Danach werden die Komponenten der Implementierung detaillierter vorgestellt. Der Abschluss des Kapitels ist die Vorstellung der verschiedenen Implementierungsalternativen, die für die Implementierung in Betracht gezogen wurden.

## 6.1 Architekturübersicht

Die Implementierung wird in Apache Spark vorgenommen, da die verwendete Provenance-Lösung *Pebble* in Apache Spark implementiert wurde [DH19; DH20]. Die hier vorgestellte Implementierung kann jedoch ebenso in anderen VSDB-Systemen und anderen Provenance-Lösungen implementiert werden. Abbildung 6.3 zeigt eine Übersicht über die Architektur der einzelnen verwendeten Komponenten. Die Grundlage der Implementierung bildet Apache Spark bzw. der Apache Spark Core. Auf den Apache Spark Core baut Apache Spark SQL auf. Apache Spark SQL ermöglicht, die Datenverarbeitung in einem relationalen Datenmodell [SDC+16; ZXW+16]. Die Abstraktion des relationalen Datenmodell ist hierbei die *DataFrame*-Klasse, die bereits im Abschnitt 2.3 beschrieben wurde. Die Provenance-Lösung *Pebble* baut auf Apache Spark SQL bzw. der *DataFrame*-Klasse auf. Dazu haben Diestelkämper und Herschel [DH20] die Funktionen der *DataFrame*-Klassen so erweitert, dass sie Provenance-Daten für jede Transformation und jedes Tupel erzeugen. Die Komponenten des DS-GVO konformen Datenmodells verwenden Apache Spark SQL und die DataFrame-Klasse und werden im Abschnitt 6.2 beschrieben. Der Index, der für die indexgestützte Neuberechnung verwendet wird, verwendet Pebble zum Aufbau der Eingabe- und Ausgabeassoziationen und wird in Abschnitt 6.3 beschrieben.

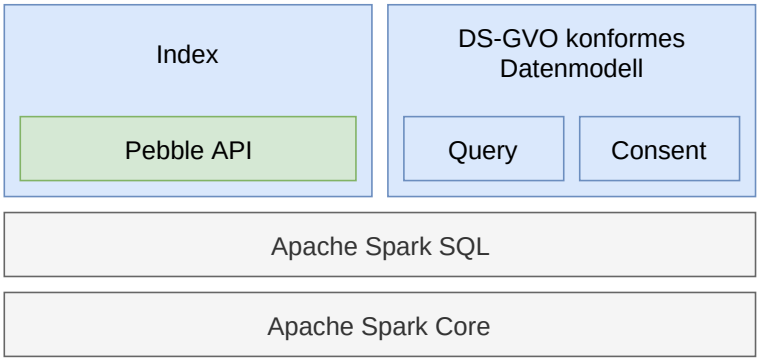



**Abbildung 6.1:** Architekturübersicht

Für die Implementierung der Prototypen wurde Scala in der Version 2.11 verwendet. Scala ist eine objektorientierte und funktionale Programmiersprache, die innerhalb der Java Virtual Machine (JVM) läuft [OAC+04]. Die Prototypen wurde in Apache Spark implementiert. Die verwendete Version ist 2.2.

## 6.2 DS-GVO konformes Datenmodell

Wie in Abschnitt 4.1 bereits beschrieben, kann die Einwilligung als eine separate Relation erzeugt werden. Dies wird in Apache Spark als DataFrame abgebildet. Dazu wird die *ConsentableDataFrame*-Klasse eingeführt. Sie erweitert die DataFrame-Klasse, um Funktionalitäten, die es ermöglichen den Eingabedaten einem Zweck zuzuweisen. Das Klassendiagramm 6.2 zeigt eine Übersicht über alle Klassen, die für die Einwilligung als DataFrame benötigt werden. Die nachfolgenden Abschnitte behandeln, die einzelnen Klassen im Detail.

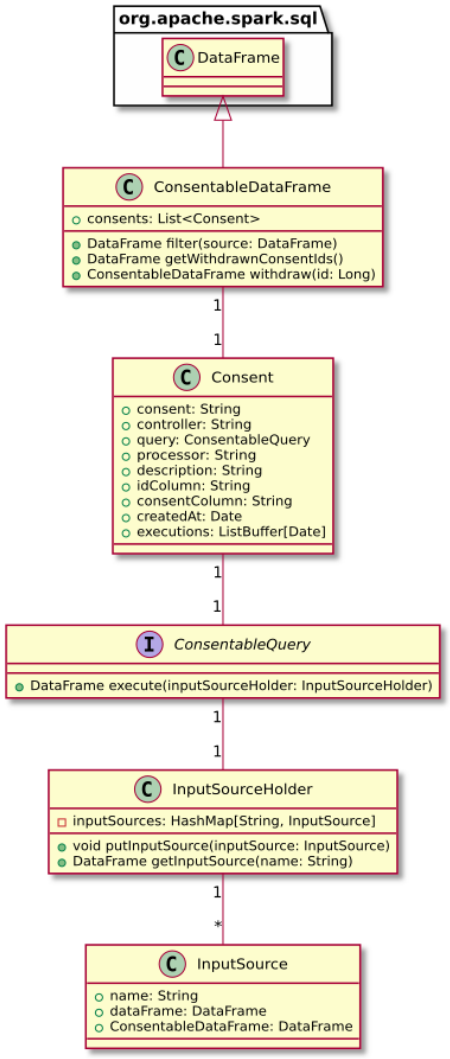



**Abbildung 6.2:** DS-GVO konformes Datenmodell: Klassendiagramm

### 6.2.1 Consent

Die Consent-Klasse bzw. das Consent-Objekt verfügt über Informationen bezüglich der Einwilligung und spiegelt somit den Zweck einer Einwilligung wider. Des weiteren beinhaltet sie die *ConsentableQuery*, für die die Einwilligung gilt. Dadurch ist eine personenbezogene Abfrage (ConsentableQuery) an ihren Zweck (Consent) gebunden. Die Klasse besteht aus den folgenden Feldern.

**consent** Der Name, der Einwilligung z. B. *Kaufverhalten*

**controller** Der Name, der für die Ausführung der Analyse und damit die Beachtung der Einwilligung verantwortlich ist

**processor** Falls eine Drittpartei für die Analyse beauftragt wurde, kann dieses Feld befüllt werden

**description** Eine zusätzliche Beschreibung des Zweckes

**idColumn** Das ist das Primärschlüsselattribut aus der Eingaberelation, für die die Einwilligung vergeben wurde

**consentColumn** Ein automatisch generiertes Feld, mit dem das *TRUE*/*FALSE*-Attribut für die jeweilige Eingaberelation identifiziert werden kann

**createdAt** Zeitpunkt an dem die Einwilligung bzw. der *Consent* erzeugt wurde

**executions** Eine Liste von Zeitpunkten an dem die Query bzw. die Einwilligung in einer Query verwendet wurde.

### 6.2.2 ConsentableDataFrame

Die *ConsentableDataFrame*-Klasse ist eine Erweiterung der Apache Spark *DataFrame*-Klasse und verfügt über Funktionen, die zum Filtern der Daten nach der Einwilligung bzw. *Consent* verwendet werden. Sie bildet die Einwilligung bzw. den Zweck mittels der *Consent*-Klasse ab, in dem die einzelnen *Consent*-Objekte als Attribute im DataFrame abgebildet werden. Sie enthält den Primärschlüssel der Eingaberelation als Fremdschlüssel, sowie Attribute welche die Einwilligungen als booleschen Wert TRUE/FALSE widerspiegeln. Der ConsentableDataFrame wird mit einer Eingaberelation und *Consents* initialisiert. Der ConsentableDataFrame dient somit als zentrale Stelle für die Verwaltung der Einwilligungen und hat folgende Funktionen.

**consents** Instanzen der Consent-Klasse. Diese Instanzen geben den Zweck der jeweiligen Einwilligung vor und werden als Attribute im *ConsentableDataFrame* abgebildet. Mittels der *idColumn* und der *consentColumn* wird die Eingaberelation gefiltert.

**filter** Die Filter-Funktion dient zum Filtern der Eingaberelation. Dazu wird die Eingaberelation bzw. der Eingabe-DataFrame mittels dem *idColumn* verbunden und und mittels aller *Consent*-Objekte gefiltert, so dass das Ergebnis nur diejenigen Eingabetupel enthält bei denen alle Einwilligungen gegeben wurde.

**getWithdrawnConsentIds** Diese Methode liefert alle Primärschlüsselattribute der Eingaberelation bei denen die Einwilligung entzogen wurde bzw. bei denen consentColumn FALSE ist. Diese Methode liefert im Gegensatz zu der *filter*-Methode nur IDs, da der Index nur IDs für die Berechnung benötigt.

**withdraw** Mittels der withdraw-Methode können Einwilligungen für Tupel entzogen werden. Dazu wird das Primärschlüsselattribut des Eingabetupels benötigt. Auf Grund Unveränderlichkeit von DataFrames wird beim Entzug der Einwilligung ein neuer *ConsentableDataFrame* erzeugt.

### 6.2.3 ConsentableQuery

Das Interface *ConsentableQuery* dient als Abstraktion von personenbezogenen Analysen in Apache Spark. Durch diese Abstraktion können Analysen benamt und einem Consent zugewiesen werden. Dazu muss die *execute*-Methode implementiert werden. Diese Methode übernimmt als Parameter einen *InputSourceHolder*, welcher mehrere DataFrames enthält. Dies ist notwendig, da eine Abfrage bzw. Analyse mehrere Eingaberelationen verwenden kann. Das Ergebnis der execute-Methode ist das Ergebnis der personenbezogenen Anfrage als DataFrame. Die einzelnen DataFrames bzw. Eingaberelationen werden hierbei durch Namen unterschieden. Des weiteren kann jedem DataFrame ein *ConsentableDataFrame* zugewiesen werden. Beim Abruf des DataFrames mittels *getInputSource* wird der betroffene DataFrame automatisch für die Query gefiltert. Das passiert aber nur, wenn die InputSource, die Eingaberelation und einen *ConsentableDataFrame* als Parameter übergeben bekommt, da dadurch die *filter*-Funktion innerhalb der *getInputSource* aufgerufen wird, bevor der DataFrame zurückgeliefert wird.

Die Implementierung 6.1 der Analyse 1.2 aus *Szenario* 1.2 sieht demnach wie folgt aus. Der Name *Kaufverhalten* ist hierbei der Name der konkreten Instanz der *ConsentableQuery*-Implementierung. Die konkrete Implementierung erwartet dabei, das die *InputSourceHolder*-Klasse über die DataFrames *Lineitem*, *Orders* und *Customers* verfügt. Die einzelnen DataFrames werden dabei durch die Methode *getInputSource* und dem Namen des DataFrames herangezogen. Der Name kann beliebig sein, muss aber mit den Namen im *InputSourceHolder* übereinstimmen. Danach können die drei DataFrames verwendet werden, um die personenbezogene Analyse des *Szenario* 1.2 mittels der Apache Spark DSL zu implementieren.

```
class Kaufverhalten extends ConsentableQuery {
        override def execute(inputSource: InputSourceHolder): DataFrame = {

        // Dieser Abschnitt extrahiert alle notwendigen Eingaberelationen bzw. DataFrames
        // Der Aufruf der getInputSource-Method sorgt dafuer das der betroffene DataFrame bereits
             nach Einwilligung gefiltert wird
        val lineitems = inputSource.getInputSource("lineitems")
        val orders = inputSource.getInputSource("orders")
        val customers = inputSource.getInputSource("customers")

        // Dieser Abschnitt erzeugt das Analyseergebnis mittels der DataFrame DSL
        return orders.where($"o_orderdate" >= "01-01-2020")
        .join(customers, $"c_custkey" === $"o_custkey")
        .join(lineitems, $"o_orderkey" === $"l_orderkey")
        .groupBy($"c_name", $"c_address", $"c_phone", $"l_category")
        //functions.sum ist eine Apache Spark SUM-Funktion wie sie etwa in nativen SQL zu finden ist
        .agg(functions.sum($"l_price"))
        }
}
```

**Listing 6.1:** Implementierung der Analyse 1.2 als ConsentableQuery

### 6.2.4 InputSouceHolder

Die Klasse *InputSourceHolder* beinhaltet alle DataFrames in einer *HashMap*. Der Schlüssel ist dabei der Name des DataFrames und der Wert ist eine InputSource. Die *InputSource*-Klasse ist eine reine Datenklasse, die über Eingabedaten als DataFrame und ein ConsentableDataFrame verfügt, sowie über einen Namen. Durch diese Klasse können mehrere DataFrames spezifiziert werden. Sie besteht aus insgesamt zwei Methoden.

**putInputSource** Fügt eine InputSource der HashMap hinzu. Als Name wird der Name der InputSource verwendet.

**getInputSource** Liefert den DataFrame der InputSource zurück. Falls die InputSource zusammen mit einem ConsentableDataFrame dem InputSourceHolder übergeben wurde, wird der DataFrame mittels ConsentableDataFrame gefiltert, sodass dieser DS-GVO konform ist.

## 6.3 Indexgestützte Neuberechnung

Nachdem die Komponenten des DS-GVO konformen Datenmodells in Abschnitt 6.2 beschrieben wurde, befasst sich dieser Abschnitt mit dem Index aus Kapitel 5. Zunächst wird ein Überblick über die implementierten Klassen und deren Funktionsweise in Abschnitt 6.3.1 gegeben, danach werden Implementierungsalternativen in Abschnitt 6.3.5 vorgestellt, die in dieser Arbeit in Erwägung gezogen wurden.

### 6.3.1 Klassendiagramm und Funktionsweise

Die Architektur des Index lässt sich in die Klassen *Index*, *AffectedPartitions*, *AffectedPartition* und *PartitionInformation* aufteilen. Die Hauptfunktionalität d. h. die Identifikation der betroffenen Eingabetupel, wird hierbei von der Index-Klasse erfüllt. Die Klassen AffectedPartitions und AffectedPartition sind dabei die betroffenen Partitionen bzw. Eingabetupel. Die Klasse PartitionInformation ist eine reine Datenklasse und beinhaltet nur Informationen über die Partitionierung der Eingaberelation und ist bei der Erstellung des Index wichtig.

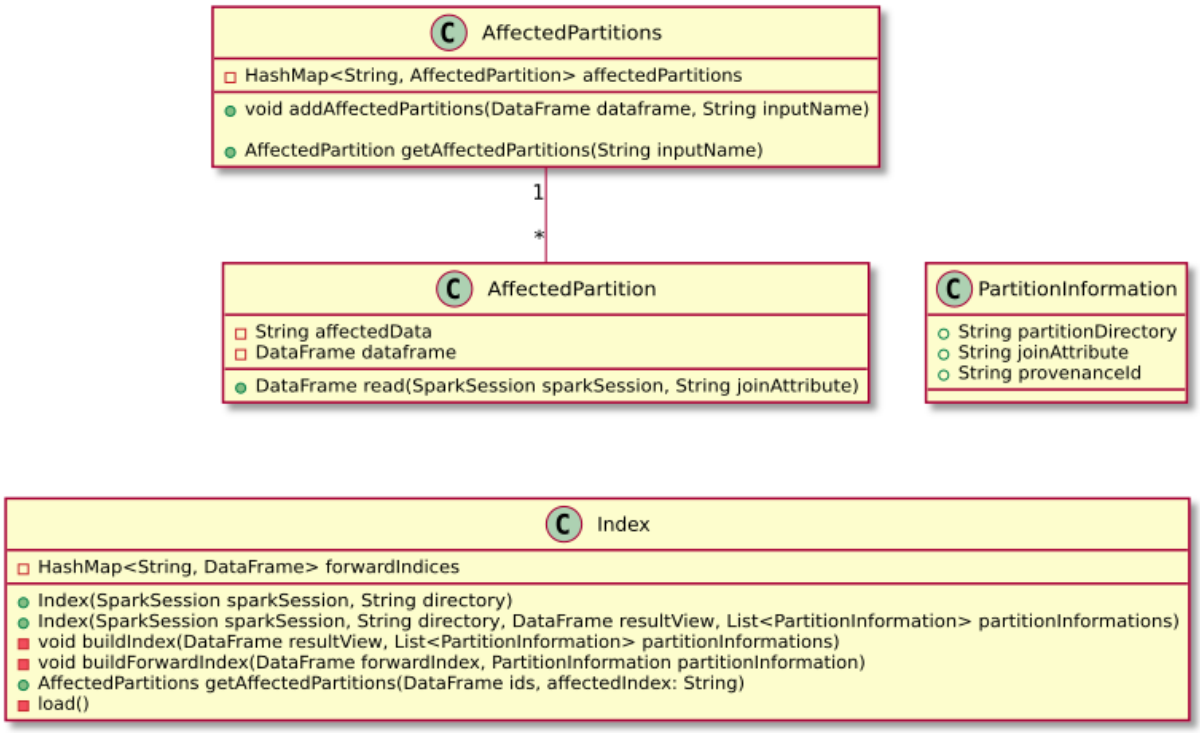



**Abbildung 6.3:** Index: Klassendiagramm

### 6.3.2 Index

Die Index-Klasse implementiert die Identifikation der Ergebnistupel. Für eine potentiell effiziente Neuberechnung wird der Index als DataFrame implementiert. Dies hat den Vorteil, dass die volle Leistung des Spark-Clusters genutzt werden kann. Der Index muss dabei initial mittels Provenance-Daten der Provenance-Lösung *Pebble* einmalig erzeugt werden. Nachfolgende Berechnungen können dann den Index heranziehen, um partielle Neuberechnung durchzuführen. Die Klasse verfügt dabei über folgende Methoden und eine Menge von Indizes. Diese Indizes sind DataFrames, die die Assoziation zwischen Eingabe- und Ergebnistupel genau einer Relation beinhalten.

**buildIndex** Methode für den Aufbau des Index. Diese Methode benötigt die Ergebnisrelation, sowie Informationen über die Partitionierung der Eingaberelationen

**buildForwardIndex** Methode für den Aufbau eines Index. Zum Aufbau wird die entsprechende Eingaberelation benötigt, sowie die Information über die Partitionierung der Daten

**getAffectedPartitions** Diese Methode berechnet anhand der übergebenen Eingabetupel, sämtliche betroffene Eingabetupel. Diese Methode übergibt die betroffenen Eingabetupel samt Partitionen an die *AffectedPartitions*-Klasse.

**load** Diese Methode ermöglicht das Laden des Indexes von der Festplatte.

### 6.3.3 AffectedPartition

Die Klasse *AffectedPartition* repräsentiert die betroffenen Eingabetupel und die betroffene Eingaberelation bei der Identifikation mittels Index. Sie besteht aus einer einzigen *read-Funktion*, die die betroffenen Partitionen einliest und nur die Eingabetupel aus den Partitionen herausfiltert, die auch tatsächlich gesucht werden. Das Ergebnis sind alle betroffenen Eingabetupel.

Die *AffectedPartitions*-Klasse ist eine Hilfsklasse, die eine Anzahl von *AffectedPartition* bündelt. Sodass es möglich ist über den Namen des Primärschlüssels auf die jeweilige Partitionen zuzugreifen.

### 6.3.4 PartitionInformation

Beim Erzeugen des Index, benötigt der Index zusätzliche Informationen über die Partitionen. Diese Klasse bündelt alle benötigten Informationen. Diese bestehen aus den folgenden Teilen.

**partitionDirectory** Der Pfad zum Verzeichnis, in dem sich die Partitionsdateien befinden

**joinAttribute** Name des Primärschlüsselattributes

**provenanceId** Name des Attributes von der Provenance-Lösung. Anhand dieser Id und dem Join-Attribute werden die Daten mit Informationen der Partitionen angereichert

### 6.3.5 Implementierungsalternativen

Dieser Abschnitt befasst sich mit Implementierungsalternativen, die in dieser Arbeit in Erwägung gezogen wurden. Die Implementierung stellte sich dabei mit steigender Datengröße als schwierig heraus, wodurch mehrere Alternative evaluiert und implementiert wurden. Diese Alternativen beziehen sich auf den Aufbau des Index, die Partitionierung der Eingaberelationen und die Berechnung der betroffenen Eingabetupel. Diese Problematiken werden in den jeweiligen Abschnitten detailliert beschrieben. Zunächst wird die Indexerzeugung vorgestellt, gefolgt von der Partitionierung der Daten mittels Apache Spark. Der letzte Abschnitt befasst sich der Identifikation der Eingabetupel mittels dem Index.

### Indexerzeugung

Die *Index*-Klasse implementiert den Aufbau der Indexstruktur, die Identifizierung der einzelnen Tupel, sowie das Laden und Speichern der Indexdateien. Um eine potentiell Neuberechnung von Analysen durchzuführen, muss der Index zunächst erzeugt werden. Die Erzeugung des Index geschieht dabei unter der Verwendung von Provenance-Daten, die durch die Provenance-Lösung Pebble erzeugt werden. Pebble bietet dazu eine *backtrack*-Funktion an, die eine Zurückverfolgung von Ergebnistupel $v \in \mathcal{V}$ zu ihren jeweiligen Eingabetupel $r \in \mathcal{R}$ ermöglicht. Pebble annotiert die Eingaberelationen dazu mit einem Provenance-ID-Attribut, so dass alle Transformationen bis zu ihrem Ursprung zurückverfolgt werden können. Dazu werden entweder zufällige und eindeutige Provenance-IDs von Pebble generiert oder es werden bestehende Attribute der Eingaberelationen verwendet [DH20]. Da in dieser Arbeit davon ausgegangen wird, dass jede Eingaberelation über eindeutige Primärschlüsselattribute verfügt, werden diese als Provenance-IDs verwendet. Diese IDs dienen als grundlegende Information für den Index, anhand derer die Identifikation der Eingabetupel durchgeführt wird.

In der ersten Version der Implementierung wurde der Index als Scala-Hashtabelle im lokalen Arbeitsspeicher des Driver-Programms aufgebaut. Hashtabellen haben eine konstante Suchzeit von $O(1)$ und sind somit eine schnellere Alternative als Binärbäume [Kne19]. Dazu wurde in der ersten Version dieser Implementierung, die vollständige Eingaberelation mittels Pebble zurückverfolgt. Da die Eingaberelation als DataFrame innerhalb von Apache Spark dargestellt wird, wurden zunächst die betroffenen Provenance-IDs extrahiert. Dazu wurde zunächst die Eingaberelation auf dem Driver-Programm mittels der *collect*-Methode geladen und alle Provenance-IDs ausgelesen. Dies stellte sich jedoch als nicht durchführbar heraus, da die gesamte Eingaberelation in den Arbeitsspeichers des Driver-Programms geladen werden muss. Je nach Größe der Arbeitsspeichers des Knotens, auf dem sich das Driver-Programm befindet, und je nach Größe der Eingabedaten kann dies zum Absturz des Driver-Programms führen, da die Größe der Eingabedaten um ein Vielfaches größer sein kann als der Arbeitsspeicher des Driver-Programms [ApaoDc].

Um einen Absturz durch einen zu kleinen Arbeitsspeicher zu verhindern, wurde der Index soweit angepasst, das jeweils immer nur ein Tupel aus der Ergebnisrelation $\mathcal{V}$ zurückverfolgt wurde. Dies wurde durch die Verwendung der *limit*-Methode des Ergebnis-DataFrame erreicht. Die *limit*-Methode erwartet eine Zahl anhand derer die Ergebnisrelation reduziert wird. Dieser Ansatz funktioniert für einen Datensatz mit 10 bis 20 Ergebnis- und Eingabetupel weist jedoch einen Nachteil auf, sobald der Datensatz größer wird. Die Iteration über alle Ergebnistupel anhand der *limit*-Funktion erzeugt einen sehr langen und komplexen Ausführungsplan. Des weiteren muss nach jeder Iteration, das verwendete Ergebnistupel aus der Ergebnisrelation herausgefiltert werden. Dies führt ebenso dazu, dass der Ausführungsplan pro Iteration wächst. All diese Nachteile führen dazu, dass die Zeit pro Iteration stark ansteigt. Bei einer hohen Anzahl von Ergebnistupeln summiert sich das Backtracking zu einer nicht vertretbaren Zeit auf.

Um die genannten Probleme zu bewältigen, wurde der Index als DataFrame implementiert. Dies hat den Vorteil, das dadurch die volle Leistung des Clusters genutzt wird. Die Backtracking-Funktion der Provenance-Lösung wurde dabei so angepasst, dass die Assoziation sämtlicher Ergebnistupel zu Eingabetupel pro Relation bereits in der Provenance-Lösung erzeugt werden. Pebble liefert für jedes Ergebnistupel, die entsprechenden Eingaberelationen. Die Tupel der zurückverfolgten Eingaberelationen habe, jedoch keine Information über das Ergebnistupel. Dies führt zu der bereits erwähnte Problematik, das durch jedes Ergebnistupel iteriert werden muss, um die Assoziation

zwischen Eingabe und Ergebnis zu erzeugen. Um diese langwierigen Iterationen zu vermeiden und um direkt alle Assoziationen innerhalb Pebble aufzubauen, wurde die Logik der Backtracking-Funktion dabei so erweitert, dass die Funktion nicht die betroffenen Tupel der Eingaberelation zurückliefert, sondern nur die betroffenen Primärschlüssel des Eingabetupel zum jeweiligen Ergebnistupel. Durch diese Logik wird der Index bereits teilweise in der Provenance-Lösung aufgebaut. Dies führte zu einer schnelleren und skalierbaren Indexerzeugung, da hier die Logik vom Pebble genutzt werden konnte. Der folgende Pseudocode stellt die modifizierte Backtracking-Funktion dar.

**Input:** $C, L_{\mathcal{I}_\mathcal{R}}$
**Result:** $L_{\mathcal{I}_\mathcal{R}}$
1 transformations $\leftarrow getTransformations(C)$;
2 **for** $transformation \in transformations$ **do**
3 $\quad \mathcal{R} \leftarrow transformation.dataset$;
4 $\quad$ **if** *isInitialDataset(*$\mathcal{R}$*)* **then**
5 $\quad\quad \mathcal{I}_\mathcal{R} \leftarrow \pi_{INPUT,OUTPUT}(\mathcal{R}')$;
6 $\quad\quad append(L_{\mathcal{I}_\mathcal{R}}, \mathcal{I}_\mathcal{R})$;
7 $\quad\quad$ return $L_{\mathcal{I}_\mathcal{R}}$;
8 $\quad$ **else**
9 $\quad\quad \mathcal{R}' \leftarrow \mathcal{R} \bowtie_{INPUT=INPUT} C$;
10 $\quad\quad$ return backtrack($\mathcal{R}', L_{\mathcal{I}_\mathcal{R}}$);

**Algorithmus 6.1:** Modifizierte Backtracking-Funktion: $backtrack(I, L_{\mathcal{I}_\mathcal{R}})$

Mittels des Algorithmus 6.1 werden alle Eingabe- und Ausgabetupel identifiziert, in dem über alle angewandten Transformationen iteriert wird. Der Algorithmus erwartet als Eingabe eine geänderte Eingaberelation und eine leere Liste in die Indizes hinzugefügt werden. In Zeile 1 werden zunächst alle Transformationen der geänderten Eingaberelation extrahiert. Diese Transformation beinhalten immer eine Relation, die von der Transformation verwendet wird. Dies geschieht in Zeile 3. In Zeile 4 wird geprüft, ob weitere Transformationen vorhanden sind, falls dies nicht der Fall ist, hat der Algorithmus das Ergebnis bis zur initiale Eingaberelation zurückverfolgt und fügt die zurückverfolgte Relation bzw. Index in die Liste hinzu. Falls weitere Transformationen vorhanden sind, wird der Algorithmus in Zeile 10 rekursiv aufgerufen. Der rekursive Aufruf bekommt dabei als Eingabe die Liste mit allen Indexrelationen, sowie die Eingaberelation aus der Transformation. Diese Eingaberelation wird aber vorher so gefiltert, dass nur die Tupel weiterverfolgt werden, die im initialen Aufruf dem Algorithmus übergeben wurden. Das Ergebnis ist eine Liste bzw. eine Menge mit allen zurückverfolgten Eingaberelationen, die jedoch nur über ihren Primärschlüssel verfügen und den Primärschlüssel des Ergebnis zu dem sie beigetragen haben.

#### Partitionierung

Die Identifikation der Eingabetupel erfolgt durch die Identifikation der Partition. Eine Partition ist dabei ein Teil der Daten einer Eingaberelation, die sich auf der Festplatte befindet. Diese Teildaten beinhalten eine Menge der Eingabetupel. Eine optimale Partitionierung wäre eine Partitionsdatei pro Eingabetupel. Aus praktischen Gründen ist dies nicht möglich, da eine Eingaberelation aus über mehrere Millionen bis Milliarden Eingabetupeln bestehen kann. Um dies zu verhindern, werden

Eingabetupel auf eine begrenzte Anzahl von Partitionen verteilt. Die Partitionierung der Daten ist ein wesentlicher Bestandteil der indexgestützten Neuberechnung, da dadurch Daten überspringen werden können, die für eine Neuberechnung nicht benötigt werden. Mittels eines Partitionsattribut werden die Eingabetupel auf Partitionen verteilt. Das hierbei verwendete Partitionsattribut ist der eindeutige Primärschlüssel der Eingaberelation. Apache Spark bietet insgesamt drei Möglichkeiten zur Partitionierung von Daten an.

**partitionBy-Methode** Apache Spark bietet eine *partitionBy*-Methode an, die Daten anhand eines Partitionsattributes partitioniert und sie sortiert nach Partitionsattribut auf die Festplatte schreibt [ApaoDb].

**bucketBy-Methode** Eine ähnliche Methode wie partitionBy, die aber Hive-Tabellen verwendet [ApaoDb].

**repartition-Methode** Eine ähnliche Methode wie partitionBy, die einen DataFrame im Hauptspeicher partitioniert [LasoD].

In der ersten Iteration wurde die *partitionBy*-Methode in Betracht gezogen. Diese Methode partitioniert die Daten anhand eines Attributes und schreibt sie auf die Festplatte. Operatoren wie etwa Selektion-Operatoren können dann Daten anhand des Attributes überspringen. Da für diese Lösung eindeutige Primärschlüssel verwendet werden, wäre es problematisch den Primärschlüssel als Partitionsattribut zu verwenden, da jeder Schlüssel eindeutig ist und somit für jedes Eingabetupel eine Datei bzw. eine Partition erstellt werden muss. Um dieses Problem zu entgehen wurde eine Modulo-Hash-Funktion eingeführt, die die Eingabetupel auf eine vorgegebene Anzahl an Partitionen verteilt. Dieser Schritt führt zu einem temporären Erfolg, jedoch hat die partitionBy-Methode einen wesentlichen Nachteil. Sobald DataFrame mit der *cache*-Methode im Hauptspeicher gecacht werden, können die Partitionen nicht mehr übersprungen werden, da das interne Cache-Format Partitionen nicht mehr berücksichtigt. Caching ist aber ein unerlässlicher Teil für die Beschleunigung von Berechnung. Das Beibehalten von Partitionsinformationen für gecachte DataFrames wurde erst in der Apache Spark Version 2.3 implementiert [Apa17].

Die *bucketBy*-Methode ist eine alternative Methode für das Partitionieren der Daten anhand eines Partitionsattribut. Die Methode verwendet hierbei ebenso ein Hashing-Verfahren für das Verteilen der Daten auf Partitionen. Die Methode verwendet dabei Hive-Tabellen und HiveQL als Abfragesprache [ApaoDb]. Da die Implementierung in dieser Arbeit hauptsächlich auf der DSL der DataFrame-API aufbaut, wurde diese Methode nicht weiter verfolgt.

Da die beiden Varianten nicht für die Partitionierung verwendet werden konnten, wurde eine eigene Partitionierungslogik entwickelt. Diese Logik verwendet dabei die *repartition-Methode*. Diese Methode teilt dabei einen DataFrame mittels eines Hash-Verfahrens und einem Partitionsattribut in eine vordefinierte Anzahl an Partitionen [LasoD]. Die Aufteilung der Partitionen ist dabei nur ein Teil der Logik. Damit die Daten korrekt vom Index geladen werden können, muss jedem Primärschlüssel d. h. jedem Eingabetupel die entsprechende Partitionsdatei zugewiesen werden. Dies wurde erreicht, in dem die *input_file_name*-Methode[1] verwendet wurde. Sie ermöglicht die Zuweisung einzelner Tupel zu ihrer Partition. Die Partitionierung der Daten passiert dabei unabhängig vom Index. Der folgende Pseudocode zeigt die Funktionsweise der Partitionierungslogik.

[1] https://spark.apache.org/docs/2.2.0/api/java/org/apache/spark/sql/functions.html#input_file_name()

**Input:** $\mathcal{R}, directory, primarykey$
**Result:** $\mathcal{I}_{\mathcal{R}}$
1 $\mathcal{I}_{\mathcal{R}} \leftarrow \emptyset$;
2 $\mathcal{R}' \leftarrow load(directory)$;
3 $\mathcal{R}' \leftarrow withColumn(\mathcal{R}', "PARTITION", input\_file\_name())$;
4 $\mathcal{I}_{\mathcal{R}} \leftarrow \mathcal{R} \bowtie_{INPUT=primarykey} \mathcal{R}'$;
5 return $\mathcal{I}_{\mathcal{R}}$;

**Algorithmus 6.2:** Partition- und Indexerzeugung: $buildIndex(\mathcal{R}, directory, primarykey)$

Der Algorithmus 6.2 erzeugt die Partitionen für das jeweilige Eingabetupel. Als Eingabe erhält der Algorithmus eine zurückverfolgte Eingaberelation, das Verzeichnis der Eingaberelation und ein Primärschlüsselattribut. Der Algorithmus erzeugt zunächst einen leeren Index in Zeile 1. In Zeile 2 wird mittels einer load-Funktion, die Eingaberelation geladen, für die der Index erstellt werden soll. Dazu wird der *directory*-Parameter verwendet. Dieser Parameter ist der absolute Dateipfad zu der Eingaberelation $\mathcal{R}'$. Die geladene Eingaberelation $\mathcal{R}'$ wird in Zeile 3 mit einem *PARTITION*-Attribut erweitert. Das Attribut beinhaltet den vollständigen Partitionspfad, in dem sich ein Tupel aus der geladenen Eingabelrelation $\mathcal{R}'$ befindet. Anschließend wird in Zeile 4 die zurückverfolgte Eingaberelation $\mathcal{R}$ mit der geladenen Eingaberelation $\mathcal{R}'$ mittels dem Parameter *primarykey* verbunden. Dieser Parameter ist der Primärschlüssel in beiden Relationen. Der Primärschlüssel in der zurückverfolgten Eingaberelation $\mathcal{R}$ hat einen anderen Namen. Für die Join-Operation, wird deshalb die Bezeichnung des Parameters *primarykey* benötigt. Das Ergebnis ist ein Index mit allen Eingabe- und Ergebnistupel für eine Relation, sowie dem absoluten Dateipfad der Partition pro Tupel.

Tabelle 6.1 zeigt den Index für die *Orders*-Tabelle aus dem *Szenario* 1.2 . Die Tabelle besteht aus einem *OUTPUT* Attribut und einem *INPUT*-Attribut. Beide Attribut verweisen auf das jeweilige Ergebnis- und Eingabetupel. Das *PARTITION*-Attribut ist der vollständige Pfad zu der Partition in der sich das Eingabetupel befindet.

**Tabelle 6.1:** Instanz einer Orders-Tabelle

| OUTPUT | INPUT | PARTITION |
|---|---|---|
| 1 | A | file:///PARTITION_1.parquet |
| 2 | A | file:///PARTITION_1.parquet |
| 1 | B | file:///PARTITION_3.parquet |
| 3 | C | file:///PARTITION_4.parquet |
| 4 | D | file:///PARTITION_2.parquet |

#### Identifikation der betroffenen Eingabetupel

Nach der Indexerzeugung und der Partitionierung der Daten kann der Index für die effiziente Identifikation der betroffenen Eingabetupel verwendet werden. Die Identifikation erfolgt dabei in zwei Schritten und wurde in Kapitel 5 detailliert beschrieben. Zunächst werden betroffene Ergebnistupel ermittelt und daraus alle betroffenen Eingabetupel. Die Eingabetupel verweisen dabei

auf eine Partition, die geladen werden muss. Ein weiterer Schritt muss jedoch getan werden, bevor die Daten für eine Neuberechnung verwendet werden können. Da eine Partitionsdatei potentiell über mehrere Eingabetupel verfügt, müssen diese vorher noch gefiltert werden, sodass nur die betroffenen Eingabetupel berechnet werden. Der folgende Pseudocode zeigt die Funktionsweise der Identifikation.

**Input:** $C, primarykey$
**Data:** Index $\mathcal{I}$
**Data:** Index $\mathcal{I}_C$
**Result:** Partitionen $P$

```
1 P ← ∅
2 outputIds ← C ⋈_{primarykey=I_C.INPUT} I_C;
3 R' ← outputIds ⋈_{OUTPUT=I_C.OUTPUT ∧ IN≠I_C.IN} I_C;
4 append(P, R');
5 for I_R ∈ I \ I_C do
6 |   R' ← outputids ⋈_{OUTPUT=I_R.OUTPUT};
7 |   append(P, R');
8 return P;
```

**Algorithmus 6.3:** Identifikation-Funktion: $identify(C, primarykey)$

Mittels des Algorithmus 6.3 werden die betroffenen Partitionen bzw. betroffenen Eingabetupel identifiziert. Der Algorithmus erwartet eine Relation der geänderten Eingaberelation $C$, sowie den Primärschlüsselattribut dieser Relation. In Zeile 1 wird zunächst eine leere Menge mit allen Partitionen erzeugt. In Zeile 2 werden alle Ergebnistupel bzw. der Primärschlüssel der Ergebnistupel geladen. Dies wird erreicht in dem der Index $\mathcal{I}_C$ mit der Ergebnisrelation $C$ verbunden wird. Dies geschieht durch den Primärschlüssel $primarykey$ der geänderten Eingaberelation und durch den Fremdschlüssel des $\mathcal{I}_C$ in Zeile 2. In Zeile 3 können nun durch den Index alle betroffenen Eingabetupel $C$ identifiziert werden. Damit aber nur betroffenen Eingabetupel identifiziert werden und nicht geänderte Eingabetupel, werden alle geänderten Eingabetupel herausgefiltert. Das Ergebnis ist eine Teil der Eingaberelation mit den Pfaden zu den betroffenen Partitionen, die der Menge der Partitionen $P$ in Zeile 4 hinzugefügt werden. In den Zeile 5-7 passiert der selbige Ablauf wie in Zeile 3. Es werden alle Indizes, mit Ausnahme des bereits verwenden Index $\mathcal{I}_C$ für die Identifikation der betroffenen Partitionen bzw. Eingabetupel verwendet und den Partitionen $P$ hinzugefügt.

Das Ergebnis kann nun verwendet werden, um alle Partitionen zu laden und um eine personenbezogenen Analyse erneut durchzuführen. Der Algorithmus 6.4 beschreibt die Logik zum Laden der Partitionen.

**Input:** $P$, primarykey
**Result:** $\mathcal{R}'$
1 $\mathcal{R}' \leftarrow \emptyset$;
2 $files \leftarrow \pi_{PARTITION}(P)$;
3 $\mathcal{R}' \leftarrow load(files)$;
4 $\mathcal{R}' \leftarrow \mathcal{R}' \bowtie_{primarkey=P.INPUT} P$;
5 return $\mathcal{R}'$;

**Algorithmus 6.4:** Laden einer Partition: $load(P, primarykey)$

Algorithmus 6.4 zeigt die Logik für das Laden einer Partition, die durch den Algorithmus 6.3 identifiziert wurde. Der Algorithmus erwartet als Eingabe eine Relation $P$ und den *primarykey* der zu ladenden Eingaberelation. Die Relation $P$ enthält als Tupel alle Partitionen und den Primärschlüssel der Eingaberelation, die vom Index geladen wurden. In Zeile 2 werden alle Dateipfade der Relation $P$ selektiert. Mit dieser Selektion und der load-Funktion werden Teildaten in Zeile 3 geladen. Da die geladenen Eingaberelation $\mathcal{R}'$ mehr Eingabetupel enthält als eigentlich geändert wurden, werden diese in Zeile 4 gefiltert, sodass die geladenen Eingaberelation $\mathcal{R}'$ anschließend nur alle betroffenen Eingabetupel enthält.

# 7 Evaluation

In diesem Kapitel werden die Ergebnisse der Evaluation bezüglich des Konzeptes und der Implementierung vorgestellt. Zunächst wird die Workload vorgestellt, die für das Testen der Implementierung verwendet wurde, gefolgt vom verwendeten Setup. Die nächsten vier Abschnitte befassen sich anschließend mit den konkreten Ergebnissen der Evaluation. Dabei werden zunächst die Ergebnisse der Ausführung von personenbezogenen Anfragen unter der Verwendung des DS-GVO konformen Datenmodell im Vergleich zu einer Ausführung ohne DS-GVO konformen Datenmodell vorgestellt, gefolgt von den Ergebnissen der Laufzeitevaluation jeweils mit und ohne Index. Der letzte Abschnitt beschreibt die Laufzeitergebnisse nach Optimierung der Ausführungspläne von personenbezogenen Anfragen.

## 7.1 Workload und Setup

Für die Evaluation der Implementierung wurden zwei Datensätze verwendet. Darunter sind die bereits vorgestellte und synthetisch erzeugten Daten aus dem TPC-H Benchmark. Zusätzlich dazu wurden Echtdaten aus DBLP[1] verwendet. Der DBLP-Datensatz verfügen dabei über insgesamt vier Tabellen, die hauptsächlich Daten über Autoren, Konferenzen, Artikel und Publikationen aufweisen. Die Datensätze wurden dabei von 100 GB, 300 GB bis 500 GB skaliert. Pro Datensatz wurden dabei jeweils 5 verschiedene Anfragen definiert, die in den folgenden beiden Abschnitten detailliert beschrieben werden.

### 7.1.1 TPC-H

Insgesamt wurden fünf verschiedene TPC-H Anfragen definiert. Diese sind in den folgenden Abschnitt aufgelistet. Die Anfrage werden dabei jeweils mit H1 bis H5 bezeichnet.

#### H1

Szenario *H1* ist eine personenbezogene Anfrage, die Kunden aus dem Marktsegment *AUTOMOBILE* ausgibt. Die Anfrage liefert alle Kunden aus der *Customer*-Tabelle mit Namen, Adresse, Telefonnummer und Marktsegment zurück, nachdem auf das Marktsegment *AUTOMOBILE* gefiltert wurde. Der Operatorenbaum in Abbildung 7.1 zeigt die Anfrage in relationaler Algebra.

[1] https://dblp.uni-trier.de/

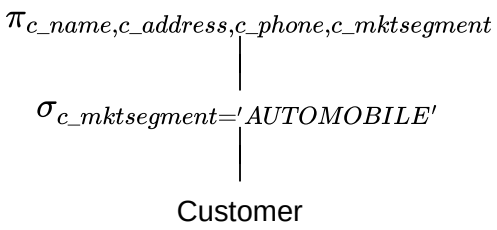



**Abbildung 7.1:** Operatorenbaum: H1

#### H2

Szenario *H2* ist eine personenbezogene Anfrage, die den Umsatz eines Kunden berechnet. Dazu werden die Tabellen *Customer* und *Orders* mittels den Attributen *c_custkey* und *o_custkey* verbunden. Das Ergebnis der Vereinigung wird anhand des *o_custkey* aus der *Orders*-Tabelle gruppiert und der Bestellpreis *o_totalprice* aufsummiert. Der Operatorenbaum in Abbildung 7.2 zeigt die Anfrage in relationaler Algebra.

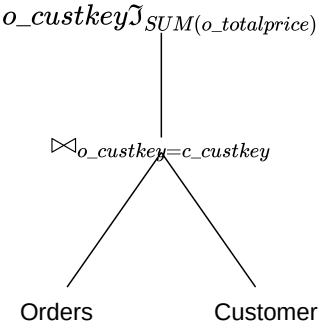



**Abbildung 7.2:** Operatorenbaum: H2

#### H3

Szenario *H3* ist eine personenbezogene Anfrage, die den Gesamtverlust von retournierten Artikeln pro Kunde aufsummiert. Dazu werden die *Customer*-Tabelle und *Nation*-Tabelle mittels den Attributen *c_nationkey* und *n_nationkey* verbunden. Die *Orders*-Tabelle wird dabei gefiltert, sodass nur Bestellungen verwendet werden, die zwischen 1993-10-01 und 1994-01-01 getätigt wurden. Die *Lineitem*-Tabelle wird gefiltert, sodass diese nur noch retournierte Posten enthält. Die beiden gefilterten Tabellen *Orders* und *Lineitem* werden anschließend mittels den Attributen *o_orderkey* und *l_orderkey* verbunden. Dieses Zwischenergebnis wird wiederum mit den Zwischenergebnis aus dem Verbund von *Customer* und *Nation* mittels den Attributen *o_custkey* und *c_custkey* verbunden. Aus dem Zwischenergebnis werden die folgenden Attribute selektiert: Primärschlüssel der *Customer*-Tabelle, Kundenname, Postenpreis, Kontostand des Kunden, Land in dem der Kunde sich befindet, Adresse des Kunden, Telefonnummer des Kunden und Kommentare die vom Kunden geschrieben wurden. Anhand dieser Attribute findet eine Gruppierung statt, sodass der Postenpreis *l_extendedprice* aufsummiert werden kann. Das Ergebnis der Summierung wird aufsteigend sortiert. Der Operatorenbaum in Abbildung 7.3 zeigt die Anfrage in relationaler Algebra.

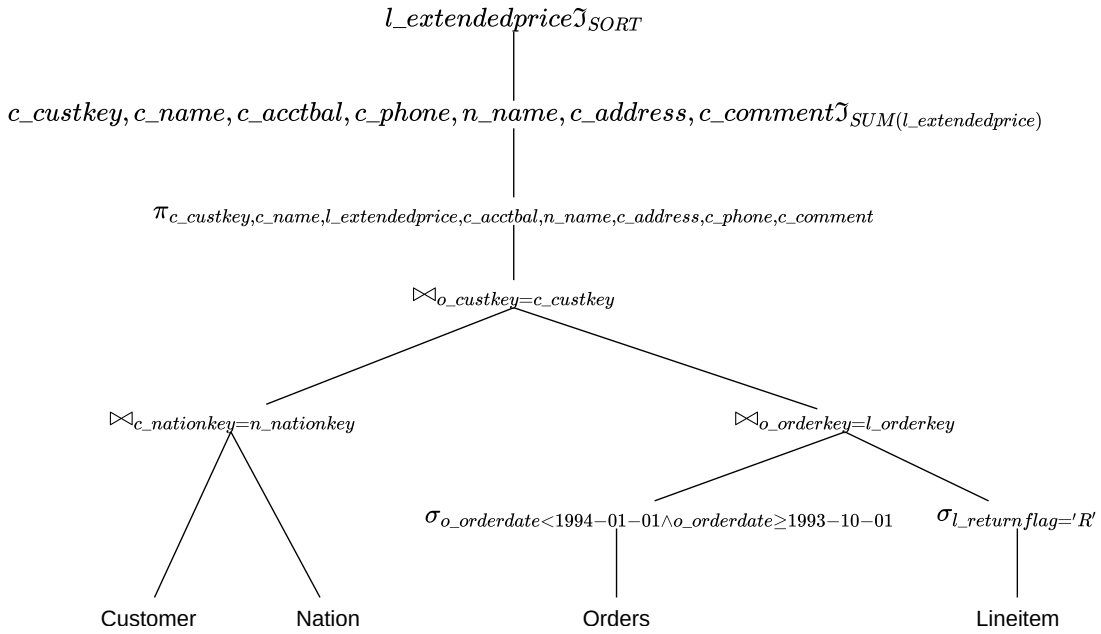



**Abbildung 7.3:** Operatorenbaum: H3

#### H4

Szenario *H4* ist eine personenbezogene Anfrage, die alle Produkte eines Lieferanden zählt und den Verkaufspreis aller Produkte des Lieferanden aufsummiert. Dazu werden die Tabellen *Supplier* und *PartSupps* miteinander durch die Attribute *s_suppkey* und *ps_suppkey* verbunden. Dieses Zwischenergebnis wird dann mit der *Part*-Tabelle mittels der Attribute *p_partkey* und *ps_partkey* verbunden. Anschließend wird das Zwischenergebnis gruppiert. Die Gruppierung erfolgt durch den Namen des Lieferanden. Anhand dieser Gruppierung werden die Produkte mittels des Produktnamen gezählt und ihre Verkaufspreis aufsummiert. Der Operatorenbaum in Abbildung 7.4 zeigt die Anfrage in relationaler Algebra.

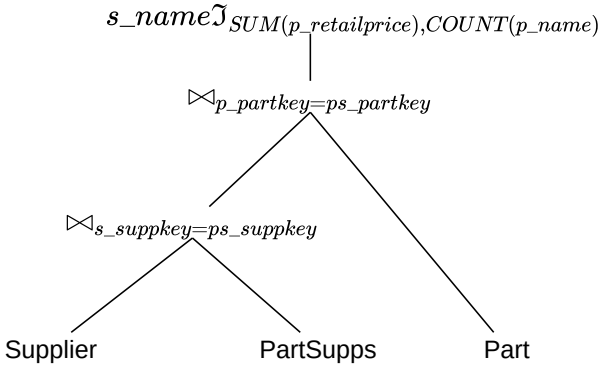



**Abbildung 7.4:** Operatorenbaum: H4

#### H5

Szenario *H5* ist eine personenbezogene Anfrage, die den Kontostand aller Lieferanden und Kunden eines Landes aufsummiert. Für die Vereinigung werden jeweils die Attribute Name, Adresse, Telefonnummer, Fremdschlüssel des Landes, und Kontostand ausgewählt. Danach werden die *Customer* und *Supplier*-Tabelle vereinigt. Dieses Zwischenergebnis wird gefiltert, sodass nur Kunden und Lieferanden im Zwischenergebnis vorhanden sind, die einen positiven Kontostand haben. Danach wird das gefilterte Zwischenergebnis mittels des Fremdschlüssel des Landes gruppiert und die Kontostände aufsummiert. Der Operatorenbaum in Abbildung 7.5 zeigt die Anfrage in relationaler Algebra.

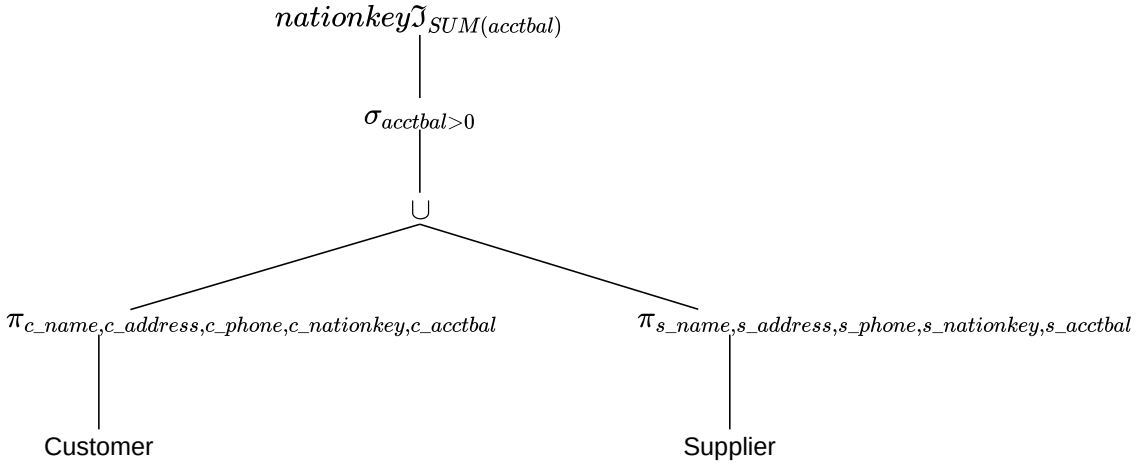



**Abbildung 7.5:** Operatorenbaum: H5

### 7.1.2 DBLP

Analog zu den TPC-H Anfragen, wurden für den DBLP-Datensatz fünf Anfragen definiert. Sie werden dabei als D1 bis D5 bezeichnet. Der DBLP-Datensatz beinhaltet Informationen über Artikel, Autoren, Homepage und Konferenzen und teilt sich in mehrere Tabellen auf [Ley09]. Die Anfragen verwenden die vier folgenden Tabellen aus dem DBLP-Datensatz, die freundlicherweise von Diestelkämper und Herschel [DH20] zur Verfügung gestellt wurden.

**Article** Informationen über einen einzelnen Artikel bspw. in einem Journal. Besteht aus Titel, Autoren und Jahr, sowie weitere Attribute.

**WWW** Informationen über Homepages von Autoren. Besteht aus Autoren und Datum, sowie weitere Attribute.

**Inproceedings** Informationen über ein Konferenzbeitrag. Besteht aus Namen, Autoren, Datum und einer Referenz zur zugehörigen Konferenz.

**Proceedings** Informationen über Konferenz. Beinhaltet den Konferenznamen, Editoren und Jahr.

Die DBLP-Daten sind hierbei geschachtelte JSON-Daten. Apache Spark bietet für diesen Zweck gesonderte Funktionen an, wie etwa *explode*. Explode ermöglicht das Abflachen von JSON-Listen. Das Abflachen sorgt, dafür das für jedes Element in der Liste ein zusätzliches Tupel erzeugt wird. Diestelkämper und Herschel [DH20] beschreiben in ihrer Arbeit die Ausführungssemantik eines *flatten*-Operators, der die Abflachung von Listen ermöglicht. Dieser Operator wird in den Operatorenbäumen der Anfragen verwendet.

#### D1

Szenario *D1* ist eine personenbezogene Anfrage, die alle Artikel eines Autors aufsummiert. An einem Artikel können mehrere Autoren mitwirken. Dies wird in der *Article*-Tabelle als Liste dargestellt. Damit die Autoren gezählt werden können, wird die Liste abgeflacht. Das Zwischenergebnis wird anschließend anhand des Autorennamen gruppiert und das Vorkommen des Autorennamen gezählt. Nach der Zählung werden alle Autoren herausgefiltert, die an nur einem Artikel mitgewirkt haben. Der Operatorenbaum in Abbildung 7.6 zeigt die Anfrage in relationaler Algebra.

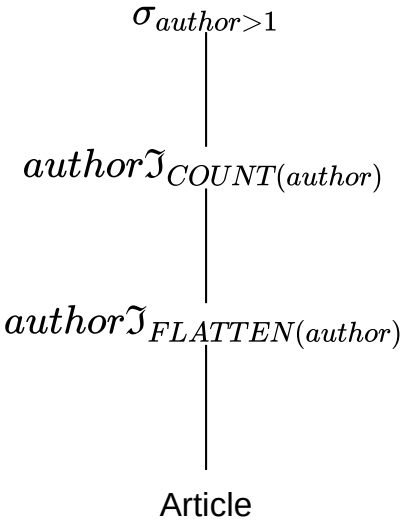



**Abbildung 7.6:** Operatorenbaum: D1

#### D2

Szenario *D2* ist eine personenbezogene Anfrage, die alle Konferenzbeiträge für eine Konferenz zählt. Ein Konferenzbeitrag kann in mehreren Konferenzen vorkommen. Die Verweise auf Konferenzen werden in der *Inproceedings*-Tabelle dabei als Liste im Attribut *crossref* gespeichert. Deshalb wird die Liste abgeflacht und im neuen Attribut *cref* dargestellt. Danach werden alle Konferenzbeiträge gefiltert, sodass nur Beiträge in Betracht gezogen werden, die im Jahr 2000 und in den nachfolgenden Jahren geschrieben wurden. Nach der Filterung wird der Titel des Konferenzbeitrages und der Verweis, auf die jeweilige Konferenz ausgewählt. Der Konferenzbeitrag wird mit den Konferenz

anhand der Attribute *crf* und *_key* verbunden. Das Zwischenergebnis wird mittels des Konferenztitels und des Jahres gruppiert und alle Titel der Konferenz werden gezählt. Der Operatorenbaum in Abbildung 7.7 zeigt die Anfrage in relationaler Algebra.

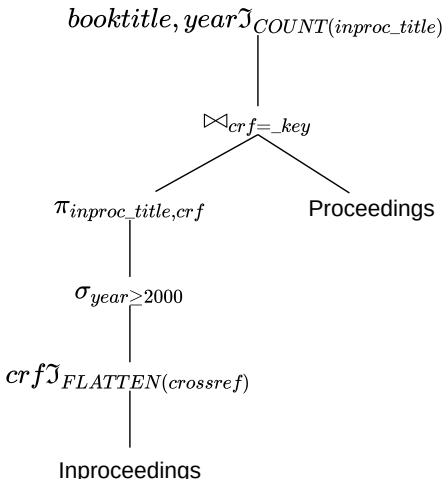



**Abbildung 7.7:** Operatorenbaum: D2

#### D3

Szenario *D3* ist eine personenbezogene Anfrage, die alle Autoren eines Konferenzbeitrags zählt, die über eine Homepage verfügen. Dazu werden alle Konferenzbeiträge gefiltert, die nicht im Jahr 2015 herausgebracht wurden. Die *Inproceedings* und *WWW*-Tabellen verfügen beide über ein Autoren-Attribut. Das Attribut ist eine Liste von Autoren, sodass für beide Tabellen zunächst die Liste abgeflacht wird. Das, nun flache, Autor-Attribut und das Titel Attribut wird für die Konferenzbeiträge ausgewählt. Für die abgeflachte *WWW*-Tabelle wird nur das Autor-Attribut benötigt. Anhand des Autor-Attributes werden beide Tabellen miteinander verbunden. Das Zwischenergebnis wird nach Titel gruppiert und die Autoren pro Konferenzbeitrag gezählt. Der Operatorenbaum in Abbildung 7.8 zeigt die Anfrage in relationaler Algebra.

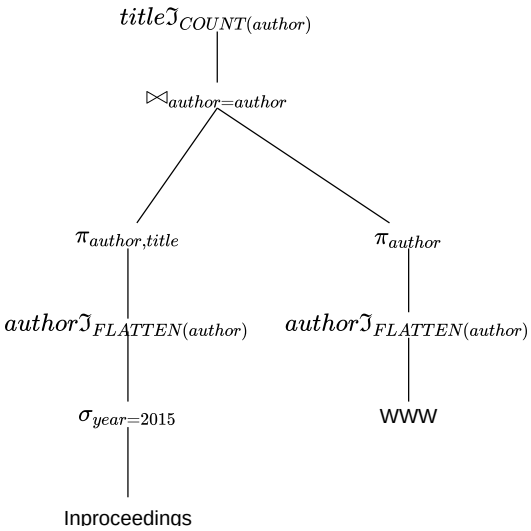



**Abbildung 7.8:** Operatorenbaum: D3

#### D4

Szenario *D4* ist eine personenbezogene Anfrage, die alle Konferenzbeiträge für eine Konferenz in einer Liste sammelt. Es werden nur Konferenzbeiträge in Betracht gezogen, die im Jahre 2015 oder später herausgebracht wurden. Die Verweise eines Konferenzbeitrages werden im Attribut *crossref* gespeichert. Dieses Attribut wird abgeflacht, da es sich um eine Liste von Verweisen handelt. Anschließend wird das Verweis-Attribut und eine Struktur aus den Attributen Verweis, Autor, Titel und Datum ausgewählt. Eine Struktur ist dabei eine Liste, die mehrere Attribute zusammenfasst. Diese Struktur stellt dabei selbst ein Attribut da. Die *Inproceedings*-Tabelle wird anschließend mit der *Proceedings*-Tabelle mittels der Attribute *crf* und *_key* verbunden. Der Operatorenbaum in Abbildung 7.9 zeigt die Anfrage in relationaler Algebra.

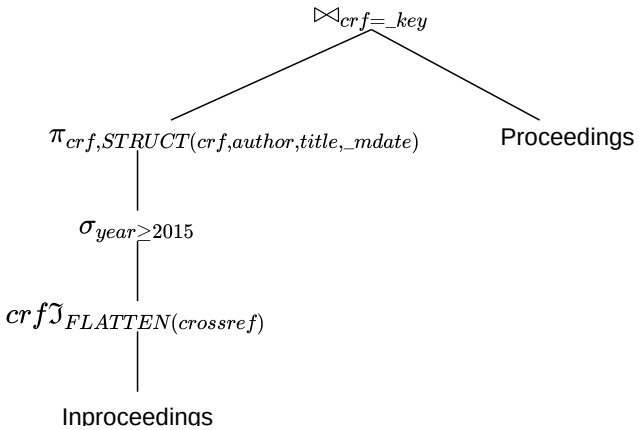



**Abbildung 7.9:** Operatorenbaum: D4

#### D5

Szenario *D5* ist eine personenbezogene Anfrage, die alle Konferenzbeiträge und Artikel im Jahr 2015 und später zählt, an denen ein Autor mitgewirkt hat. Dazu werden alle Konferenzbeiträge und Artikel herausgefiltert, die früher als 2015 herausgebracht wurden. Anschließend werden in beiden Tabellen, die Autoren abgeflacht, da jeder Artikel und jeder Konferenzbeitrag über mehrere Autoren verfügt, die als Liste abgebildet werden. Anschließend werden für beide Tabellen nur der Autor und Titel ausgewählt. Dies ist notwendig, um eine Vereinigung beider Tabellen durchzuführen. Das Ergebnis dieser Vereinigung wird anschließend anhand des Autorennamen gruppiert und alle veröffentlichte Titel gezählt. Der Operatorenbaum in Abbildung 7.10 zeigt die Anfrage in relationaler Algebra.

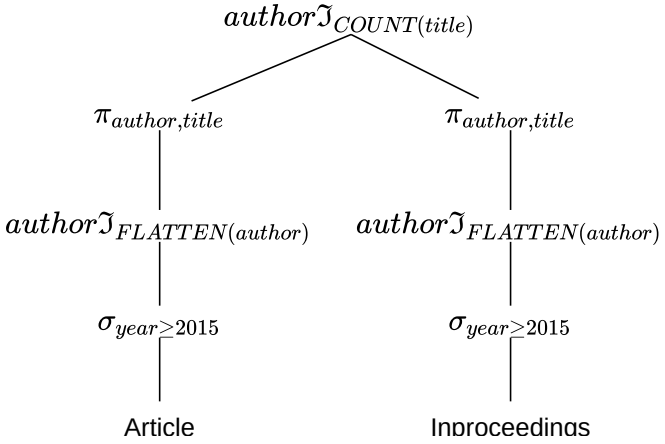



**Abbildung 7.10:** Operatorenbaum: D5

### 7.1.3 Setup

Die Evaluation wurde auf einen Apache-Spark Cluster mit insgesamt sechs Knoten durchgeführt. Jeder der Knoten verfügt dabei über 180GB an Arbeitsspeicher und jeweils 20 Kerne, sowie SSD-Speicher. Die verwendete Software ist dabei Apache Spark 2.4, Hadoop 3.2 und Scala 2.11. Die Konfiguration für den Durchlauf einer Berechnung bestand dabei aus 22 Executors mit jeweils 5 Kernen, sowie 80GB Arbeitsspeicher. Für das Driver-Programm wurden 30GB an Arbeitsspeicher reserviert.

Die Datensätze wurden mittels Apache Spark in gleich große Partitionen aufgeteilt. Entsprechend ihrer Größe sind es 10000 Partitionen bei 100GB, 30000 Partitionen bei 300GB und 50000 Partitionen bei 500GB. Wenn nicht anders beschrieben, werden die Mittelwerte von drei Läufen dargestellt. Die Ergebnisse der Berechnungen wurden dabei auf die Festplatte geschrieben, um eine vollständige Ausführung der Apache Spark Pläne zu erzwingen.

## 7.2 DS-GVO konformes Datenmodell

In diesem Abschnitt wird die Berechnungsdauer der analytischen Anfragen H1-H5 und D1-D5 unter der Berücksichtigung des DS-GVO konformen Datenmodell aus Kapitel 4 mit der Berechnungsdauer von Apache Spark ohne DS-GVO konformen Datenmodell verglichen. Abbildung 7.11 zeigt die Ausführungszeit in Sekunden (y-Achse) für jeweils die Anfragen H1-H5 und D1-D5 (x-Achse). Die einzelnen Farben kennzeichnen die Berechnungsdauer. Blaue Balken zeigen die Berechnungsdauer ohne DS-GVO und orangene Balken den Mehraufwand mittels des DS-GVO konformen Datenmodell aus Kapitel 4. Des weiteren wird der Mehraufwand in Prozent pro Balken dargestellt. Die Berechnungsdauer wird hierbei für den 100 GB Datensatz und für 100 Einwilligungsentzüge dargestellt, da keine Unterschiede bei Steigerung der Datengröße oder bei Erhöhung der Einwilligungsentzüge.

Alle Anfragen zeigen bei Verwendung des DS-GVO konformen Datenmodell eine erhöhte Berechnungsdauer. Für den TPC-H Datensatz variiert der Mehraufwand pro Anfrage, da sich hierbei die Komplexität pro Anfrage unterscheidet. H3 ist hierbei die Anfrage, bei der das DS-GVO konforme Datenmodell den größten Mehraufwand verursacht. Das liegt zu einem an der Komplexität der Anfrage, da H3 viele Operationen verwendet, und an der Größe der Einwilligungsrelation. Die Größe der Einwilligungsrelation beträgt 2 GB. In dieser Anfrage wurde die Einwilligungsrelation für die *Lineitem*-Tabelle erzeugt. Diese Tabelle ist mit 50GB, die größte Tabelle im TPC-H Benchmark

Der DBLP-Datensatz zeigt einen gleichmäßigeren Mehraufwand im Vergleich zum TPC-H Datensatz. Das liegt zu einem daran, das der Datensatz aus vier Tabellen besteht und die Anfragen eine ähnliche Komplexität im Bezug auf die Anzahl der verwendeten Transformationen haben. D5 hat den kleinsten Mehraufwand, das liegt an der Größe der verwendeten Einwilligungsrelation, die ca. 30MB groß ist. Die restliche Tabellen im DBLP-Datensatz weisen eine Größe von 60 bis 550MB auf.

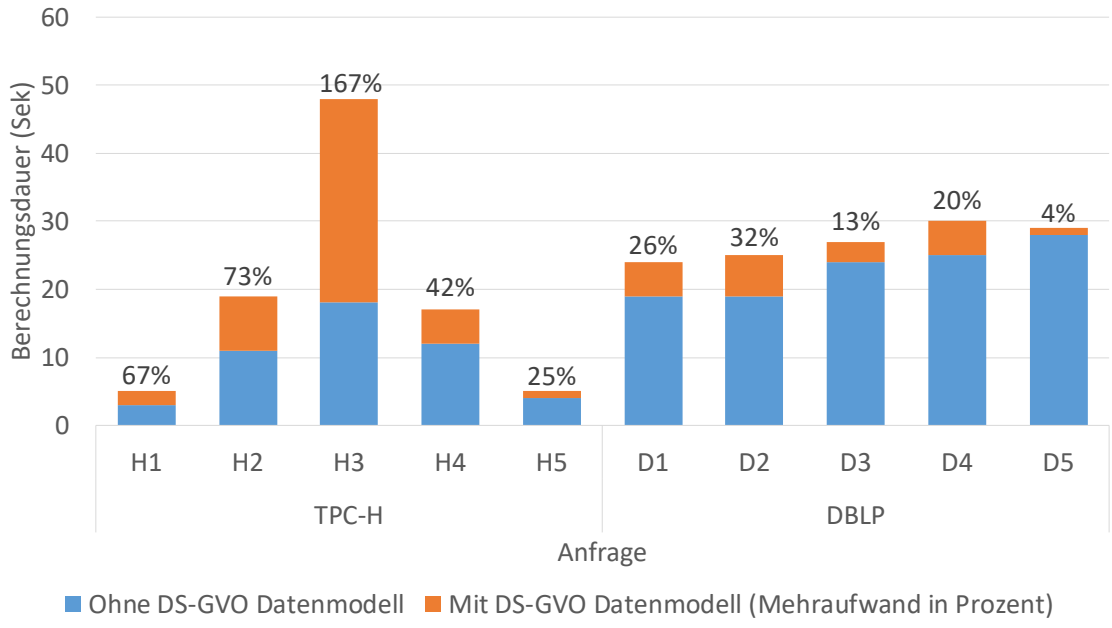



**Abbildung 7.11:** Berechnungsdauer mit und ohne DS-GVO konformes Datenmodell

Der Mehraufwand der in beiden Datensätzen vorhanden ist, wird durch die zusätzliche Filterung aus Abschnitt 4.2 erzeugt. Diese Filterung bzw. Selektion ist notwendig, damit nur DS-GVO konforme Daten in der Anfrage verwendet werden. Diese wurden in Apache Spark als Join-Operationen implementiert. Damit die Daten entsprechend mittels der Join-Operation gefiltert werden können, müssen die Daten der einzelnen Knoten im Cluster zusammengeführt werden. Diese Operation heißt *Shuffle*-Operation und ist sehr kostspielig, da hierbei die Daten der einzelnen Knoten eingelesen, sortiert und über das Netzwerk transportiert werden müssen [ApaoDc]. Je größer die Einwilligungsrelation ist, umso größer ist der Aufwand für eine *Shuffle*-Operation.

## 7.3 Index

Dieser Abschnitt befasst sich mit der Evaluation im Bezug auf die indexgestützte Neuberechnung aus Kapitel 5. Die indexgestützte Neuberechnung wurde dabei im Hinblick auf die Indexerzeugung, Berechnungsdauer bei Einwilligungsentzug, Skalierbarkeit bei steigender Datengröße, Indexgestützte Neuberechnung und Optimierung von Ausführungsplänen evaluiert. Die Ergebnisse der Evaluation werden in den folgenden Abschnitten vorgestellt und diskutiert.

### 7.3.1 Indexerzeugung

In diesem Abschnitt wird die Berechnungsdauer für die Indexerzeugung aus Kapitel 5 gezeigt. Abbildung 7.12 zeigt die Indexerzeugung für den TPC-H Datensatz und Abbildung 7.13 zeigt die Indexerzeugung für den DBLP Datensatz. Die Linien stellen dabei den jeweiligen Index für die Anfragen H1-H5 bzw. D1-D5 dar. Gemessen wurde die Indexerzeugung anhand der steigenden Datengröße (x-Achse). Die Berechnungsdauer (y-Achse) des Index wurde dabei in Minuten gemessen.

Beide Datensätze weisen einen relative linearen Verlauf der Indexerzeugung im Bezug auf die Berechnungsdauer auf. Die Zeit hängt dabei maßgeblich von der Komplexität der Anfrage ab d. h. das Anfragen die über viele Relationen und über eine komplexe Verzweigung der Operationen verfügen, deutlich mehr Zeit benötigen. Diese Komplexität wirkt auf die verwendete Provenance-Lösung ein, da dadurch das Backtracking entsprechend länger dauert. Dies ist in den Anfragen H1 und H3 deutlich zu erkennen. H1 ist eine simple Anfrage mit einer Projektion und einer Selektion, die auf eine Relation angewendet werden. H3 hingegen ist deutlich komplexer und verfügt über mehrere Joins, sowie über mehrere Selektionen, Projektionen und über eine Gruppierung. Dadurch benötigt die Anfrage H3 deutlich mehr Zeit für die Erzeugung des Index als die restlichen Anfragen.

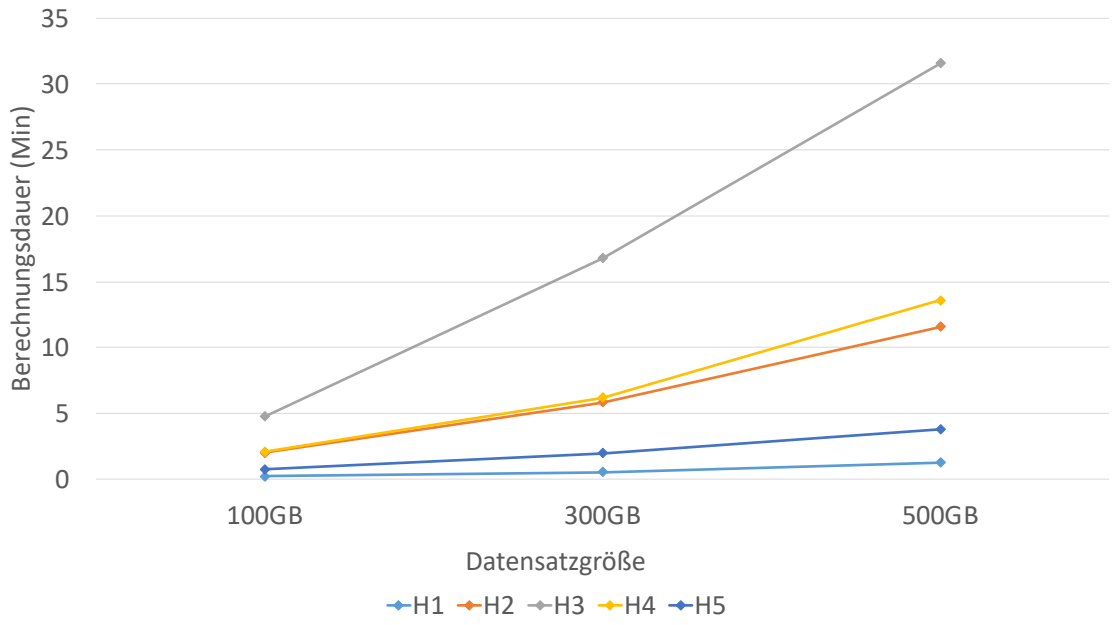



**Abbildung 7.12:** TPC-H: Berechnungsdauer für die Indexerzeugung

Die Messergebnisse für DBLP in der Abbildung 7.13 zeigen das die Berechnungsdauer der einzelnen Anfragen sich relativ ähneln. Dies liegt daran, das die Anfragen D1-D4 eine ähnliche Komplexität aufweisen. Sie verfügen jeweils über eine Selektion, Projektion und über ein Join. D5 hingegen benötigt mehr Berechnungszeit für das Erzeugen des Index. Dies liegt an dem Backtracking der Provenance-Lösung und an den vereinigten Datensatz der über das Netzwerk transportiert werden muss. Die Überschneidung von D1 und D3 ist hierbei eine Besonderheit, die nicht genauer identifiziert werden konnte. Es ist jedoch anzunehmen, dass die Überschneidung durch Apache Spark bzw. durch den verwendeten Cluster verursacht wurde. Eine Veränderung der ausgeführten Pläne konnten für die zunehmende Datensatzgröße nicht festgestellt werden.

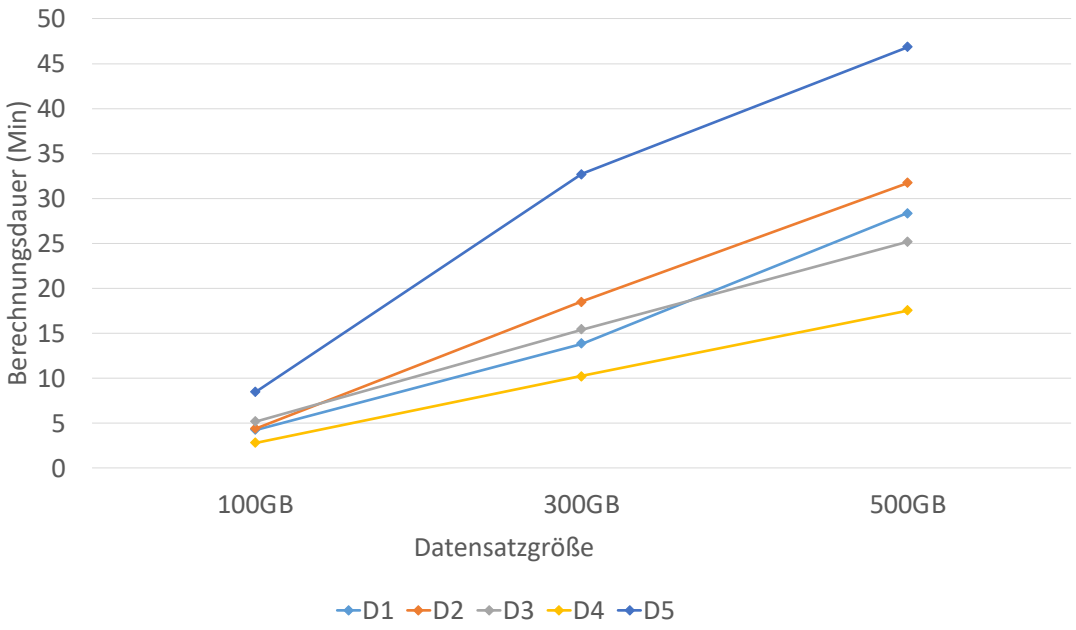



**Abbildung 7.13:** DBLP: Berechnungsdauer für die Indexerzeugung

Die Berechnungsdauer hängt von der Komplexität der Anfrage, der Größe des Datensatzes und von der Berechnung der Provenance ab. Diese Zeiten sind jedoch vertretbar, da die initiale Berechnung des Index nur einmalig durchgeführt werden muss. Der Großteil der Berechnungsdauer ist zurückzuführen, auf die Berechnung der Provenance-Daten, die für die Erzeugung des Index benötigt werden. Die Implementierung der Provenance-Lösung war nicht Teil dieser Arbeit und wurde in Kapitel 3 anhand verschiedener Kriterien ausgewählt. Eine schnellere Provenance-Lösung kann dazu führen, das der Index entsprechend schneller erzeugt wird.

### 7.3.2 Berechnungsdauer bei Einwilligungsentzug

In diesem Abschnitt wird die Berechnungsdauer der indexgestützten Berechnung aus Kapitel 5 gezeigt. Dazu werden die einzelnen Anfragen für den TPC-H Datensatz in Abbildung 7.14 und für den DBLP Datensatz in Abbildung 7.15 dargestellt. Die y-Achse zeigt die Berechnungsdauer in Sekunden, während die x-Achse die Datengröße anzeigt. Die farbliche markierten Linien sind die Anfragen H1-H5 und D1-D5. Zusätzlich werden die eingelesenen Partitionen in Abhängigkeit von der Anzahl der Einwilligungsentzüge in den Tabellen 7.1 und 7.2 für die Datensätze TPC-H und DBLP angezeigt. Die Tabellen zeigen dabei die jeweilige Anfrage und die Tabellen die in der Anfrage verwendet wurden, sowie die Anzahl der eingelesene Partitionen pro Einwilligungsentzug. Die folgenden Daten basieren hierbei auf den 100GB Datensatz, da die Daten eine lineare Skalierung bei steigender Datengröße aufweisen. Die Skalierung wird in Abschnitt 7.3.3 diskutiert.

Für beide Datensätze ist ein deutlicher Zusammenhang zwischen Berechnungsdauer und eingelesener Partition zu sehen. Bei steigender Anzahl an Partitionen, steigt die Berechnungsdauer an. H5 liest dabei für alle Einwilligungsentzüge, sämtliche Partitionen ein. Da jedoch die Filterung der Daten in den Partitionen vorgenommen werden muss, steigt die Berechnungsdauer in Abhängigkeit zu

den Einwilligungsentzügen an. Die Anfrage H3 profitiert von der indexgestützten Berechnung am meisten, da hier selbst bei einem Entzug von 1000 Einwilligungen die Anzahl der Partitionen unter 1500 bleibt. H2 und H4 profitieren ebenfalls von der indexgestützten Berechnung, solange die Anzahl der einzulesenden Partitionen gering bleibt. Für H2 werden bereits ab 500 Einwilligungsentzüge mehr als 50% der Partitionen für die *Customer*-Tabelle benötigt. Für H4 hingegen werden ab 100 Einwilligungsentzügen bereits 50% der Partitionen für die Tabellen *Part* und *PartSupp* geladen. Ab 1000 Einwilligungsentzüge laden beiden Anfragen 80-99% der Partitionen, wobei die Anfrage H2 nur ca 9.5% der Partitionen für die *Customer*-Tabelle einliest. Die Anfrage H1 ist hierbei ein Sonderfall, für die keine Partitionen eingelesen werden müssen, da die Anfrage aus einer Selektion und Projektion besteht. Das Ergebnis ist entsprechend eine 1:1 Beziehung von der Eingabe zur Ausgabe. Bei einem Einwilligungsentzug in H1 wird entsprechend nur das betroffene Ergebnistupel mittels dem Index identifiziert. Dieses kann genutzt werden, um das jeweilige Tupel aus der Ergebnisrelation zu entfernen. Bei zunehmender Anzahl an Entzügen ist hier ebenso ein zunehmender Aufwand für die Identifizierung festzustellen.

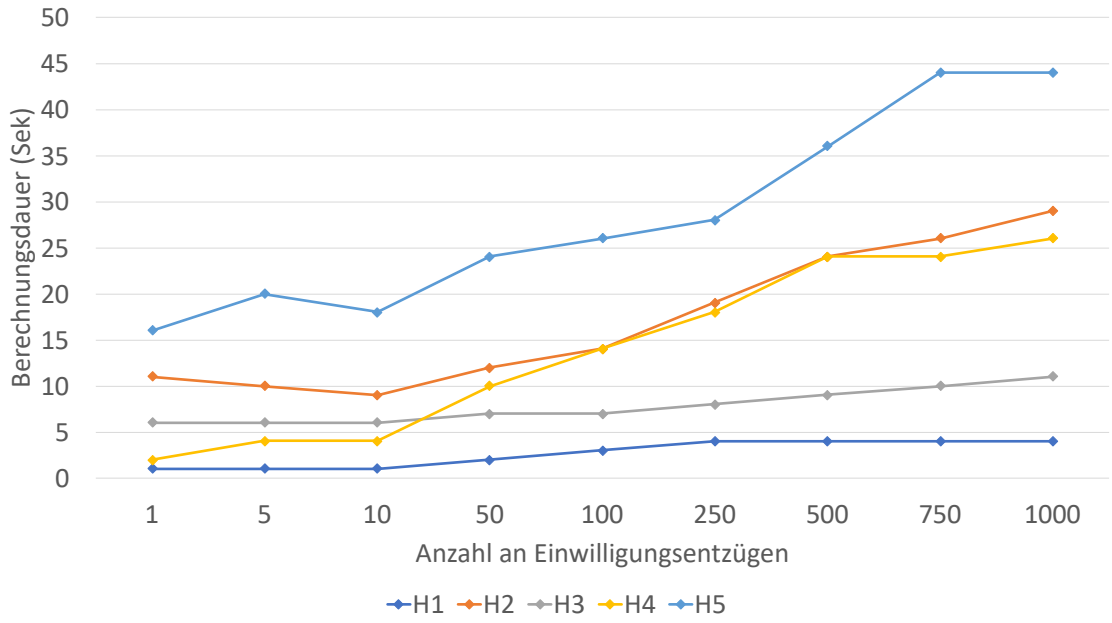



**Abbildung 7.14:** TPC-H 100GB: Berechnungsdauer bei zunehmenden Einwilligungsentzügen

| Anfrage | Tabelle | Anzahl der Änderungen | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 5 | 10 | 50 | 100 | 250 | 500 | 750 | 1000 |
| H1 | Customer | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H2 | Orders | 11 | 93 | 187 | 781 | 1544 | 3412 | 5593 | 7044 | 8067 |
| | Customer | 1 | 5 | 10 | 50 | 100 | 249 | 484 | 720 | 950 |
| H3 | Customer | 1 | 2 | 4 | 23 | 41 | 103 | 206 | 307 | 422 |
| | Orders | 1 | 2 | 5 | 27 | 47 | 85 | 142 | 202 | 259 |
| | Lineitem | 3 | 4 | 12 | 58 | 127 | 310 | 633 | 967 | 1276 |
| | Nation | 1 | 2 | 3 | 19 | 23 | 24 | 25 | 25 | 25 |
| H4 | Supplier | 8 | 41 | 81 | 408 | 785 | 1844 | 3311 | 4548 | 5543 |
| | Part | 80 | 394 | 769 | 3308 | 5492 | 8632 | 9817 | 9979 | 9995 |
| | Partsupp | 79 | 388 | 763 | 3246 | 5466 | 8655 | 9800 | 9974 | 9999 |
| H5 | Supplier | 9723 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 |
| | Customer | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 | 10000 |

**Tabelle 7.1:** TPC-H 100GB: Anzahl der Partitionen in Abhängigkeit von der Anzahl der Einwilligungsentzüge pro Eingabetabelle

Ein ähnlicher Zusammenhang wie bei TPC-H ist im DBLP-Datensatz zu beobachten. Mit steigender Anzahl an einzulesenden Partitionen, steigt die Berechnungsdauer. D4 profitiert von allen Anfragen aus dem DBLP Szenario, am meisten von der indexgestützten Neuberechnung, da hier selbst bei einem Einwilligungsentzug von 1000 nur ca 9.5% der Partitionen von der Inproceedings-Tabelle und 0.2% der Partitionen der Proceedings-Tabelle geladen werden. Für D2 müssen bei einem Entzug von 50 und 100 Einwilligungen gleich viele Partitionen geladen werden, so dass die Zeit sich hierbei kaum unterscheidet. Außerdem zeigt diese Anfrage einen sprunghaften Anstieg der Partitionen bei 100 Einwilligungsentzügen, so dass ca 95% der Partitionen für die *Inproceedings*-Tabelle geladen werden müssen. Dies hängt mit der künstlichen Skalierung der Daten zusammen. Diese Verhalten lässt ist auch in der Anfrage D1 zu erkennen. Es findet ein deutlicher Anstieg in der Anzahl der einzulesenden Partitionen ab 50 Einwilligungsentzügen statt. Bei 750 und 1000 Einwilligungen werden fast sämtliche Partitionen geladen, sodass die Zeiten sich kaum unterscheiden. D3 und D4 zeigen einen flachen Anstieg der Berechnungsdauer pro Einwilligungsentzug, da beide Anfragen erst ab 500 und 100 Einwilligungsentzüge mehr als 10% der Partitionen pro Tabelle einlesen. D5 zeigt einen zunehmenden Anstieg an einzulesenden Partitionen bei zunehmender Anzahl an Einwilligungsentzügen. Somit steigt die Berechnungsdauer für diese Anfrage.

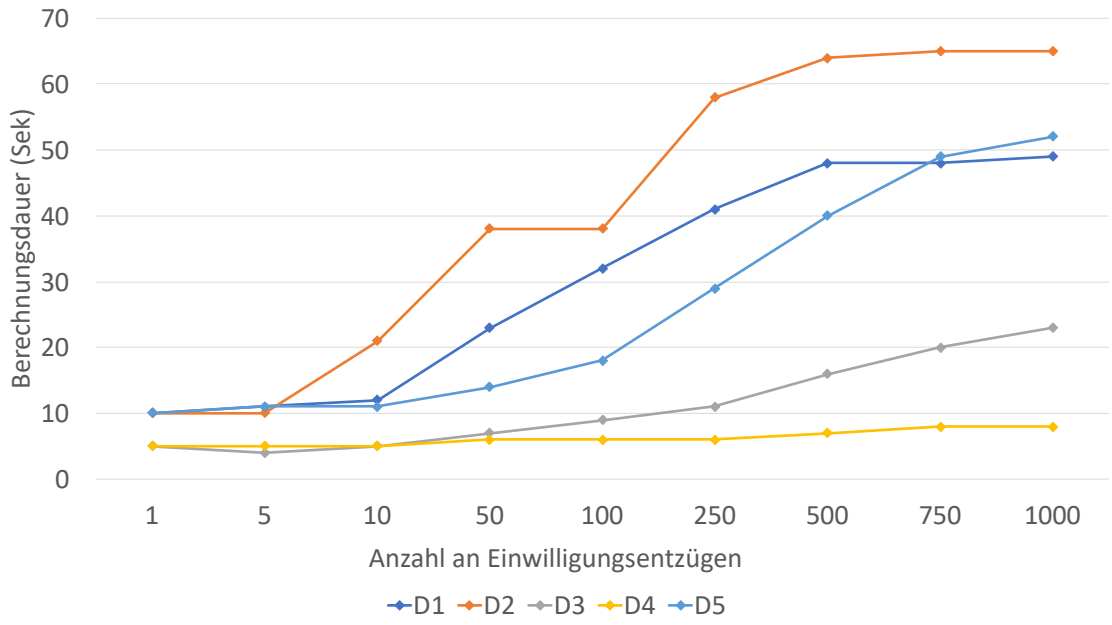



**Abbildung 7.15:** DBLP 100GB: Berechnungsdauer bei zunehmenden Einwilligungsentzügen

| **Anfrage** | **Tabelle** | **Anzahl der Änderungen** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 5 | 10 | 50 | 100 | 250 | 500 | 750 | 1000 |
| D1 | Article | 23 | 242 | 379 | 2944 | 5464 | 8163 | 9764 | 9969 | 9994 |
| D2 | Inproceedings | 318 | 319 | 3631 | 9442 | 9442 | 9581 | 9581 | 9907 | 9961 |
| | Proceedings | 55 | 55 | 164 | 848 | 848 | 999 | 999 | 1573 | 1755 |
| D3 | WWW | 17 | 25 | 72 | 338 | 710 | 1509 | 2801 | 3908 | 4727 |
| | Inproceedings | 8 | 13 | 28 | 127 | 280 | 631 | 1179 | 1736 | 2229 |
| D4 | Proceedings | 1 | 2 | 2 | 4 | 4 | 10 | 16 | 20 | 26 |
| | Inproceedings | 39 | 68 | 68 | 111 | 111 | 374 | 583 | 731 | 955 |
| D5 | Article | 44 | 145 | 213 | 884 | 1476 | 3405 | 5881 | 7329 | 8092 |
| | Inproceedings | 2 | 22 | 57 | 424 | 874 | 2857 | 4885 | 6264 | 7251 |

**Tabelle 7.2:** DBLP 100GB: Anzahl der Partitionen in Abhängigkeit von der Anzahl der Einwilligungsentzüge pro Eingabetabelle

In diesem Abschnitt wurde gezeigt, dass ein positiver Zusammenhang zwischen der Berechnungsdauer und der Anzahl der eingelesen Partitionen besteht. Bei zunehmenden Einwilligungsentzügen steigt die Anzahl an betroffenen Eingabetupeln und somit die Anzahl der einzulesenden Partitionen, sowie der Aufwand, der benötigt wird, um die Partitionen entsprechend zu filtern. Diese Filterung der Partitionen ist notwendig, um nur die Tupel herauszufiltern, die auch tatsächlich von einem Entzug einer Einwilligung betroffen sind.

### 7.3.3 Skalierbarkeit bei steigender Datengröße

In diesem Abschnitt wird die Skalierbarkeit der Berechnungsdauer bei steigender Datengröße gezeigt. Die einzelnen Anfragen aus dem TPC-H und DBLP Szenario werden dabei in den Abbildungen 7.16 und 7.17 abgebildet. Die x-Achse stellt dabei die Größe des verwendeten Datensatz dar, während die y-Achse die Berechnungsdauer in Sekunden zeigt. Die farblichen markierten Linien sind hierbei, die jeweiligen Anfragen aus den Szenarien. Zusätzlich zu den Abbildungen werden in den Tabellen 7.3 und 7.4, die eingelesenen Partitionen gezeigt. Aus Übersichtsgründen werden hierbei nur die Messdaten für 100 Einwilligungsentzüge dargestellt, da sich die Skalierung für die restlichen Einwilligungsentzüge ähnlich verhält.

H1 zeigt einen konstanten Verlauf, da hier keine Partitionen geladen werden müssen und die Berechnungsdauer für die Identifizierung der betroffenen Ergebnistupel konstant bleibt. H5 ist ebenso linear, da sämtliche Partitionen geladen werden. Diese Anfrage weist eine Vereinigung und Gruppierung, im Gegensatz zu H1 auf und ist somit komplexer. H2 zeigt einen einen geringfügigen Anstieg von ca. 3% der Partitionen pro Datensatzgröße. H4 zeigt ebenfalls einen geringfügigen Anstieg und einen Abfall der zu einzulesenden Partitionen ab 300GB für die *Supplier*-Tabelle. Das liegt an der Verteilung der Daten auf 30000 Partitionen. Bei 50000 Partitionen steigt die Anzahl der einzulesenden Partitionen, befindet sich aber immer noch unter der Anzahl der Partitionen bei 100GB. H3 zeigt eine konstante Berechnungsdauer, da hier für jeden Datensatz konstant viele Partitionen geladen werden. Tabelle *Nation* und *Lineitem* weisen ab 300 GB ein leicht kleiner Anzahl an einzulesenden Partitionen auf.

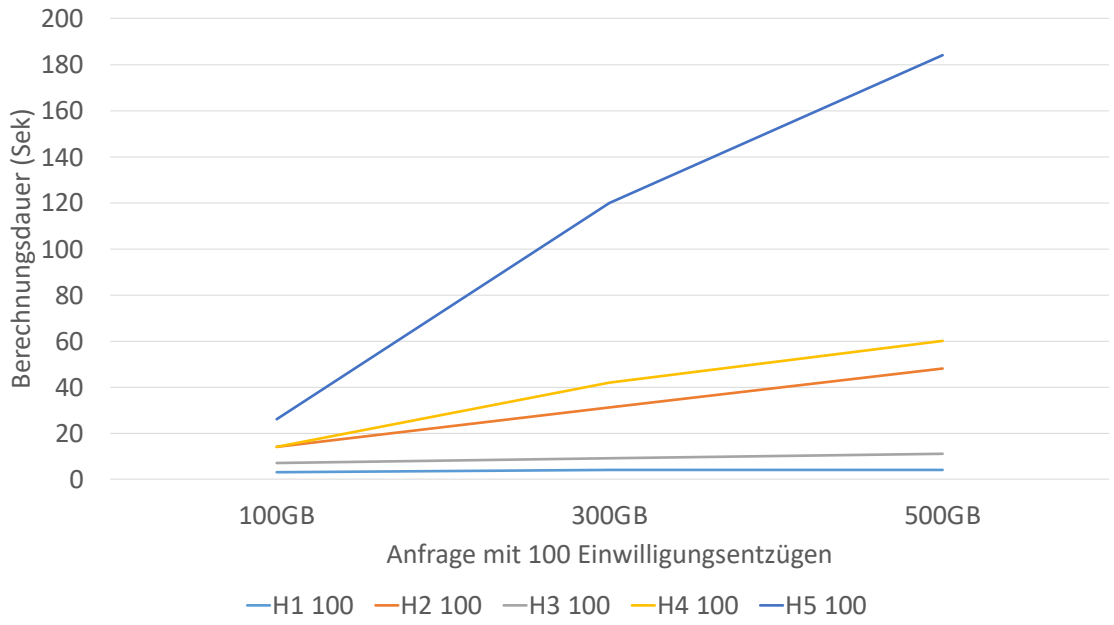



**Abbildung 7.16:** TPC-H: Berechnungsdauer bei zunehmender Datengröße

| **Anfrage** | **Tabelle** | **Datensatzgröße** | | |
|---|---|---|---|---|
| | | 100GB | 300GB | 500GB |
| H1 | Customer | 0 | 0 | 0 |
| H2 | Orders | 1544 | 1586 | 1594 |
| | Customer | 100 | 100 | 100 |
| H3 | Customer | 41 | 41 | 41 |
| | Orders | 47 | 50 | 54 |
| | Lineitem | 127 | 121 | 121 |
| | Nation | 23 | 20 | 20 |
| H4 | Supplier | 785 | 187 | 687 |
| | Part | 5492 | 7013 | 7414 |
| | Partsupp | 5466 | 6921 | 7315 |
| H5 | Supplier | 10000 | 30000 | 50000 |
| | Customer | 10000 | 30000 | 50000 |

**Tabelle 7.3:** TPC-H: Anzahl der Partitionen in Abhängigkeit von der Datengröße pro Eingabetabelle

Diese Verhalten ist im DBLP Datensatz wiederzufinden. D2 liest dabei pro Datensatzgröße ca. 90% der Partitionen für die *Inproceedings*-Tabelle ein, sodass die Berechnungsdauer ansteigt. D1 zeigt einen linearen Anstieg, da pro Datensatzgröße die Anzahl der einzulesenden Partitionen um ca. 700 - 1700 Partitionen wächst. D3, D4 und D5 weisen eine konstante Berechnungsdauer auf, da sich die Anzahl der Partitionen nur geringfügig ändert.

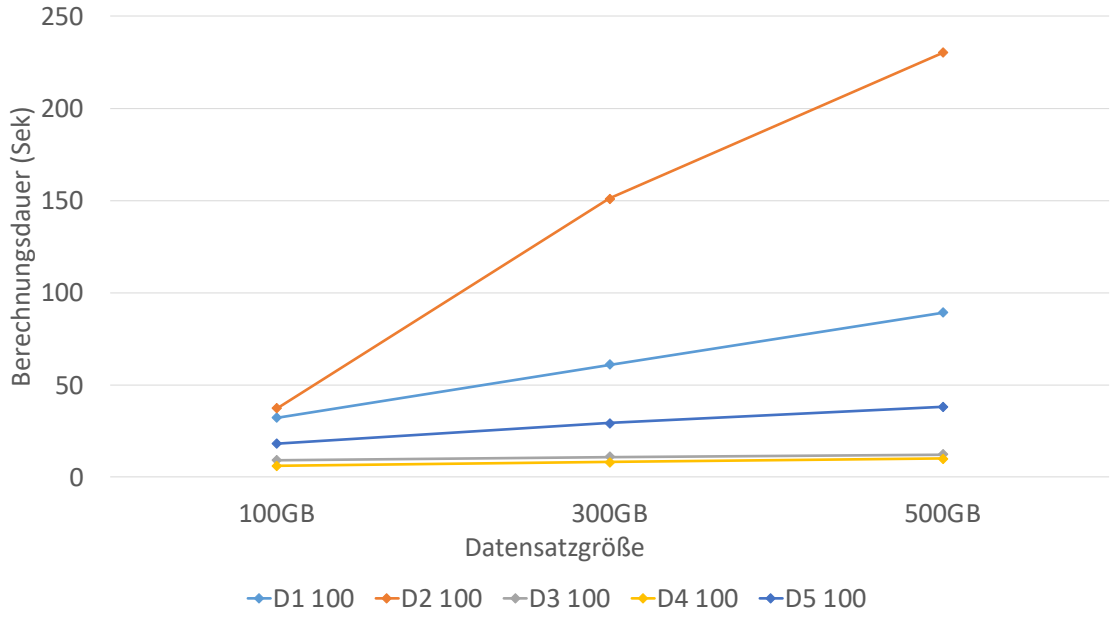



**Abbildung 7.17:** DBLP: Berechnungsdauer bei zunehmender Datengröße

| Anfrage | Tabelle | Datensatzgröße | | |
|---|---|---|---|---|
| | | 100GB | 300GB | 500GB |
| D1 | Article | 5464 | 6194 | 7842 |
| D2 | Inproceedings | 9442 | 28365 | 47195 |
| | Proceedings | 848 | 2526 | 4225 |
| D3 | WWW | 710 | 729 | 734 |
| | Inproceedings | 280 | 284 | 283 |
| D4 | Proceedings | 4 | 4 | 4 |
| | Inproceedings | 111 | 111 | 111 |
| D5 | Article | 1476 | 1566 | 1589 |
| | Inproceedings | 874 | 909 | 911 |

**Tabelle 7.4:** DBLP: Anzahl der Partitionen in Abhängigkeit von der Datengröße pro Eingabetabelle

Dieser Abschnitt hat gezeigt, das die indexgestützte Berechnung bei steigender Datengröße skaliert. Es konnte festgestellt werden, das dabei bei einigen Anfragen wie etwa H3 und H4 sogar weniger Partitionen geladen werden. Dies muss aber keinen zwangsläufigen positiven Einfluss auf die Berechnungsdauer aufweisen, da in einer Partition bei 300GB mehr Tupel vorhanden sein können als bei einer Partition bei 100GB. Im DBLP Datensatz zeigen 3 von 5 Anfragen eine fast konstante Berechnungsdauer, da sich die Anzahl der einzulesenden Partitionen kaum unterscheidet.

### 7.3.4 Indexgestützte Neuberechnung

In diesem Abschnitt wird die Berechnungsdauer der indexgestützten Neuberechnung aus Kapitel 5 im Vergleich zu einer Neuberechnung ohne Index gezeigt. Die Abbildungen 7.18 und 7.19 sind dabei die jeweiligen Diagramme für den TPC-H und DBLP Datensatz. Die x-Achse zeigt dabei die einzelnen Anfragen der Datensätze, die jeweils unterteilt in die drei Datengrößen 100GB, 300GB und 500GB sind. Die y-Achse zeigt die Berechnungsdauer in Sekunden. Die grauen Balken stellen dabei die Berechnungsdauer der Identifikation der betroffenen Eingabetupel aus Abschnitt 5.2 da. Die orangenen Balken stellen die Berechnungsdauer der indexgestützten Neuberechnung d. h. die Berechnung auf Teildaten aus dem Abschnitt 5.2.2 da. Die blauen Rauten stellen die Berechnungsdauer ohne Index d. h. die Berechnung auf allen Eingabedaten da. Aus Übersichtsgründen werden dabei nur 100 Einwilligungsentzüge dargestellt, da die Berechnungsdauer bei steigender Datengröße skaliert.

Für den TPC-H Datensatz konnte festgestellt werden, das die Anfrage H5 von einer Neuberechnung mittels Index nicht profitiert. Dies liegt an hauptsächlich an dem erhöhten Aufwand der Identifikation der Eingabetupel. Da H1 bei Änderungen in den Einwilligungen keine Neuberechnung benötigt, profitiert diese Anfrage durch die Identifikation der betroffenen Ergebnistupel, um jeweils 40%, 66% und 75%. H2 profitiert durch die indexgestützte Neuberechnung um jeweils ca. 16%, 41% und 35%. H4 profitiert erst ab einer Datensatzgröße von 500GB von einer indexgestützte Berechnung um 14%. H3 profitiert am meisten vom Index, da für diese Anfrage ein geringe Aufwand bezüglich der Identifikation der Eingabetupel entsteht. Die Berechnung dieser Anfrage ist komplex, sodass die Berechnung mittels des Index und der reduzierten Datenmenge deutlich schneller ist als die Berechnung auf dem vollständigen Datensatz. Dies führt zu einer beschleunigten Berechnung, um jeweils 80%, 90% und 92%.

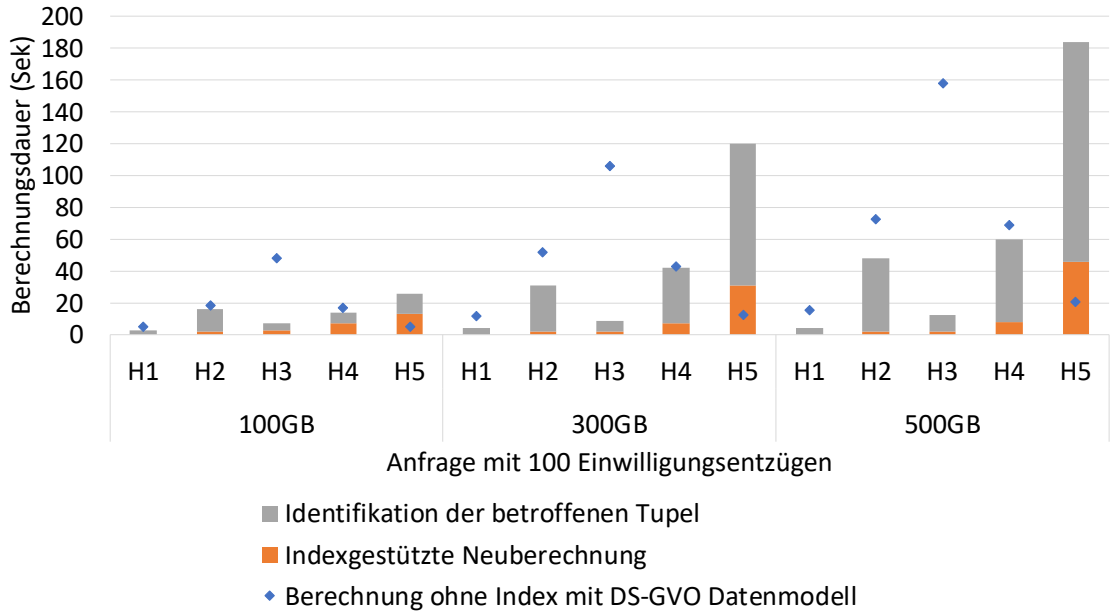



**Abbildung 7.18:** TPC-H: Berechnungsdauer der Anfragen mit und ohne Nutzung des Index

Die Anfragen des DBLP Datensatzes profitieren am meisten von der Verwendung des Indexes. Eine Ausnahme bildet hier die Anfrage D2. Der Mehraufwand ist hierbei die Identifikation der Eingabetupel und der damit verbunden erhöhten Berechnungsdauer. Die Änderungen führen dazu, dass fast sämtliche Partitionen geladen werden müssen. Dies liegt zu einem an der Anfrage, da hierbei eine Vielzahl an Daten auf einen Wert reduziert werden. Das führt dazu, dass Änderungen entsprechend viele Eingabetupel und somit Partitionen treffen. Die Anfragen D3, D4 und D5 profitiere hingegen von der Verwendung des Index. D1 profitiert von Index ab der Datengröße 300GB und 500GB, um jeweils ca. 9% und 20%. Für D3 ist die Berechnungsdauer um ca. 70%, 85% und 90% schneller. D4 ist ca. 80%, 91% und 93% mit der indexgestützte Neuberechnung schneller. D5 ist ca. 45%, 63% und 71% mit der indexgestützten Neuberechnung schneller.

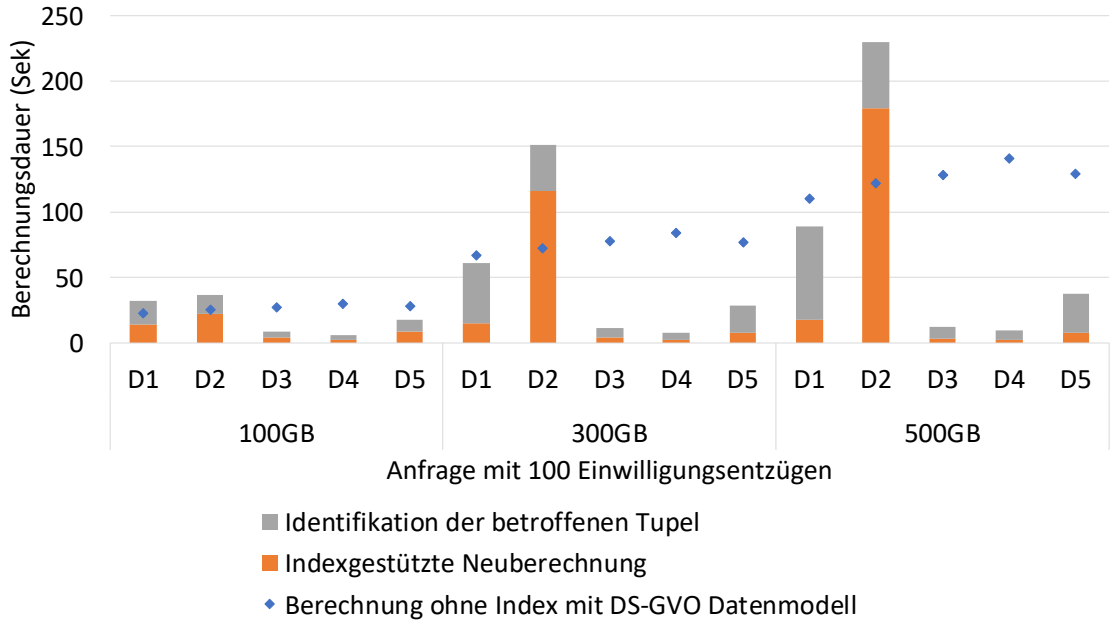



**Abbildung 7.19:** DBLP: Berechnungsdauer der Anfragen mit und ohne Nutzung des Index

### 7.3.5 Optimierung von Ausführungsplänen

Dieser Abschnitt befasst sich mit den Messergebnis im Bezug auf die Ausführung des Index mit Optimierung und ohne Optimierung. Abbildung 7.20 zeigt die Unterschiede für TPC-H und DBLP. Die y-Achse stellt dabei die Berechnungsdauer in Sekunden dar, während die x-Achse die Anfragen pro Szenario darstellt. Aus Gründen der Übersicht werden nur 100 Einwilligungsentzüge gezeigt, da sich das Verhalten bei mehr Entzügen nicht ändert. Die gefärbten Balken stellen dabei jeweils die Berechnungsdauer für einen Berechnung mit Filter und ohne Filter da. Die Optimierung der Ausführungspläne basiert dabei auf der Annahme das Filter-Operationen in der indexgestützten Berechnung entfernt werden können, da die Daten bei der Indexerzeugung bereits selektiert sind. Dadurch verweist der Index bereits auf Daten, die vom Filter-Operator ausgewählt wurden. Nicht alle Anfragen verfügen dabei über einen Filter. Für TPC-H wurden die Anfrage H3 optimiert und für DBLP die Anfragen D2 bis D5. H1 verfügt zwar ebenso über einen Filter-Operator, jedoch wird keine

Neuberechnung durchgeführt, da nur die Ergebnistupel der Ergebnisrelation identifiziert werden. Aus diesem Grund kann H1 nicht optimiert werden, da die Anfrage als Teil einer Neuberechnung nicht ausgeführt wird.

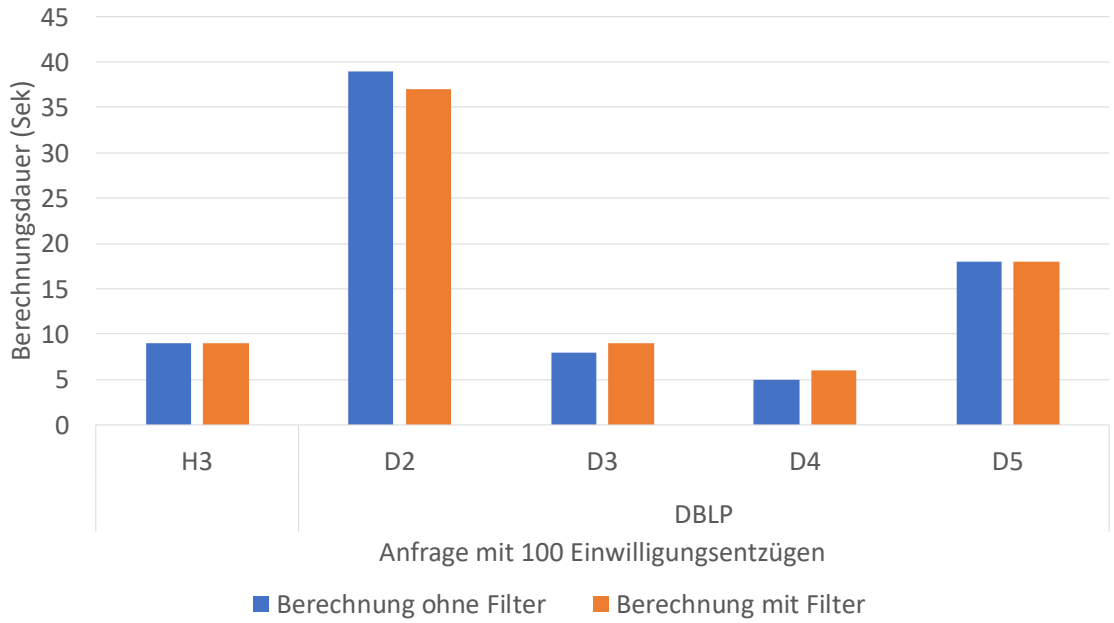



**Abbildung 7.20:** Berechnungsdauer mit und ohne Filter-Operationen

Für beide Datensätze ist keine schnellere Berechnungszeit festzustellen. Leichte Schwankungen in den Zeiten können dabei ignoriert werden, da dies begründet durch die verteilte Berechnung ist. Apache Spark optimiert das Filtern der Daten, in dem Filter-Operation für die jeweilige Relation zusammengefasst und zusammen ausgeführt werden. Dies geschieht durch die *PushDownPredicate*-Logik bevor die Daten in den Hauptspeicher geladen und an die Knoten verteilt werden. Dadurch findet bereits eine Optimierung der Filterung statt.

## 7.4 Zusammenfassung der Ergebnisse

In diesem Abschnitt werden die Erkenntnisse der Evaluation zusammengefasst und Rückschlüsse aus den Ergebnissen gezogen. Aus der Evaluation der prototypischen Implementierung lassen sich folgende Beobachtungen anstellen. Die Berechnungsdauer der Indexerzeugung steigt linear bei zunehmender Datengröße, die Zeit hängt dabei maßgeblich von der verwendeten Provenance-Lösung ab. Je schneller die Provenance-Lösung, die Ergebnistupel bis zu ihren Eingabetupel zurückverfolgt umso schneller kann der Index aufgebaut werden. Die Zeiten von bis zu 45 Minuten sind aber dennoch vertretbar, da der Index nur einmal aufgebaut werden muss. Die anschließende indexgestützte Neuberechnung hat eine effizientere Neuberechnung von bis zu 93% im Hinblick auf die Berechnungsdauer ermöglicht, jedoch nicht für alle Anfragen. Anfrage H5 für TPC-H und D2 für DBLP profitieren nicht vom Index. Dies liegt daran, das bei diesen Anfragen fast sämtliche Partitionen geladen werden müssen. Die Identifikation der betroffenen Tupel benötigt dadurch eine

längere Berechnungsdauer. Daraus lässt sich schließen, dass indexgestützte Neuberechnung genau dann effizienter als Neuberechnung ohne Index sind, wenn die Anzahl an eingelesene Partitionen gering ist. Dies ist deutlich in der Anfrage H3 des TPC-H Datensatzes zu sehen. Die Anfrage benötigt für die initiale Berechnung am längsten von allen Anfragen in dem TPC-H Datensatz und profitiert von der indexgestützte Neuberechnung am meisten. Das liegt an der geringen Anzahl an Partitionen, die geladen werden müssen. Im Vergleich dazu braucht, die Anfrage H5 am längsten. Diese Anfrage gruppiert und aggregiert die Werte auf 25 Gruppen. Bei einem Einwilligungsentzug hat das zur Folge, das der Entzug eine Vielzahl an Eingabetupeln betrifft.

Dieses Verhalten ist ebenso im DBLP Datensatz zu finden. Mittels des Index können die Anfragen D1 und D3-D5 effizienter neu berechnet werden, wobei D1 erst bei steigender Datengröße eine effiziente Neuberechnung zeigt. D2 zeigt das gleiche Verhalten wie H5, da eine Großteil der *Inproceedings*-Tabelle geladen werden muss. Eine effiziente indexgestützte Neuberechnung ist demnach nur dann möglich, wenn die Anzahl an einzulesenden Partitionen gering bleibt d. h. die Gruppierung und Aggregation in den Anfragen verteilt die Daten auf eine Vielzahl an Gruppen, sodass bei einer Änderung die Anzahl der betroffenen Eingabetupel in der Gruppe gering bleibt. Der Index ist somit abhängig von der jeweiligen Anfrage.

# 8 Zusammenfassung und Ausblick

Die Datenschutz-Grundverordnung ermöglicht EU-Bürgern die volle Kontrolle über ihre personenbezogenen Daten. Einwilligungen für die Verarbeitung diese Daten in personenbezogenen Analysen können zu jeder Zeit entzogen werden. Damit die Daten konform zur DS-GVO berechnet werden können, wurde in dieser Arbeit eine Datenmodell entwickelt, dass die Abbildung der Einwilligungen auf den Daten ermöglicht. Mittels diesem Modell können Neuberechnung DS-GVO konform durchgeführt werden. Die Neuberechnung auf den vollständigen Datensatz im Fall eines Einwilligungsentzuges ist jedoch ineffizient, insbesondere wenn die Datenmenge sehr groß und die personenbezogenen Analyse komplex ist. Des weiteren müssen Daten für die die Einwilligung entzogen worden ist, nicht zwangsläufig neu berechnet werden, wenn sie auf Grund von Filter-Operationen ohnehin nicht in das Ergebnis eingehen. In dieser Arbeit wurde ein Verfahren entwickelt, welches die Identifikation der tatsächlich relevanten Eingabedaten und Ergebnisdaten ermöglicht. Durch eine Indexstruktur, welche Eingabedaten zu ihren Ergebnisdaten zuordnet, wird dies bewerkstelligt. Durch die Indizierung der Eingabe- und Ergebnisdaten ist eine effiziente Identifikation der Ergebnisdaten und der betroffenen Eingabedaten bei Änderungen möglich. Dadurch kann eine Neuberechnung effizienter durchgeführt werden, indem die Berechnung nur auf den geänderten Daten stattfindet, anstelle auf dem vollständigen Datensatz. Das vorherige und nicht mehr DS-GVO konforme Ergebnis wird nach der Neuberechnung durch das neue und DS-GVO konforme Ergebnis ersetzt.

Die Indexstruktur basiert dabei auf Provenance-Daten. Dazu wurde auf Basis einer umfangreichen Literaturrecherche und einem Vergleich von bestehenden Provenance-Lösungen, die Provenance-Lösung *Pebble* als geeignet für die Umsetzung dieses Verfahrens identifiziert. Pebble ermöglicht das Zurückverfolgen von Ausgabedaten zu ihren Eingabedaten und ist in Apache Spark implementiert. Die Implementierung des DS-GVO konformen Datenmodell und der Indexstruktur wurde ebenfalls in Apache Spark vorgenommen, dabei wurden verschiedene Implementierung der Indexstruktur evaluiert. Die bestmögliche Implementierung basiert dabei auf der Verwendung der *DataFrame*-Abstraktion von Apache Spark, da durch die volle Leistung von Apache Spark genutzt werden konnte. Die Konzepte lassen sich aber auf anderen VSDB-Systeme und andere Provenance-Lösungen übertragen.

Zuletzt wurde die Implementierung der Konzepte innerhalb Apache Spark mittels dem TPC-H Benchmark und DBLP, sowie jeweils fünf Anfragen evaluiert. Das Ergebnis der Evaluation hat einen erwartbaren Mehraufwand im Bezug auf das DS-GVO-konformen Datenmodell gezeigt. Dies liegt an der zusätzlichen Filter-Operation, mit der Daten herausgefiltert werden, für die keine Einwilligung mehr besteht. Das Datenmodell hat aber den Vorteil, dass das Modell auf beliebig viele Einwilligungen erweitert werden kann. Der Index kann nach der einmaligen Indexerzeugung genutzt werden und ermöglicht eine bis zu 93% schnellere Neuberechnung auf Teildaten. Anfragen die Ergebnisse mittels Gruppierungs- und Aggregationsoperationen auf viele verschiedene Gruppen verteilen, weisen eine tendenziell effizientere indexgestützte Neuberechnung auf. Über die Berechnungsdauer mittels der Indexstruktur auf Teildaten kann jedoch keine generelle Aussage getroffen

werden, da dies abhängig von der jeweiligen Anfrage ist. Die Entscheidung, wann die Indexstruktur genutzt werden soll kann vorab anhand der Anfrage und dem zu erwartenden Ergebnis getroffen werden. Unternehmen sollten dazu, die Berechnungsdauer durch Experimente messen und den Index entsprechend nur dann einsetzen, wenn die Anfrage eine lange Berechnungsdauer aufweist und wenn die Änderungen der Einwilligung selten vorkommt.

## Ausblick

Die durchgeführte Evaluation zeigt, dass die Verwendung des entwickelten Verfahrens nicht zwangsläufig zu einer schnelleren Berechnung führt. Mit steigender Anzahl an Änderungen ist ein zunehmender Berechnungsaufwand im Bezug auf die Identifizierung der betroffenen Daten nötig. Dies trifft insbesondere zu, wenn die Daten auf viele verschiedenen Partitionen verteilt sind. Es wäre wünschenswert anhand von Metriken zu entscheiden, ob eine Neuberechnung auf Teildaten oder auf dem vollständigen Datensatz empfehlenswerter ist. Vielversprechende Metriken hierzu sind etwa:

- **Anzahl der geänderten Eingabetupel**
  Je größer die Anzahl der geänderten Eingabetupel ist, umso weniger ist die Verwendung der Indexstruktur zu empfehlen, da der Identifikationsaufwand pro geänderten Eingabetupel steigt.
- **Anzahl der betroffenen Ergebnistupel**
  Je größer die Anzahl der betroffenen Ergebnistupel ist, umso mehr Eingabetupel müssen identifiziert werden, sodass die Identifikation einen zunehmenden Berechnungsaufwand aufweisen kann.
- **Dauer der Berechnung**
  Anfragen die eine lange Berechnungsdauer aufweisen sind potentielle Kandidaten für die Neuberechnung mittels Index, da dadurch die Zeiten stark reduziert werden können. Falls Anfragen nur wenige Sekunden für eine Berechnung auf allen Daten benötigen, ist die Verwendung eines Index in den meisten Fällen nicht notwendig, da die Identifikation der Daten zusätzlichen Aufwand verursacht, länger benötigt als die vollständige Neuberechnung.
- **Verfügbare Hardwareressourcen**
  Die verfügbaren Hardwareressourcen können ein ausschlaggebender Punkt für die Neuberechnung von Analysen sein. Die Neuberechnung mittels Index kann potentiell auf leistungsschwacher Hardware ausgeführt werden. Dadurch können Hardwareressourcen eingespart werden.
- **Verteilung der Eingabedaten auf Partitionen**
  Die Verteilung der Eingabedaten auf Partitionen kann ausschlaggebend dafür sein, ob die Identifikation mittels Index und die Berechnung effizient durchgeführt werden kann. Eine optimale Verteilung ist hierbei eine Partition pro Tupel. Da dies in der Praxis meistens nicht möglich ist, sollten die Daten auf eine fest vordefinierte Anzahl an Partitionen verteilt werden.

Ein weiteres Einsatzgebiet, für das in dieser Arbeit entwickelte Verfahren, kann die explorative Datenanalyse sein. Häufig werden hierzu interaktive Data-Mashup-Werkzeuge eingesetzt [DM14]. Domänenexperten sind in der Lage mit Hilfe dieser Werkzeuge Datenquellen und Operationen

auf interaktive Weise beliebig zu verknüpfen und einen kompletten Analyseablauf zu spezifizieren. Durch die Verwendung des DS-GVO konformen Datenmodells kann sichergestellt werden, dass bei der explorativen Analyse nur Daten verwendet werden, für die eine Einwilligung seitens des Verbrauchers vorliegt bzw. andernfalls die Anzeige von Ergebnissen verhindert werden. Die in dieser Arbeit entwickelte Indexstruktur kann verwendet werden, um die Assoziation zwischen Eingabe und Ausgabe darzustellen. Dadurch kann der gesamte Analyseablauf oder Teile des Analyseablauf effizient neu berechnet werden, falls eine Änderung in den Daten eine Auswirkung auf den Analyseablauf hat. Die Ergebnisse können somit schneller an die Domänenexperten zurückgeliefert werden, dies sorgt für eine erhöhte Interaktivität, da der Domänenexperte auf die Berechnung von Neuberechnung nicht lange warten muss. Dadurch kann mehr Domänenwissen in kürzerer Zeit eingebracht werden, dies kann zu belastbaren Analysen führen [BHM18].

# Literaturverzeichnis


[ADD+11] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, V. Tannen. „Putting lipstick on pig: Enabling database-style workflow provenance“. In: *Proceedings of the VLDB Endowment* 5.4 (2011), S. 346–357 (zitiert auf S. 45, 47).

[Apa17] Apache Software Foundation. *partitioning reporting*. Apache Software Foundation. 2017. URL: https://issues.apache.org/jira/browse/SPARK-22389 (zitiert auf S. 78).

[ApaoDa] Apache Software Foundation. *Cluster Mode Overview. Components*. Apache Software Foundation. o.D. URL: https://spark.apache.org/docs/latest/cluster-overview.html (zitiert auf S. 27).

[ApaoDb] Apache Software Foundation. *Parquet Files*. Apache Software Foundation. o.D. URL: https://spark.apache.org/docs/2.4.0/api/java/org/apache/spark/sql/DataFrameWriter.html#partitionBy-scala.collection.Seq- (zitiert auf S. 78).

[ApaoDc] Apache Software Foundation. *RDD Programming Guide*. Apache Software Foundation. o.D. URL: https://spark.apache.org/docs/latest/rdd-programming-guide.html (zitiert auf S. 76, 92).

[ApaoDd] Apache Software Foundation. *Spark SQL, DataFrames and Datasets Guide*. Apache Software Foundation. o.D. URL: https://spark.apache.org/docs/2.2.0/sql-programming-guide.html#spark-sql-dataframes-and-datasets-guide (zitiert auf S. 27).

[BHM18] M. Behringer, P. Hirmer, B. Mitschang. „A Human-Centered Approach for Interactive Data Processing and Analytics“. In: Juni 2018, S. 498–514. ISBN: 978-3-319-93374-0. DOI: 10.1007/978-3-319-93375-7_23 (zitiert auf S. 107).

[BKT02] P. Buneman, S. Khanna, W.-C. Tan. „On Propagation of Deletions and Annotations through Views“. In: *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '02. Madison, Wisconsin: Association for Computing Machinery, 2002, S. 150–158. ISBN: 1581135076. DOI: 10.1145/543613.543633. URL: https://doi.org/10.1145/543613.543633 (zitiert auf S. 50).

[BMJoD] BMJV. *Datenschutz-Grundverordnung*. Bundesministerium für Justiz und Verbraucherschutz. o.D. URL: https://www.bmjv.de/DE/Themen/FokusThemen/DSGVO/DSVGO_node.html (zitiert auf S. 24).

[BMWoD] BMWi. *Europäische Datenschutz-Grundverordnung*. Bundesministerium für Wirtschaft und Energie. o.D. URL: https://www.bmwi.de/Redaktion/DE/Artikel/Digitale-Welt/europaeische-datenschutzgrundverordnung.html (zitiert auf S. 16, 23, 24).

[BS81] F. Bancilhon, N. Spyratos. „Update Semantics of Relational Views“. In: *ACM Trans. Database Syst.* 6.4 (Dez. 1981), S. 557–575. ISSN: 0362-5915. DOI: 10.1145/319628.319634. URL: https://doi.org/10.1145/319628.319634 (zitiert auf S. 50).

[BWD+18] K. Barley, A. Wambach, R. Dewenter, C. Hildebrandt, H. Hosseini, H. Schmidt, P. Buxmann. „Big Data als Geschäftsmodell: Wie mit der Macht der Internetfirmen umgehen?“ ger. In: *ifo Schnelldienst* 71.10 (2018), S. 3–21. ISSN: 0018-974X. URL: http://hdl.handle.net/10419/181103 (zitiert auf S. 15).

[CCT09] J. Cheney, L. Chiticariu, W.-c. Tan. „Provenance in Databases: Why, How, and Where“. In: *Foundations and Trends in Databases* 1 (Jan. 2009), S. 379–474. DOI: 10.1561/1900000006 (zitiert auf S. 30).

[CL11] H. Chen, H. Liao. „A Survey to View Update Problem“. In: *International Journal of Computer Theory and Engineering* (Jan. 2011), S. 23–31. DOI: 10.7763/IJCTE.2011.V3.278 (zitiert auf S. 50).

[CMY14] M. Chen, S. L. Mao, Yunhao. „Big Data: A Survey“. In: *Mobile Networks and Applications* 19.2 (2014), S. 171–209. ISSN: 1572-8153. DOI: 10.1007/s11036-013-0489-0. URL: https://doi.org/10.1007/s11036-013-0489-0 (zitiert auf S. 25).

[Cou18] T. P. P. Council. *TPC Benchmark H. (Decision Support) Standard Specification*. Hrsg. von T. P. P. Council. Transaction Processing Performance Council (TPC. 6. Dez. 2018. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf (zitiert auf S. 17).

[DG04] J. Dean, S. Ghemawat. „MapReduce: Simplified Data Processing on Large Clusters“. In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. San Francisco, CA, 2004, S. 137–150 (zitiert auf S. 26, 40).

[DH19] R. Diestelkämper, M. Herschel. „Capturing and Querying Structural Provenance in Spark with Pebble“. In: *Proceedings of the 2019 International Conference on Management of Data*. ACM. 2019, S. 1893–1896 (zitiert auf S. 30, 44, 47, 69).

[DH20] R. Diestelkämper, M. Herschel. „Tracing nested data with structural provenance for big data analytics“. In: *Proceedings of the 22nd International Conference on Extending Database Technology*. EDBT. 2020, S. 1893–1896 (zitiert auf S. 30, 44, 47, 59, 69, 76, 86, 87).

[DM14] F. Daniel, M. Matera. „Mashups“. In: *Mashups*. Springer, 2014, S. 137–181 (zitiert auf S. 106).

[EN15] R. Elmasri, S. B. Navathe. *Fundamentals of Database Systems*. 7th. Pearson, 2015. ISBN: 0133970779 (zitiert auf S. 25, 32–34, 52).

[FG12] E. Franconi, P. Guagliardo. *The View Update Problem Revisited*. 2012. arXiv: 1211.3016 [cs.DB] (zitiert auf S. 50).

[GIY+16] M. A. Gulzar, M. Interlandi, S. Yoo, S. D. Tetali, T. Condie, T. Millstein, M. Kim. „Bigdebug: Debugging primitives for interactive big data processing in spark“. In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE. 2016, S. 784–795 (zitiert auf S. 43, 46, 47).

[GKT07] T. J. Green, G. Karvounarakis, V. Tannen. „Provenance Semirings“. In: *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '07. Beijing, China: Association for Computing Machinery, 2007, S. 31–40. ISBN: 9781595936851. DOI: 10.1145/1265530.1265535. URL: https://doi.org/10.1145/1265530.1265535 (zitiert auf S. 30).

[GM13] P. Groth, L. Moreau. *PROV-Overview. An Overview of the PROV Family of Documents*. World Wide Web Consortium. 30. Apr. 2013. URL: https://www.w3.org/TR/prov-overview/ (zitiert auf S. 38).

[GMB+13] Y. Gil, S. Miles, K. Belhajjame, H. Deus, D. Garijo, G. Klyne, P. Missier, S. Soiland-Reyes, S. Zednik. *PROV Model Primer*. World Wide Web Consortium. 30. Apr. 2013. URL: https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/ (zitiert auf S. 37, 38).

[GSM+18] T. Guedes, V. Silva, M. Mattoso, M. V. Bedo, D. de Oliveira. „A practical roadmap for provenance capture and data analysis in spark-based scientific workflows“. In: *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE. 2018, S. 31–41 (zitiert auf S. 46, 47).

[HDB17] M. Herschel, R. Diestelkämper, H. Ben Lahmar. „A survey on provenance: What for? What form? What from?“ In: *The VLDB Journal—The International Journal on Very Large Data Bases* 26.6 (2017), S. 881–906. DOI: 10.1007/s00778-017-0486-1 (zitiert auf S. 29–31).

[Hua19] S. Hua. *Alibaba erzielt neuen Verkaufsrekord beim Singles' Day. Wie mit deinen Daten gehandelt wird*. 11. Nov. 2019. URL: https://www.handelsblatt.com/unternehmen/handel-konsumgueter/online-gigant-alibaba-erzielt-neuen-verkaufsrekord-beim-singles-day/25213624.html?ticket=ST-3657705-c1iaa5deIScj1CeiX5SD-ap6 (zitiert auf S. 21).

[IES+18] M. Interlandi, A. Ekmekji, K. Shah, M. A. Gulzar, S. D. Tetali, M. Kim, T. Millstein, T. Condie. „Adding data provenance support to Apache Spark“. In: *The VLDB Journal—The International Journal on Very Large Data Bases* 27.5 (2018), S. 595–615 (zitiert auf S. 42).

[IPW11] R. Ikeda, H. Park, J. Widom. „Provenance for Generalized Map and Reduce Workflows.“ In: Jan. 2011, S. 273–283 (zitiert auf S. 40, 41, 43, 47).

[IST+15] M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. Millstein, T. Condie. „Titian: Data provenance support in spark“. In: *Proceedings of the VLDB Endowment* 9.3 (2015), S. 216–227 (zitiert auf S. 42–44, 46, 47).

[Kne19] H. Knebl. *Algorithmen und Datenstrukturen - Grundlagen und probabilistische Methoden für den Entwurf und die Analyse*. Berlin Heidelberg New York: Springer-Verlag, 2019. ISBN: 978-3-658-26512-0 (zitiert auf S. 76).

[KSV06] Y. Kotidis, D. Srivastava, Y. Velegrakis. „Updates Through Views: A New Hope“. In: *22nd International Conference on Data Engineering (ICDE'06)*. Apr. 2006, S. 2–2. DOI: 10.1109/ICDE.2006.167 (zitiert auf S. 50).

[LasoD] J. Laskowski. *Repartition Logical Operators — Repartition and RepartitionByExpression*. o.D. URL: https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql-LogicalPlan-Repartition-RepartitionByExpression.html (zitiert auf S. 78).

[LDY13] D. Logothetis, S. De, K. Yocum. „Scalable lineage capture for debugging disc analytics“. In: *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM. 2013, S. 17 (zitiert auf S. 42, 43, 47).

[Ley09] M. Ley. „DBLP: Some Lessons Learned“. In: *Proc. VLDB Endow.* 2.2 (Aug. 2009), S. 1493–1500. ISSN: 2150-8097. DOI: 10.14778/1687553.1687577. URL: https://doi.org/10.14778/1687553.1687577 (zitiert auf S. 86).

[MC19] P. Missier, J. Cala. „Efficient Re-Computation of Big Data Analytics Processes in the Presence of Changes: Computational Framework, Reference Architecture, and Applications“. In: *2019 IEEE International Congress on Big Data (BigDataCongress)*. Juli 2019, S. 24–34. DOI: 10.1109/BigDataCongress.2019.00017 (zitiert auf S. 49).

[NIS15] NIST. *NIST Special Publication 1500-1. NIST Big Data Interoperability Framework: Volume 1, Definitions*. Final Version 1. 1. National Institute of Standards und Technology, 1. Sep. 2015. 32 S. DOI: 10.6028/NIST.SP.1500-1 (zitiert auf S. 25).

[OAC+04] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, M. Zenger. *An overview of the Scala programming language*. Techn. Ber. 2004 (zitiert auf S. 70).

[PL17] H. J. Pandit, D. Lewis. „Modelling Provenance for GDPR Compliance using Linked Open Data Vocabularies.“ In: *PrivOn@ ISWC*. 2017 (zitiert auf S. 39, 40).

[PW18] F. Psallidas, E. Wu. „Smoke: Fine-grained lineage at interactive speed“. In: *Proceedings of the VLDB Endowment* 11.6 (2018), S. 719–732 (zitiert auf S. 48, 49).

[SDC+16] S. Salloum, R. Dautov, X. Chen, P. X. Peng, J. Z. Huang. „Big data analytics on Apache Spark“. In: *International Journal of Data Science and Analytics* 1.3 (Nov. 2016), S. 145–164. ISSN: 2364-4168. URL: https://doi.org/10.1007/s41060-016-0027-9 (zitiert auf S. 26, 27, 69).

[UC92] S. D. Urban, K. Chalmers. „An investigation of the view update problem for object-oriented views“. In: *Eleventh Annual International Phoenix Conference on Computers and Communication [1992 Conference Proceedings]*. Apr. 1992, S. 156–163. DOI: 10.1109/PCCC.1992.200553 (zitiert auf S. 50).

[Uni16] E. Union. *Verordnungen*. Europäischen Union. 27. Apr. 2016. URL: https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32016R0679&from=DE (zitiert auf S. 15, 16, 23, 24).

[Uns10] F. Unseld. *Die Kommerzialisierung personenbezogener Daten*. Bd. 769. Herbert Utz Verlag, 2010 (zitiert auf S. 15).

[WB13] J. S. Ward, A. Barker. „Undefined By Data: A Survey of Big Data Definitions“. In: *CoRR* abs/1309.5821 (2013). arXiv: 1309.5821. URL: http://arxiv.org/abs/1309.5821 (zitiert auf S. 25).

[Wie91] G. Wiederhold. „Views, Objects, and Databases“. In: *On Object-Oriented Database Systems*. Hrsg. von K. R. Dittrich, U. Dayal, A. P. Buchmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, S. 29–43. ISBN: 978-3-642-84374-7. DOI: 10.1007/978-3-642-84374-7_3. URL: https://doi.org/10.1007/978-3-642-84374-7_3 (zitiert auf S. 50).

[ZAI19] N. Zheng, A. Alawini, Z. G. Ives. „Fine-Grained Provenance for Matching & ETL“. In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE. 2019, S. 184–195 (zitiert auf S. 45, 47).

[ZXW+16] M. Zaharia, R. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica. „Apache spark: A unified engine for big data processing“. In: *Communications of the ACM* 59 (Nov. 2016), S. 56–65. DOI: 10.1145/2934664 (zitiert auf S. 26, 27, 29, 69).

Alle URLs wurden zuletzt am 05. 07. 2020 geprüft.

**Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift