Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik


Masterarbeit

# Evaluierung der Effizienz einer semi-virtuellen Data Warehouse Architektur im Vergleich zu einem multidimensionalen Datenwürfel im Hinblick auf Volumen

Autor:

Rudi Bikschentajew

15. Oktober 2018

Betreuer:

Prof. Dr. rer. nat. habil. Gunter Saake
Dr. Veit Köppen
David Broneske

Institut für Technische und Betriebliche Informationssysteme

Externe Betreuer: Collin Rogowski (inovex GmbH)

**Bikschentajew, Rudi:**
Evaluierung der Effizienz einer semi-virtuellen Data Warehouse
Architektur im Vergleich zu einem multidimensionalen Datenwürfel im Hinblick auf Volumen

Masterarbeit, Otto-von-Guericke-Universität Magdeburg, 2018.

# Abstract

Virtualisierung ist ein Konzept, bei dem Ressourcen von deren technischen Details abgekoppelt werden. Im Falle eines Data Warehouses werden dessen persistierte Schichten durch virtuelle Tabellen ersetzt, wodurch eine erhöhte Flexibilität, schnellere Entwicklungszeiten und eine einfachere Wartbarkeit des Systems erreicht werden kann. Anstatt die Ergebnisse von ETL-Prozessen zu persistieren, werden diese Prozesse in virtuellen Tabellen gespeichert und nur bei Abruf der Tabellen ausgeführt. Die oben erwähnten Vorteile ergeben sich durch eine Reduzierung von Redundanzen bei den gehaltenen Daten.

Mit der Virtualisierung kommen aber auch Nachteile, der größte davon ist eine reduzierte Performanz bei der Verarbeitung von analytischen Anfragen. Der Grund hierfür ist die Notwendigkeit, die ETL-Prozesse bei Abruf spontan durchführen zu müssen. Es muss somit anhand der Anforderungen (technische Anforderungen durch die Anzahl der Quellen, Datenvolumen, etc.) entschieden werden, ob die Vorteile einer Virtualisierung die Nachteile überwiegen. Dies ist jedoch nicht einfach, weil die Folgen der Virtualisierung auf die Performanz nur schwer einzuschätzen sind. Diese Arbeit macht die Einflüsse auf die Leistung anhand eines Use-Cases greifbarer und konkreter. Es wird untersucht, welche Auswirkungen die Virtualisierung von ETL-Prozessen eines Data Warehouses auf dessen analytische Leistungen und Skalierbarkeit haben kann und ob Parameter wie die Anzahl der Quellsysteme eine virtuelle Lösung anders beeinflussen als eine traditionelle Lösung.

Um diese Ziele zu erreichen, wird ein traditionelles Data Warehouse mit einem semivirtuellen Data Warehouse verglichen. Beide werden in derselben Umgebung und mit ähnlichen Technologien umgesetzt, um die gemessenen Unterschiede soweit es geht auf die Architekturunterschiede zu reduzieren. Anhand eines Use-Cases werden Anforderungen definiert. Um festzustellen, wie gut diese erfüllt werden, werden die Verarbeitungszeiten, die Skalierbarkeit und die Einflüsse von bestimmten Faktoren gemessen und evaluiert. Für den gegebenen Use-Case stellte sich das virtuelle Data Warehouse als valide Alternative heraus. Es war bei allen untersuchten Datenmengen in der Lage genauso gut zu skalieren, wie der traditionelle Ansatz. Der vorhandene Nachteil war eine Erhöhung der Verarbeitungszeiten um das drei- bis fünffache. Bei den absoluten Laufzeiten stellte sich dieser Faktor jedoch als weniger Relevant heraus. Es konnte gezeigt werden, dass ein virtuelles Data Warehouse eine valide Alternative mit Vorteilen bieten kann, wenn die gestellten Anforderungen die Grenzen des Ansatzes nicht übersteigen. Jedoch konnten diese Grenzen bei den Tests nicht erreicht werden, hierfür wäre ein umfangreicheres Setup notwendig.

# Inhaltsverzeichnis

**Abbildungsverzeichnis** **xii**

**Tabellenverzeichnis** **xiii**

**1 Einführung** **1**
1.1 Hintergrund . . . 1
1.2 Problem und Forschungsfragen . . . 2
1.3 Abgrenzung des Themenbereiches . . . 2
1.4 Methodische Vorgehensweise . . . 3
1.5 Gliederung der Arbeit . . . 4

**2 Grundlagen** **5**
2.1 Operative und Analytische Systeme . . . 6
2.1.1 OLTP vs. OLAP . . . 6
2.1.2 Möglichkeiten der Datenintegration . . . 7
2.2 Data Warehouse . . . 9
2.2.1 Einführung . . . 9
2.2.2 Architektur . . . 12
2.2.2.1 Architektur nach Inmon . . . 12
2.2.2.2 Architektur nach Kimball . . . 13
2.2.2.3 Unterschiede Inmon / Kimball und Entscheidung . . . 17
2.2.3 Komponenten der Architektur . . . 18
2.2.4 Neuere Architekturen . . . 20
2.3 Virtuelles Data Warehouse . . . 22
2.3.1 Einführung Virtualisierung . . . 22
2.3.2 Architektur (semi-)virtuelles Data Warehouse . . . 25
2.3.2.1 Überblick . . . 25
2.3.2.2 Quellen und Wrapper . . . 26
2.3.2.3 Virtuelle Tabellen und Abbildungen . . . 27
2.3.2.4 Semi-virtuelles Data Warehouse . . . 28
2.3.3 Optimierung . . . 29
2.3.4 Abgrenzung zu traditionellem Data Warehouse . . . 30
2.4 Big Data . . . 32
2.4.1 Definition und Überschneidung mit Data Warehouses . . . 32
2.4.2 Hadoop . . . 33
2.4.3 Verwendete Hadoop-Projekte . . . 35

**3 Implementierung** **39**

3.1 Quellen . . . 40
3.1.1 Quelldatengenerierung . . . 40
3.1.2 Quellsysteme . . . 42
3.2 Vergleichbarkeit der Implementierungen . . . 45
3.3 Traditionelles Data Warehouse . . . 47
3.3.1 Quellen und Extraktion . . . 47
3.3.2 Transformation und Laden . . . 48
3.3.3 Optimierung . . . 51
3.3.4 Abfragen . . . 52
3.4 Semi-virtuelles Data Warehouse . . . 53
3.4.1 Quellen und Wrapper . . . 53
3.4.2 Virtuelle Tabellen . . . 54
3.4.3 Optimierungen . . . 56
3.4.4 Abfragen . . . 57

**4 Evaluierung 59**
4.1 Überblick . . . 60
4.1.1 Szenarien, Abfragen und Testdurchführung . . . 60
4.1.2 Testsystem . . . 63
4.1.3 Performanz Metriken . . . 64
4.1.4 Grenzen der Untersuchungen . . . 65
4.2 Traditionelles Data Warehouse . . . 67
4.2.1 Generelle Trends und Ergebnisse . . . 67
4.2.2 Einflüsse durch die Partitionierung . . . 69
4.2.3 Einflüsse durch die Anzahl der Quellen . . . 71
4.2.4 Einflüsse durch die Arten von Anfragen . . . 72
4.3 Virtuelles Data Warehouse . . . 75
4.3.1 Generelle Trends und Ergebnisse . . . 75
4.3.2 Einflüsse durch die Partitionierung . . . 77
4.3.3 Einflüsse durch die Anzahl der Quellen . . . 80
4.3.4 Einflüsse durch die Arten von Anfragen . . . 82
4.4 Traditionell vs. Virtuell . . . 84
4.4.1 Genereller Vergleich . . . 84
4.4.2 Stabilität und Relevanz der Ergebnisse . . . 86
4.4.3 Einfluss der Partitionierung . . . 87
4.4.4 Einflüsse durch die Arten von Anfragen . . . 88
4.5 Diskussion & Schlussfolgerungen . . . 90
4.5.1 Auswirkungen der Parameter . . . 90
4.5.2 Generelle Schlussfolgerungen und Empfehlung . . . 93
4.5.3 Beantwortung der Forschungsfragen . . . 95

**5 Zusammenfassung und zukünftige Arbeiten 97**
5.1 Reflexion der Vorgehensweise . . . 97
5.2 Beantwortung der Forschungsfragen . . . 99
5.3 Kritische Betrachtung . . . 100
5.4 Ausblick . . . 100

**A Anhang 103**

A.1 Ausreißer beim virtuellen Data Warehouse . . . . . 103
A.2 Einflüsse der Anzahl der Quellen . . . . . 107
A.3 Einflüsse der Partitionierung . . . . . 108
A.4 Ablaufpläne virtuelles Data Warehouse . . . . . 109

**Literaturverzeichnis** **111**

# Abbildungsverzeichnis


2.1 Datenbankschema einer OLTP-Datenbank . . . . . 6

2.2 Vereinfachte Darstellung der CIF Architektur nach Inmon . . . . . 12

2.3 Architektur des Data Warehouses nach Kimball . . . . . 13

2.4 Star-Schema und OLAP-Würfel . . . . . 15

2.5 Studie der Aberdeen Gruppe. Teilnehmer berichten von verkürzten Zeitfenstern für Entscheidungen und sich ändernde Nachfragen an Management-Informationen . . . . . 22

2.6 Virtualisierungsschicht zwischen Anwender und Datenquelle . . . . . 24

2.7 Grobe Architektur eines virtuellen Data Warehouses . . . . . 25

2.8 Zusammenhang aus Quelle, Wrappertabelle und virtueller Tabelle . . 28

2.9 Query mit und ohne Caching . . . . . 29

2.10 Die drei Dimensionen entlang derer Big Data definiert werden kann . 32

2.11 Ein Überblick über das Hadoop Ökosystem . . . . . 35

2.12 Ein Überblick über die Architektur von Impala . . . . . 37

3.1 Datenbankschema der Quellsysteme . . . . . 41

3.2 Extraktion der Quellen aus dem Gesamtdatensatz . . . . . 43

3.3 Implementierung traditionelles Data Warehouse . . . . . 47

3.4 Ordnerstruktur HDFS /etl . . . . . 48

3.5 Sternschema der Präsentationsschicht . . . . . 51

3.6 Implementierung semi-virtuelles Data Warehouse . . . . . 53

3.7 Beispiel Wrappertabelle . . . . . 54

3.8 Aufbau der virtuellen Schicht . . . . . 55

4.1 Anfragen auf dem traditionellen Data Warehouse . . . . . 67

4.2 Relative Performanz nach Einführung der Partitionierung . . . . . 69

4.3 Relative Performanz nach Erhöhung der Quellanzahl . . . . . 71

4.4 Einfluss des Anfragevolumens und der Anfragekomplexität auf die Leistungen des partitionierten traditionellen Data Warehouses . . . . 73

4.5 Anfragen auf dem virtuellen Data Warehouse . . . . . 75

4.6 Relative Performanz nach Einführung der Partitionierung . . . . . 77

4.7 Relative Performanz nach Erhöhung der Quellanzahl . . . . . 80

4.8 Einfluss des Anfragevolumens und der Anfragekomplexität auf die Leistungen des partitionierten virtuellen Data Warehouses . . . . . 82

4.9 Alle Anfragen auf den partitionierten Data Warehouses . . . . . 84

4.10 Abweichungen der Anfragen über mehrere Durchläufe . . . . . 86

4.11 Relative Performanz nach Einführung der Partitionierung . . . . . 88

A.1 Einfluss Quellen; ohne Partitionierung . . . . . 107

A.2 Einfluss Quellen; mit Partitionierung nach Jahr / Monat . . . . . 107

A.3 Einfluss Partitionierung; mit drei Quellen . . . . . 108

A.4 Einfluss Partitionierung; mit zehn Quellen . . . . . 108

A.5 Ablaufplan traditionelles Data Warehouse bei hohem Volumen, hoher Komplexität und zehn Quellsystemen . . . . . 109

A.6 Ablaufplan virtuelles Data Warehouse bei hohem Volumen, hoher Komplexität und zehn Quellsystemen . . . . . 109

A.7 Ablaufplan virtuelles Data Warehouse bei hohem Volumen, hoher Komplexität und zehn Quellsystemen; Vergrößert . . . . . 110

# Tabellenverzeichnis

4.1 Traditionell: Median der Verarbeitungszeiten bei verschiedenen Datenmengen . . . . . 68

4.2 Virtuell: Median der Verarbeitungszeiten bei verschiedenen Datenmengen . . . . . 76

4.3 Mittlere Laufzeit in Sekunden bei verschiedenen Datenmengen. Die letzte Zeile gibt den Faktor an, um den der virtuelle Ansatz langsamer ist . . . . . 85

4.4 Skalierungsfaktoren bei verschiedenen Datenübergängen . . . . . 85

# 1. Einführung

Die Notwendigkeit für Unternehmen die internen Daten für eine verbesserte Entscheidungsfindung zu analysieren, wird immer größer. Gleichzeitig wird diese Aufgabe aufgrund der steigenden Datenmengen anspruchsvoller. Die Daten können auf einer Vielzahl von heterogenen Quellsystemen verteilt sein, einen hohen Umfang besitzen und schnell wachsen. Um aus ihnen einen verlässlichen und maximalen Nutzen generieren zu können, müssen diese bereinigt, standardisiert und integriert werden. Ein hierfür verwendetes System muss also gewisse Anforderungen an Performanz und Skalierbarkeit erfüllen. Demgegenüber stehen meist begrenzte Ressourcen. Ein Projekt für die Umsetzung solch eines Systems muss mit einem begrenzten Budget, und innerhalb eines begrenzten Zeitraumes fertiggestellt werden. Die umzusetzende Architektur muss somit alle Anforderungen zukunftssicher erfüllen und möglichst wenige Ressourcen verbrauchen.

## 1.1 Hintergrund

Eine Möglichkeit firmeninterne Daten zu integrieren und für Analysen zu nutzen ist die Verwendung eines Data Warehouses (siehe Abschnitt 2.2.1). Hier gibt es eine Menge Möglichkeiten, wie man solch ein Data Warehouse praktisch umsetzten kann. Viele dieser Möglichkeiten können in zwei Bereiche eingeordnet werden. In den Bereich der ”traditionellen” Data Warehouses, bei denen die Schichten der Architektur persistiert werden. Und in den Bereich der virtuellen Data Warehouses, bei denen die Schichten der Architektur mehr oder weniger nur logisch beschrieben werden und keine / kaum physische Repräsentationen existieren. Die traditionellen Ansätze versuchen durch eine redundante Datenhaltung die Performanz bei der Verarbeitung von analytischen Anfragen zu optimieren, die abzufragende Präsentationsschicht wird hierbei häufig durch einen multidimensionalen Data Mart repräsentiert [Gluchowski und Chamoni, 2016]. Die virtuellen oder meist semi-virtuellen Ansätze versuchen Redundanzen zu minimieren, indem Prozesse nur logisch beschrieben werden und nur bei Abruf ”on the fly” berechnet werden. Es wird somit Performanz für eine erhöhte Flexibilität und eine schnellere Entwicklung geopfert [van der Lans, 2012].

Diese beiden Ansätze liegen somit auf verschiedenen Enden des Trade-offs zwischen hoher Performanz und hoher Flexibilität.

Dieser Trade-off kann auf die erwähnten Anforderungen einer hohen Effizienz und Skalierbarkeit mit begrenzten Ressourcen abgebildet werden. Der virtuelle Ansatz weist durch die hohe Flexibilität und schnellere Entwicklung Vorteile auf. Er ist eine gute Wahl, wenn die Anforderungen an die Effizienz und Skalierbarkeit zukunftssicher erfüllt werden können. Hierdurch kann eine flexible Data Warehouse Architektur effizient aufgebaut werden. Wenn die Anforderungen die Kapazitäten des Ansatzes jedoch übersteigen sollten, dann ist eine Umsetzung mit einer auf Effizienz optimierten traditionellen Architektur sinnvoller. Flexibilität und eine schnelle Entwicklung bringen einem nichts, wenn benötigte analytische Anfragen nicht rechtzeitig abgeschlossen werden können.

## 1.2 Problem und Forschungsfragen

Das Problem ist nun, dass es schwierig ist den Einfluss der Virtualisierung auf die Performanz einzuschätzen. Zum einen gibt es wenig Material in diesem Bereich, zum anderen gibt es viele Faktoren, die die Auswirkungen der Virtualisierung beeinflussen. Aus diesem Grunde werden in dieser Arbeit die Auswirkungen der Virtualisierung auf die Verarbeitungszeiten von analytischen Anfragen untersucht. Konkret wird ein traditionelles Data Warehouse, bei dem die Präsentationsschicht durch einen multidimensionalen Datenwürfel repräsentiert wird, mit einem semi-virtuellen Data Warehouse verglichen. Das semi-virtuelle Data Warehouse persistiert nur die unverarbeiteten historisierten Quelldaten, sämtliche (E)TL-Prozesse werden virtualisiert. Die Möglichkeiten des virtuellen Ansatzes sollen konkreter und vergleichbarer gemacht werden, indem die folgenden Forschungsfragen beantwortet werden:

- Inwiefern ist ein semi-virtuelles Data Warehouse verglichen mit einem traditionellen Ansatz in der Lage, bei steigenden Datenmengen zu skalieren?
- Welchen Einfluss hat die Virtualisierung eines semi-virtuellen Data Warehouses auf dessen Verarbeitungszeiten bei analytischen Anfragen?
- Ändert sich durch die Virtualisierung der Einfluss von Parametern wie Partitionierung, Anzahl der Quellsysteme und verschiedene Arten von analytischen Anfragen?

## 1.3 Abgrenzung des Themenbereiches

Dies ist ein sehr weit gefächerter Themenbereich. Damit die Arbeit in dem gesetzten Zeitrahmen abgeschlossen werden kann, wird dieser Bereich anhand mehrerer Kriterien eingegrenzt. Zunächst findet eine Eingrenzung anhand der untersuchten Anforderungen statt. In dieser Arbeit liegt der Fokus darauf, den Umgang mit verschiedenen Datenvolumen zu untersuchen. Es wird geschaut, wie gut die untersuchten Ansätze bei steigendem Volumen skalieren. Nicht untersucht werden die Auswirkungen einer steigenden "Velocity" oder "Variety". Durch diese wird beschrieben,

mit welcher Geschwindigkeit neue Daten in den Quellen generiert werden und wie hoch die Varianz der Datenformate ist (mehr dazu in Abschnitt 2.4.1).

Zusätzlich wird die Menge an untersuchten Architekturen eingegrenzt. Es gibt viele Möglichkeiten, ein Data Warehouse mit persistierten Schichten umzusetzen. Die Menge an Schichten und die Art der Umsetzung spielt für die Untersuchung der Verarbeitungszeiten keine Rolle. Nur die Umsetzung der letztendlich abgefragten Präsentationsschicht ist relevant. Hier wird durch die dimensionalen Data Marts ein Ansatz untersucht, der sehr verbreitet ist. Auch für das virtuelle Data Warehouse gibt es viele verschiedene Umsetzungsmöglichkeiten, die durch den Grad der Virtualisierung beschrieben werden können. Diese Arbeit fokussiert sich auf das semi-virtuelle Data Warehouse. Bei diesem werden nur die Ergebnisse der Extraktionsphase persistiert. Sämtliche darauf aufbauende Transformationen werden virtualisiert.

In dieser Arbeit sollen die Architekturansätze verglichen werden und nicht die Technologien. Jedoch haben die verwendeten Technologien einen Einfluss auf die Relevanz der Ergebnisse und werden daher kurz vorgestellt. Die Tests werden bei beiden Architekturen in einer Hadoop Umgebung mit einer verteilten SQL-Query-Engine durchgeführt. Die Architekturen werden soweit es geht mit den selben Technologien und den selben Optimierungen aufgebaut, damit die gemessenen Unterschiede sich möglichst auf die Architekturansätze reduzieren lassen. Die Ergebnisse der Evaluierung lassen sich auch auf andere verteilte SQL-Query-Engines innerhalb von Hadoop übertragen. Auch außerhalb sind die Ergebnisse mehr oder weniger gültig.

## 1.4 Methodische Vorgehensweise

Für die Beantwortung der Forschungsfragen wird die folgende methodische Vorgehensweise gewählt:

Zunächst werden die Quellsysteme definiert und umgesetzt. Um die Einflüsse des Datenvolumens und der Anzahl der Quellen auf die Ergebnisse untersuchen zu können, werden diese als variabel festgelegt. Die Quellen können somit mit variablen Parametern wiederhergestellt werden. Anschließend werden die beiden Architekturen auf diesen Quellsystemen aufgebaut. Sie führen dieselben ETL-Prozesse durch und wenden soweit es geht ähnliche Optimierungen an (Kompression, spaltenorientierte Speicherung, etc.).

Nach der Implementierung werden die Tests durchgeführt. Die untersuchten Parameter werden zu Szenarien zusammengefasst, jedes Szenario repräsentiert eine Kombination von Parametern. Für jedes der vorhandenen sechzehn Szenarien müssen die Quellsysteme und die Data Warehouses mit den jeweiligen Parametern erzeugt werden, anschließend können die Tests ausgeführt werden. Diese Tests bestehen aus einer Menge von analytischen Anfragen mit variabler Komplexität. Zum Schluss werden die Ergebnisse der Tests grafisch dargestellt, interpretiert und evaluiert.

## 1.5 Gliederung der Arbeit

In Kapitel 2 werden alle Informationen präsentiert, die für das weitere Verständnis der Arbeit notwendig sind. Es werden Konzepte und Ansätze erläutert, miteinander verglichen und Entscheidungen begründet. Zu den behandelten Bereichen gehören Data Warehouses, virtuelle Data Warehouses, Big Data und die Überschneidungen zwischen allen.

Anschließend werden in Kapitel 3 die umgesetzten Architekturen vorgestellt. Es wird auf die Quellsysteme und auf die darauf aufbauenden Data Warehouse Architekturen eingegangen. Hierzu gehört ein traditionelles Data Warehouse nach dem Ansatz von Kimball und ein semi-virtuelles Data Warehouse. Diese werden auf einer abstrakten Ebene erläutert und die Details anhand von Beispielen verdeutlicht.

Die Evaluierung und der Vergleich der beiden Architekturen finden anschließend in Kapitel 4 statt. Hierbei wird zunächst die Testumgebung, die zur Messung der Leistung verwendeten Metriken und die Grenzen der Untersuchungen aufgezeigt. Anschließend werden die Ergebnisse der Tests für jeden Ansatz für sich und miteinander ausgewertet. Diese Ergebnisse werden mit den aus der Theorie stammenden Erwartungen abgeglichen, diskutiert und Schlussfolgerungen gezogen. Zum Schluss werden anhand dieser Ergebnisse die gestellten Forschungsfragen beantwortet.

# 2. Grundlagen

In dem folgenden Kapitel werden die Wissensgrundlagen in den behandelten Fachgebieten etabliert, um ein besseres Verständnis des praktischen Teils zu ermöglichen. Es wird ein Überblick über den Stand des Wissens und der Technik in den Bereichen Data Warehouses, Big Data und virtuelle Data Warehouses geschaffen. Die im praktischen Teil verwendeten Ansätze und Technologien werden beschrieben und die Entscheidungen für diese begründet.

Zunächst werden operative und analytische Systeme voneinander abgegrenzt. Es werden die Unterschiede anhand eines praktischen Beispiels aufgezeigt und die Motivation für die Verwendung von zentralen analytischen Datenlagern dargestellt.

Anschließend wird das Konzept eines Data Warehouse vorgestellt, welches eine Möglichkeit ist, solch ein zentrales analytischen Datenlager umzusetzen. Zunächst werden allgemeine Konzepte und Begriffe erläutert, dann werden diese anhand von zwei sehr bekannten traditionellen Architekturen detaillierter beschrieben. Im Anschluss wird die Entscheidung für die im praktischen Teil verwendete traditionelle Architektur begründet. Gegen Ende des Abschnitts werden neuere Ansätze und deren Relevanz für diese Arbeit behandelt.

Nachfolgend wird auf das Vergleichsobjekt, das semi-virtuelle Data Warehouse, eingegangen. Die Motivation für dessen Verwendung, das allgemeine Konzept der Virtualisierung und die Architektur dieses Data Warehouses werden vorgestellt. Zusätzlich wird dieses konkret von den traditionellen Ansätzen abgegrenzt.

Der letzte Abschnitt geht auf den Bereich Big Data und die Überschneidungen zum Bereich der Data Warehouses ein. Es werden die Möglichkeiten der Kombination von diesen beiden aufgezeigt, und beschrieben, inwiefern dies im praktischen Teil geschehen ist und wieso. Zum Schluss wird auf die verwendeten Komponenten eingegangen.

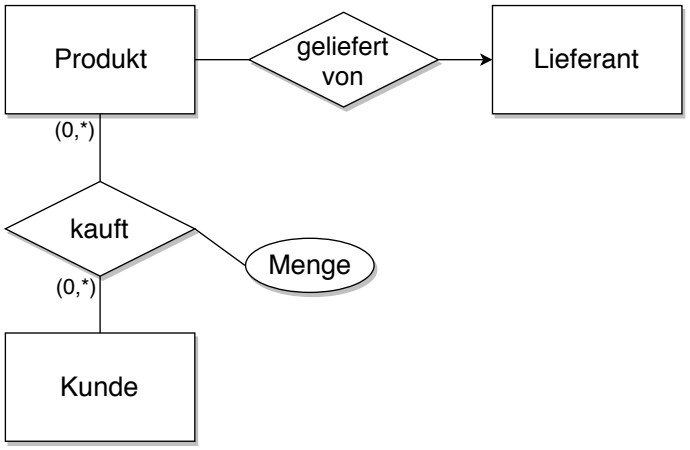



Abbildung 2.1: Datenbankschema einer OLTP-Datenbank
Quelle: [Köppen u. a., 2012]

## 2.1 Operative und Analytische Systeme

Um die Unterschiede zwischen operativen und analytischen Transaktionen aufzuzeigen, werden diese anhand eines Beispiels verdeutlicht. Es geht um einen Getränkemarkt, der Deutschlandweit Filialen besitzt. Aufgrund der Menge an Filialen und der durch diese entstehenden Transaktionen hat man sich dazu entschieden, eine operative Datenbank für jede Filiale anzulegen. Es wurde keine zentralisierte Datenbank verwendet, weil das Risiko eines Ausfalls durch die hohe Last zu groß wäre. Die Geschäftsführung möchte unter anderem nach neuen Standorten für weitere Filialen suchen.

### 2.1.1 OLTP vs. OLAP

**OLTP**
"Online Transactional Processing"(OLTP) charakterisiert Anfragen an Datenbanken, die häufig durch operative Tätigkeiten erzeugt werden.[Köppen u. a., 2012] Eine operative Transaktion ist eine einfache Datenbankanfrage, die meist wenige Datensätze betrifft, möglichst direkt abgeschlossen werden sollte und von vielen Anwendern gleichzeitig ausgeführt werden kann. Meistens handelt es sich dabei um einfache SELECT-, INSERT-, UPDATE- oder DELETE-Anfragen.

Ein praktisches Beispiel ist eine einfache INSERT-Anfrage im operativen Betrieb einer Getränkefiliale aus dem oben erwähnten Getränkemarkt. In Abbildung 2.1 wird ein mögliches Datenbankschema einer solchen Getränkefiliale dargestellt. Kunden können Getränke kaufen, welche von Lieferanten übermittelt werden. Für jeden Einkauf wird zusätzlich die Menge der verkauften Produkte in der Datenbank gesichert. Wenn ein Kunde sich dazu entschieden hat ein Getränk zu kaufen und an die Kasse geht, werden in dem Moment, wo die Geschäftstransaktion abgeschlossen wird (der Kunde hat für die Produkte bezahlt), ein oder mehrere operative Datenbanktransaktionen durchgeführt. Solche Transaktionen müssen einfach und kurz sein, weil viele von ihnen gleichzeitig stattfinden können. Wenn viele Kassen gleichzeitig besetzt sind, senden alle zur selben Zeit Anfragen an die Datenbank. Aufgrund des Arbeitsflusses können sich diese Systeme keine langen Wartezeiten leisen.

**OLAP**
Auf der anderen Seite gibt es "Online Analytical Processing"(OLAP), welches eine Charakterisierung für analytische Anwendungsfälle darstellt. Bei diesen liegt der Fokus auf lang andauernden und komplexen analytischen Lesetransaktionen, die Beispielsweise durch ein Data Warehouse ausgeführt werden können. Die Eigenschaften und Anforderungen an OLTP- und OLAP-Systemen unterscheiden sich dadurch sehr.

In dem bereits erwähnten Beispiel könnte die Geschäftsführung Fragen stellen, für dessen Beantwortung ein OLTP-System ungeeignet wäre. Um zukünftige Entscheidungen besser fällen zu können, möchte die Geschäftsführung wissen, welche Filialen Deutschlandweit den meisten Umsatz abwerfen. Diese analytische Fragestellung benötigt Daten, die über den Datensatz einer operativen Datenbank einer einzelnen Filiale hinausgeht. Man müsste alle Daten aller Filialen integrieren um diese miteinander vergleichen zu können, jede dieser Filialen könnte hierbei eine nach dem Muster von Abbildung 2.1 definierten Datenbankschema haben. Zusätzlich müssen die Daten aggregiert, gruppiert und sortiert werden. Dieses kleine Beispiel macht bereits deutlich, dass OLAP-Transaktionen eine wesentlich höhere Komplexität aufweisen. Zusätzlich müssen meistens viele Datensätze abgefragt werden, was zu einem hohen zu verarbeitenden Datenvolumen führt. Daraus ergibt sich aber auch, dass diese Art von Transaktionen wesentlich seltener vorkommen und durch eine vergleichsweise geringe Menge an Anwendern durchgeführt wird. Zusätzlich handelt es sich hierbei meistens ausschließlich um Lesezugriffe auf die Daten.

Durch diese gänzlich verschiedenen Anforderungen ergeben sich auch unterschiedliche Optimierungspotentiale für Systeme, die diese Transaktionen umsetzen müssen. Ein System kann nicht für OLTP- und OLAP-Transaktionen gleichzeitig perfekt optimiert sein.

### 2.1.2 Möglichkeiten der Datenintegration

Um diese komplexen OLAP-Transaktionen wie die Bestimmung der umsatzstärksten Filialen in einem Gebiet durchführen zu können, müssen die Daten aller Filialen integriert werden. Eine Möglichkeit dieses Problem zu lösen, wäre es für jede Filiale den Umsatz zu berechnen und die einzelnen Ergebnisse zu vereinen. Dieser Ansatz bringt einige Nachteile mit sich. Zum einen haben wir die hohe Belastung der Quellsysteme, schließlich müssen alle Quellen bei jeder Abfrage von neuem angefragt werden. Zum anderen steigt die Komplexität der Anfragen mit jeder neuen Quelle enorm an. Verschiedene Quellen verwenden häufig unterschiedliche Datenbanken und Datenschemata, was wiederum bedeutet, dass dieselbe Anfrage (Umsatz berechnen) sich von Quelle zu Quelle unterscheidet. Bei mehreren Quellen und Reports wird die Erstellung und Wartung der Anfragen mit einem enormen Aufwand verbunden.

Eine Alternative wäre die Verwendung eines Data Warehouses [Inmon, 1996]. Dieses beseitigt die meisten der bisher erwähnten Probleme durch eine redundante Datenhaltung jeglicher Quelldaten. Die Daten werden vor den Analysen in ein einheitliches Schema transformiert, bereinigt und integriert. Dies vereinfacht die Erstellung von Reports und Analysen erheblich, gleichzeitig werden die Quellsysteme nur bei der Extraktion der Daten für die redundante Datenhaltung belastet. Aus dieser redundanten Datenhaltung ergeben sich aber auch Probleme. Zum einen steigt mit jeder

redundanten Schicht der Aufwand und die Komplexität, um alle Daten synchron zu halten. Zum anderen sinkt die Flexibilität der gesamten Architektur. Bei Änderungen (z.B.: in den Quellsystemen) müssen die Transformationsprozesse und sämtliche Kopien der Daten aktualisiert werden.

Eine weitere Alternative wäre ein virtualisiertes Data Warehouse [van der Lans, 2012]. Dieser Ansatz erreicht Anwenderfreundlichkeit und Flexibilität durch eine virtualisierte Datenschicht. Für den Anwender steht eine abstrakte API zur Verfügung, mit der Reports und Abfragen definiert werden können. Die Art und Weise, wie die Daten abgerufen werden, wird durch diese Schicht von dem Nutzer abgekapselt. Der zentrale Unterschied zum oben erwähnten Data Warehouse ist, dass keine oder nur eine begrenzte redundante Datenhaltung stattfindet. Hierdurch wird die Flexibilität der Architektur erhöht. In einigen Fällen (z.B.: die Quellsysteme halten keine historischen Daten) ist eine redundante Speicherung von Daten erforderlich oder hilfreich. Abhängig davon, wie viele Daten tatsächlich zwischengespeichert werden, müssen bei dieser Architektur mehr oder weniger Daten spontan bei Abfragen berechnet werden. Dies hat einen Einfluss auf die Performanz des Data Warehouses und die Belastung der Quellsysteme.

Es gibt hier viele Möglichkeiten, um ans Ziel zu kommen mit den jeweiligen Vor- und Nachteilen. Es gibt nicht *das* Data Warehouse oder *das* virtuelle Data Warehouse. Diese beschreiben eher einige Kernaspekte für einen Ansatz. Es gibt unzählige Variationen, wie man diese letztendlich umsetzen kann.

## 2.2 Data Warehouse

In dem folgenden Abschnitt wird das Konzept des Data Warehouses zunächst generell erläutert. Der Hauptzweck eines solchen Systems wird beschrieben und die nachfolgend benötigten Begriffe definiert. Anschließend werden zwei traditionelle und sehr weit verbreitete Architekturen für Data Warehouses vorgestellt. Konkret wird auf die Architekturen von Kimball und Inmon eingegangen. Anhand dieser wird die für die Evaluierung gewählte Architektur bestimmt und die Entscheidung hierfür erläutert. Zum Schluss werden einige neuere Ansätze wie der Data Vault oder die Umsetzung mithilfe von In-Memory-Datenbanken kurz vorgestellt und deren Relevanz für diese Arbeit geklärt.

### 2.2.1 Einführung

Nachfolgend wird ein genereller Überblick über Data Warehouses gegeben und die Überschneidungen zum Bereich Big Data aufgezeigt.

**Überblick und Kontext**
Data Warehouses sind Systeme für Analysen und Reportings und stellen einen zentralen Teil der Business Intelligence(BI) dar. Sie können zum Beispiel verwendet werden um die Geschäftsführung eines Unternehmen bei der Entscheidungsfindung zu unterstützen. Es sollen Analysen und Reportings ermöglicht werden, die sämtliche Informationen des Unternehmen und teilweise auch unternehmensübergreifende Informationen vereinen und integrieren. Durch diese Menge an Daten werden komplexere und wertvollere Analysen und Schlussfolgerungen ermöglicht. Die Daten, die für die Analysen herangezogen werden, werden aus sogenannten Quellen extrahiert. Dies können relationale Datenbanken, Webseiten oder auch Sensordaten sein.

Um die Quellsysteme nicht durch wiederkehrende Analysen in ihrer alltäglichen Funktion zu beeinträchtigen, bietet sich eine redundante Datenspeicherung an. Dieses zentrale Datenlager wird als *Data Warehouse* bezeichnet. Hier liegen die historisierten und integrierten Daten für die Analysen bereit, es wird auch als "Single Version of Truth" bezeichnet [Köppen u. a., 2012]. Um dieses hochwertige Datenlager zu ermöglichen müssen die Daten zunächst einen sogenannten ETL(Extract / Transform / Load)-Prozess durchlaufen. Dieser liegt zwischen den Quellsystemen und der Data-Warehouse Datenbank. Während dieses aufwendigen Prozesses werden die Daten bereinigt, integriert und in ein einheitliches Schema gebracht, um eine performante Abfrage und hohe Datenqualität zu gewährleisten.

Aufbauend auf dem Data Warehouse befindet sich die Schicht der BI-Anwendungen. Hier befinden sich Tools, die den Anwender dabei unterstützen, Analysen und Reports zu erstellen. Diese Reports werden intern zu Anfragen an das Data Warehouse transformiert, welches wiederum die Ergebnisse liefert die im BI-Tool dargestellt werden.

**Definitionen und Begriffe**
Der Begriff Data Warehouse wurde erstmals von W.H. Inmon im Jahre 1992 definiert:

> *A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process.*

Ein Data Warehouse ist *themenorientiert*, dass heißt es werden Daten für einen bestimmten Geschäfts-/Themenbereich bereitgestellt. Diese sind von Anwendungsfall zu Anwendungsfall unterschiedlich. Die Daten kommen aus unterschiedlichen Quellen, die für gemeinsame Analysen *integriert* werden müssen. Um Analysen über verschiedene Zeitperioden durchführen zu können, werden die Daten *historisch abgespeichert*. Damit können aktuelle Ereignisse mit der Vergangenheit verglichen werden. Diese historischen Daten sind *beständig*, normalerweise sind Änderungen an Daten im Data Warehouse nicht möglich.

Unter *Data Warehousing* werden die Schritte von der Datenbeschaffung (ETL), der Datenspeicherung bis zur Datenanalyse bezeichnet. Der *ETL-Prozess* (Extraction, Transformation, Loading) umfasst alle Schritte, die notwendig sind, um die Daten aus den Quellen zu extrahieren, qualitativ aufzubereiten, zu integrieren und schließlich in den *Datenwürfel* zu laden. Der Datenwürfel ist der zentrale Speicherpunkt des Data Warehouses und ist multidimensional aufgebaut. Die *Dimensionen* liefern den Kontext für die Daten wie zum Beispiel Zeit, Ort und Produkt. Die Daten selbst werden als *Kennzahlen* bezeichnet [Köppen u. a., 2012].

**Big Data und Data Warehousing**
Mit *Big Data* werden Datenmengen beschrieben die zu groß, zu komplex oder zu unstrukturiert sind, um sie mit herkömmlichen Datenverarbeitungsmethoden zu bearbeiten[Köppen u. a., 2012, Ward und Barker, 2013]. Diese Daten können in vielen verschiedenen Formen auftreten. Es können strukturierte, unstrukturierte oder auch unverarbeitete Sensordaten analysiert werden. Big Data wird häufig durch die drei Dimensionen *Volume* (Umfang und Datenvolumen), *Velocity* (Geschwindigkeit mit der neue Daten verarbeitet werden müssen) und *Variety* (Umfang der Datentypen und -quellen) beschrieben [Köppen u. a., 2012]. Datenmengen, die diese Eigenschaften aufweisen, können nur schlecht oder gar nicht von klassischen relationalen Datenbanksystemen verarbeitet werden. Eine Skalierung dieser ist verglichen mit Big-Data Technologien aufwendiger und teurer.

Um trotzdem einen Nutzen aus diesen Datenmengen generieren zu können, wurden und werden neue Technologien entwickelt. ”Big Data” wird hierbei häufig als Sammelbegriff für diese verwendet. Einige Beispiele sind Apache Hadoop auf Basis des MapReduce-Paradigmas, NoSQL-Datenbanken und In-Memory Datenbanken.

Aus Sicht von Data Warehouses ist die Grenze zu Big Data fließend. Es gibt Data Warehouses, die Daten im Petabyte-Bereich verarbeiten können und auch Bild- und Textdateien integrieren können [Köppen u. a., 2012]. Im Kern dieser Data Warehouses befindet sich in den meisten Fällen eine relationale Datenbank oder ein persistentes multidimensionales Array. Häufig werden Apache Hadoop und MapReduce als Unterstützung verwendet, dies wird im Englischen als ”Data Warehouse Offload” bezeichnet [Grover u. a., 2015]. Dabei werden Aufgaben, die von Big Data Lösungen besser adressiert werden können, ausgelagert. Nachfolgend einige Beispiele nach [Grover u. a., 2015]:

- *ETL-Prozesse*:

  Hadoop bietet eine kostengünstige Alternative um komplexe und langwierige ETL-Prozesse zu verarbeiten, auf diese Technologie wird in Abschnitt 2.4

näher eingegangen. Dadurch können mehr Data Warehouse Ressourcen für Nutzeranfragen verwendet werden und Kosten gespart werden. Gleichzeitig eröffnet dieser Ansatz die Möglichkeit, explorative Analysen auf den unverarbeiteten Daten durchzuführen.

- *Archivierung von Daten*

  Verglichen mit Hadoop ist die Skalierung von traditionellen Data Warehouse Systemen aufwendiger und teurer. Darum werden bei sehr großem Datenvolumen Daten historisch archiviert, wodurch diese für Analysen schlecht erreichbar sind. Durch eine Archivierung durch Hadoop wären diese Daten weiterhin für Analysen erreichbar.

- *Explorative Analysen*

  Viele Daten werden aufgrund ihrer Struktur, Größe oder der Unsicherheit über den generierten Nutzen nicht in ein Data Warehouse aufgenommen. Durch Big Data Technologien können diese Daten kostengünstiger untersucht werden und potentiell Nutzen generiert werden.

Durch diese Überschneidungen von Data Warehouse- und Big Data-Technologien ergeben sich auch Überschneidungen bei den Anforderungen an diese Systeme. Der Umgang mit Datenmengen deren *Volume*, *Velocity* und *Variety* steigt muss gegeben sein, um einen Nutzen generieren zu können. Dies gilt für alle Arten und Variationen von Data Warehouses. Das virtuelle Data Warehouse beseitigt die Nachteile von traditionellen Data Warehouses (geringe Flexibilität durch die redundante Datenhaltung) indem die Transformationen "on the fly" erledigt werden und Zwischenergebnisse im Hauptspeicher gelagert werden. Hieraus ergeben sich jedoch Nachteile für die Performanz und Skalierbarkeit dieser Systeme. Weil die oben genannten Anforderungen auch für virtuelle Data Warehouses gültig sind, wird in dieser Arbeit ein Vergleich der Performanz und Skalierbarkeit eines virtuellen Data Warehouses mit der eines traditionellen verglichen.

Die neuesten Entwicklungen im Bereich Big Data und Hadoop bieten Funktionalitäten, die die Kernfunktionalitäten eines Data Warehouses erfüllen können. Dazu gehören unter anderem eine zuverlässige Sicherung der Daten und kurze Antwortzeiten auf Nutzeranfragen [Grover u. a., 2015]. Auch eine Integration mit beliebten Business Intelligence und Visualisierungstools wie zum Beispiel Tableau[1] wird unterstützt. Damit wäre es potentiell möglich, ein komplettes Data Warehouse durch Big Data Technologien umzusetzen. Aufgrund der effizienteren Skalierbarkeit von Big Data-Technologien und der potentiellen Möglichkeit, die Kernfunktionalitäten eines Data Warehouses mit Big Data-Technologien umsetzen zu können, werden die beiden untersuchten Ansätze mithilfe von diesen umgesetzt. Hierbei ist natürlich wichtig zu erwähnen, dass eine Umsetzung eines traditionellen Data Warehouses mit Big Data-Technologien nicht traditionell ist. Die verwendeten Ansätze und Konzepte sind traditionell, die verwendeten Technologien zur Umsetzung jedoch nicht. Dies wird hier billigend in Kauf genommen, da es um einen Vergleich der Konzepte und nicht um einen Vergleich der Technologien geht. Eine ausführliche Erläuterung

[1] https://www.tableau.com/de-de

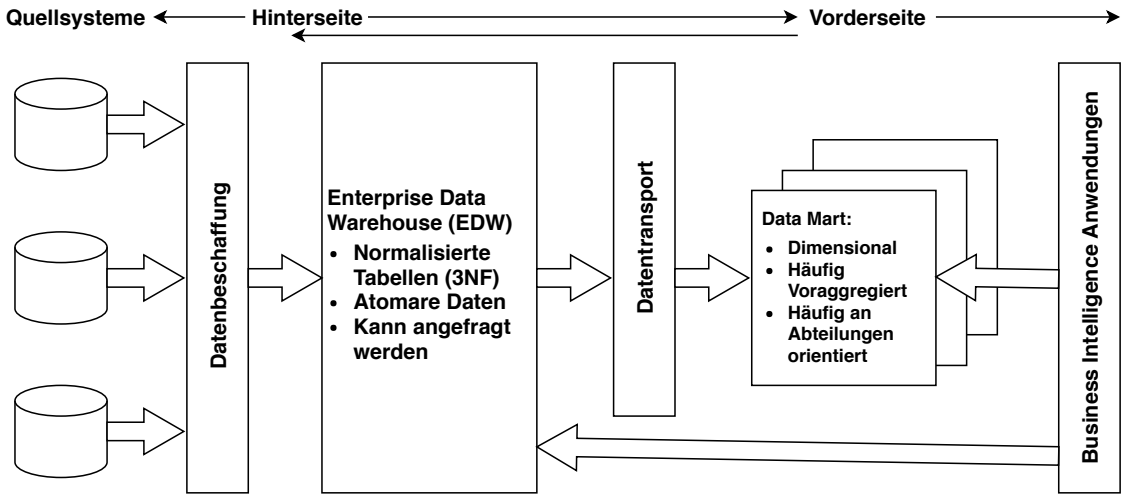



Abbildung 2.2: Vereinfachte Darstellung der CIF Architektur nach Inmon
Quelle: In Anlehnung an Inmon [1996]

und Begründung der Entscheidungen rund um Hadoop und Big Data findet in Abschnitt 2.4 statt. Die Auswirkungen dieser Entscheidungen auf die Relevanz der Ergebnisse werden in Abschnitt 4.1.4 erläutert.

### 2.2.2 Architektur

In diesem Unterabschnitt soll ein Überblick über einige mögliche Architekturen für Data Warehouse Systeme geschaffen werden. Es werden die zwei verbreitetsten Architekturen von Bill Inmon und Ralph Kimball und einige weitere mögliche Ansätze erläutert. Bill Inmon wird auch als "Vater" des Data Warehouses bezeichnet, weil er im Jahre 1990 mit seinem Buch die ersten Ansätze in diesem Bereich veröffentlicht hatte. Ralph Kimball wurde durch sein Buch "Data Warehouse Toolkit" im Jahre 1996 bekannt. Beide repräsentieren jeweils eine andere Denkrichtung und Herangehensweise an Data Warehouse Systeme. Eine Gemeinsamkeit, die sie teilen ist, dass ihrer Meinung nach ein ETL-Prozess für ein Data Warehouse, das in der Produktion eingesetzt werden soll, erforderlich ist [Breslin, 2004]. Beide Ansätze werden nachfolgend etwas ausführlicher erklärt.

#### 2.2.2.1 Architektur nach Inmon

Der *Corporate Information Factory* (CIF) Ansatz von Bill Inmon wird in Abbildung 2.2 vereinfacht dargestellt. Bis auf die Quellsysteme auf der linken Seite gehören alle Elemente zum Data Warehouse. Daten werden aus den Quellsysteme extrahiert und durch das ETL-System verarbeitet, dies wird häufig auch als "Data Acquisition" bezeichnet [Inmon, 1996]. Die transformierten Daten werden anschließend auf einer atomaren Ebene[2] integriert und in einer Datenbank in 3NF[3] zwischengespeichert. Diese Datenbank wird als *Enterprise Data Warehosue* (EDW) bezeichnet und beinhaltet sämtliche Daten des Unternehmens [Inmon, 1996]. Diese Datenbank kann abgefragt werden, aufgrund des Aufbaus und der großen Menge atomarer Daten sind

[2]feinste Granularität, kann nicht weiter aufgespalten werden
[3]Dritte Normalform

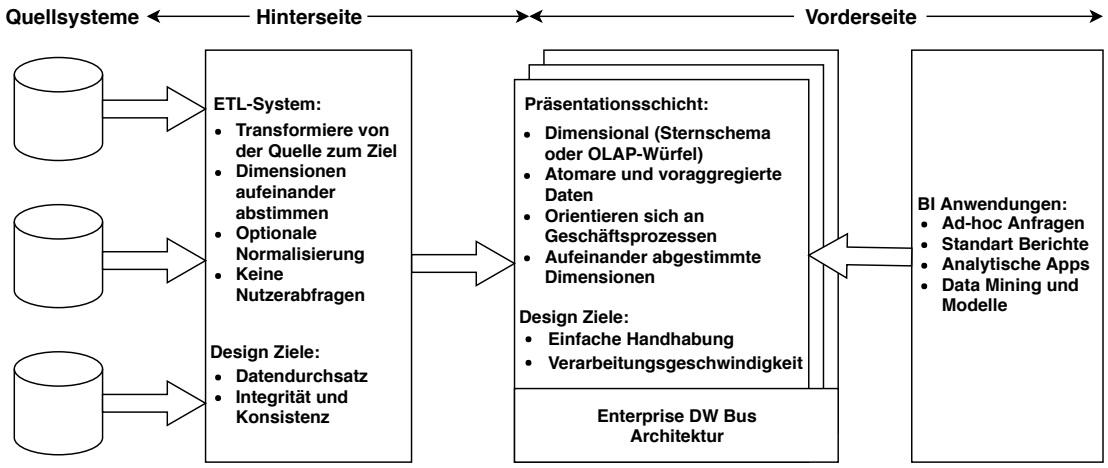



Abbildung 2.3: Architektur des Data Warehouses nach Kimball
Quelle: In Anlehnung an Kimball und Ross [2013]

jedoch nur sehr simple Anfragen möglich. Die bisher erwähnten Systeme (Quellsysteme, ETL-Systeme, EDW) werden nur selten oder gar nicht durch die Endanwender gesehen, bilden aber das Fundament für alles weitere. Sie befinden sich sozusagen auf der "Hinterseite" der gesamten Architektur.

Was in der Praxis durch die Nutzer abgefragt wird sind kleinere Datenbanken, die aus dem großen EDW abgeleitet werden. Diese nennt man *Data Marts*. In der Architektur nach Inmon werden diese häufig für unterschiedliche Abteilungen mit unterschiedlichen Anforderungen gebildet. Sie können zum Beispiel nur einen Teil des gesamten Datensatzes herausfiltern oder Datensätze bereits voraggregieren, um die Effizienz von Anfragen zu verbessern. Auch können Daten, die aus verschiedensten Gründen nur für einen Teil des Unternehmens gedacht sind, für alle anderen unsichtbar gemacht werden indem sie in deren Data Marts nicht aufgenommen werden. Dadurch, dass diese Data Marts aus einem zentralen großen Data Warehouse abgeleitet sind, soll die Integrität der Daten und Ergebnisse sichergestellt werden. Was aggregiert oder herausgefiltert werden soll, wird von den jeweiligen Abteilungen bestimmt, die die Data Marts anschließend verwenden. Wenn im Nachhinein festgestellt wird, dass durch die Aggregationen benötigte Daten verloren gegangen sind, kann immer noch das EDW angefragt werden. Komplexe Analysen sind aufgrund des Aufbaus dieser Datenbank jedoch nicht möglich.

#### 2.2.2.2 Architektur nach Kimball

Die Kernidee bei der Architektur nach Kimball ist die Technik der Dimensionalen Modellierung. Das Ziel davon ist es Datenbanken zu bauen, die leicht zu verstehen sind und eine schnelle Verarbeitung von Anfragen bieten [Kimball und Ross, 2013]. Diese Technik beschreibt eine simple Methode um Datenbanken für analytische Anwendungsfälle zu modellieren. Simplizität ist wichtig um sicherzugehen, dass Anwender die Daten leicht nachvollziehen können und die Daten schnell verarbeitet werden können [Kimball und Ross, 2013]. Außerdem ist die Wahrscheinlichkeit, dass ein fertiges Datenmodell leicht zu verstehen ist wesentlich größer, wenn die verwendete Datenmodellierungstechnik selbst leicht zu verstehen ist.

Die Daten werden bei dieser Methode durch einen mehrdimensionalen Datenwürfel dargestellt, dessen Dimensionen (Zeit, Produkt, Kunde, ...) den Kontext und die Zellen die Fakten in dem Kontext enthalten. Diese Darstellung ist vor allem für die Geschäftsführung intuitiv verständlich, weil diese es gewohnt sind, Probleme mit diesen Dimensionen zu beschreiben. Auch die Möglichkeit sich den Datenwürfel visuell vorstellen zu können vereinfacht das Verständnis dieses Datenmodells [Kimball und Ross, 2013]. Um nun an bestimmte Informationen innerhalb dieses Datenwürfels zu gelangen, wird dieser anhand der Dimensionen durchschnitten. Hierdurch entsteht ein Teilwürfel mit den selektierten Daten. Dieser kann nun bei Bedarf aggregiert, sortiert oder anderweitig weiterverarbeitet werden um an die gewünschten Informationen zu gelangen.

Die Daten in diesen Datenwürfeln werden auf einer sehr feingranularen Ebene gespeichert [Kimball und Ross, 2013]. Dies bietet langfristig ein Maximum an Flexibilität, weil keine Daten verloren gehen. Dies ist wichtig, weil sich Anforderungen an Analysen ständig ändern können. Um in der Praxis einen maximalen Nutzen erbringen zu können, werden die Würfel anhand von Geschäftsprozessen modelliert dessen Daten für Analysen verwendet werden sollen. Es wird geschaut, welche Fakten durch einen Geschäftsprozess generiert werden, und welche Informationen benötigt werden, um Anfragen an diese Daten erstellen zu können. Anhand dieses Wissens werden anschließend die Fakten und Dimensionen des Datenwürfels festgelegt [Kimball und Ross, 2013].

Damit jeder Datenwürfel nicht ein in sich geschlossenes System bildet, sondern alle zusammenhängen und als Gesamtmenge ein Data Warehouse darstellen, orientiert man sich an der *Data Warehouse Bus Architecture* [Kimball und Ross, 2013]. Durch diese wird es möglich, Ähnlichkeiten zwischen unterschiedlichen Analysen auf Geschäftsprozessen zu finden wie sich ähnelnde oder übereinstimmende Dimensionen. Durch diese Art der Modellierung können Dimensionen zwischen Datenwürfeln wiederverwendet und kombiniert werden, wodurch die Menge an Datenwürfeln im Endeffekt das Data Warehouse nach Kimball darstellt.

In Abbildung Abbildung 2.3 wird die Data Warehouse Architektur nach Kimball grob dargestellt. Die anfänglichen Schritte der Extraktion der Daten aus den Quellen und der Transformation der Daten (bereinigen, integrieren, ...) sind bei den Architekturen von Inmon und Kimball sehr ähnlich. Unterschiede gibt es bei der Art der Speicherung der Daten und der Darstellung für die Endnutzer. Im Gegensatz zu der Architektur nach Inmon gibt es hier kein EDW. Es gibt also keine zentrale normalisierte Datenbank, die alle Daten vereint. Stattdessen werden die Daten nach den ETL-Prozessen direkt in die Datenwürfel geladen. Diese Ebene der Datenwürfel, die nach dem bereits erwähnten Dimensionalen Datenmodell modelliert sind, wird ”Präsentationsschicht’ genannt. Diese kann anschließend direkt durch die BI Applications abgefragt werden.

**Praktische Umsetzung eines Datenwürfels**

Diese Datenwürfel speichern die Daten für Queries optimiert zwischen. Es gibt hierbei zwei sehr populäre Ansätze der praktischen Umsetzung, das *Star Schema* und der *OLAP* Cube [Kimball und Ross, 2013]. Bei dem Star Schema Ansatz wird der Datenwürfel durch eine relationale Datenbank umgesetzt, dessen Tabellen sternförmig

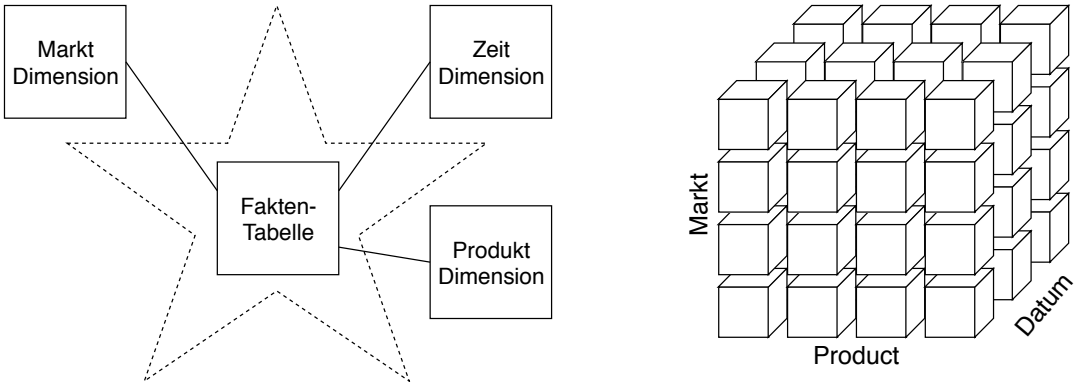



Abbildung 2.4: Star-Schema und OLAP-Würfel
Quelle: In Anlehnung an Kimball und Ross [2013]

strukturiert sind wie in Abbildung 2.4 dargestellt. Mittig befindet sich die Faktentabelle, die mit den außenstehenden dimensionalen Tabellen verbunden werden kann um einen Kontext zu bekommen. Mit diesem Kontext können die Daten selektiert, gruppiert oder anderweitig verarbeitet werden.

Auf der rechten Seite derselben Abbildung wird derselbe Datenwürfel im OLAP Speicherverfahren dargestellt. Bei dieser Methode werden die Fakten in einem mehrdimensionalen Array gespeichert, die Dimensionen des Datenwürfels bilden die Dimensionen des Arrays. Diese Methode ist eine intuitivere Umsetzung des Dimensionalen Datenmodells und ist für OLAP Operationen besser optimiert [Köppen u. a., 2012]. Andererseits hat diese Methode auch Nachteile, wie ein erhöhter Speicherverbrauch bei häufig vorkommenden spärlich befüllten Datenwürfeln [Köppen u. a., 2012]. Eine genauere Ausführung der Methoden und Vor-/Nachteile würde an dieser Stelle den Rahmen sprengen.

**Bottom-Up Approach**
Kimball und Inmon verwenden sehr unterschiedliche Herangehensweisen an die Modellierung des Data Warehouses. Kimball verwendet einen *Bottom-Up* Ansatz, bei dem ausgehend von den Geschäftsprozessen ein Data Warehouse aufgebaut wird. Um dies zu erreichen stellt Kimball einen vier Schritte Design Prozess vor, aus dem ein Datenwürfel entstehen soll. Dieser erlaubt Analysen auf den Transaktionen des modellierten Geschäftsprozesses [Kimball und Ross, 2013]. Für jeden Geschäftsprozess wird so iterativ ein Datenwürfel erzeugt, wobei die neuen Datenwürfel auf den alten Aufbauen und die Dimensionen wiederverwenden können. Dadurch wird das Data Warehouse basierend auf den Geschäftsprozessen Schritt für Schritt aufgebaut. Dieser vier Schritte Design Prozess beinhaltet die folgenden Elemente:

1. Geschäftsprozess auswählen
2. Granularität bestimmen
3. Dimensionen festlegen
4. Fakten festlegen

Im ersten Schritt soll der zu analysierende Geschäftsprozess ausgewählt werden, auf dessen Transaktionen Analysen durchgeführt werden sollen. Es ist empfehlenswert hierbei mit dem wichtigsten Geschäftsprozess anzufangen, der den meisten Nutzen generieren könnte. Die Daten aus ihm sollten möglichst zu den aktuellen Fragen der Geschäftsführung passen [Breslin, 2004]. Kimball definiert den Begriff Geschäftsprozess dabei sehr weit. Beispiele wären Geschäftstransaktionen im Einzelhandel (Point of Sale), Onlinebestellungen oder die Auslieferung eines Produktes [Breslin, 2004].

Anschließend wird die Granularität von festgehaltenen Transaktionen dieses Geschäftsprozesses bestimmt. Die feinste Granularität wird atomar genannt und bietet langfristig die höchste Flexibilität, weil keine Daten durch Aggregationen verloren gehen [Breslin, 2004]. Durch diese Ebene ergeben sich aber auch höhere Datenmengen und damit auch ein höherer Rechenaufwand. Es wird empfohlen, wenn möglich, auf der atomaren Datenebene zu arbeiten. Ein Beispiel für eine atomare Granularität im Einzelhandel wäre ein Produkt aus einem Warenkorb eines Kunden. Solch eine Transaktion kann nicht weiter unterteilt werden. Eine aggregierte Form einer Transaktion könnte zum Beispiel der gesamte Warenkorb eines Kunden sein, hierbei können die einzelnen Produkte nicht mehr voneinander unterschieden werden. Eine weitere Möglichkeit wäre es für jedes Produkt alle Verkäufe eines Tages zusammenzufassen. Diese weniger feinen Granularitäten können den Spielraum bei Analysen enorm reduzieren.

Nachdem die Granularität festgelegt wurde, werden die Dimensionen bestimmt. Man möchte die Dimensionen so wählen, dass alle gewünschten Fragen an die Daten beantwortet werden können. Da die Dimensionen den Kontext für die Fakten legen, bedeuten umfangreichere Dimensionen eine höhere Flexibilität bei der Gestaltung von Anfragen an die Daten. Außerdem ist die Anzahl der Zeilen bei den Dimensionstabellen meist vergleichsweise gering, auch bei einer großen Menge an beschreibenden Spalten ist die Größe der Gesamttabelle klein. Bei den Point of Sale Transaktionen im Einzelhandel wären typische Dimensionen Produkt, Zeit, Geschäft oder Angebot. Jede dieser Tabellen ist denormalisiert und enthält alle Attribute zur Beschreibung einer Transaktion. Die Tabelle Produkt beinhaltet beispielsweise in jeder Zeile ein Produkt, wobei jede Spalte eine Eigenschaft dieses Produktes wiedergibt wie Name oder Produktkategorie.

Zum Schluss werden die zu speichernden Fakten in der festgelegten Granularität bestimmt. Dabei können sowohl gemessene Werte der Transaktionen als auch berechnete Werte gespeichert werden [Breslin, 2004]. Häufig benötigte Fakten können hier pro Zeile vorberechnet werden um langfristig die Verarbeitungsgeschwindigkeit zu verbessern. Fakten, die im Einzelhandel aufgenommen werden könnten, wären zum Beispiel die Anzahl der verkauften Produkte dieses Produkttyps und dessen kummulierter Verkaufspreis. Ein Fakt, der aus Performanzgründen vorberechnet werden könnte, wäre zum Beispiel der Nettoverkaufspreis bei gegebenem Bruttoverkaufspreis.

Die bereits erwähnte *Bus Architecture* ist nun dafür da, die in jedem Schritt dazukommenden Datenwürfel aufeinander abzustimmen, Dimensionen wiederzuverwenden und damit ein zusammenhängendes und integriertes Data Warehouse zu erzeugen.

#### 2.2.2.3 Unterschiede Inmon / Kimball und Entscheidung

Der Hauptunterschied zwischen den beiden Ansätzen ist die Art und Weise, wie das Data Warehouse aufgebaut und strukturiert wird. Inmon verwendet hier einen Top-Down Ansatz, bei dem das Data Warehouse von oben geplant und aufgebaut wird. Dies ist notwendig, weil das EDW, welches sämtliche Daten des Unternehmens vereinen und integrieren soll, für die darauffolgenden Data Marts benötigt wird [Breslin, 2004]. Dadurch ergibt sich ein erhöhter Entwicklungsaufwand in der Anfangsphase. Es könnten Teile der Planung mit der Zeit durch neue Erkenntnisse verworfen werden, weil es schwierig ist weit in die Zukunft zu planen. Gleichzeitig könnten in dieser komplexen Planungsphase aber Probleme entdeckt werden, die bei einem iterativen Ansatz erst später aufgedeckt werden würden.

Im Gegensatz dazu steht der Bottom-Up Ansatz von Kimball. Bei diesem wird iterativ von Geschäftsprozess zu Geschäftsprozess das Data Warehouse um neue Datenwürfel erweitert. Durch diesen iterativen Ansatz ist es leichter bei der Planung und Entwicklung den Überblick zu behalten. Gleichzeitig ist es möglich, nach kürzerer Zeit ein erstes lauffähiges Data Warehouse für einige wichtige Geschäftsprozesse aufzustellen [Breslin, 2004]. An dieser Stelle liegt die Herausforderung eher darin, die Datenwürfel so aufeinander abzustimmen, dass die Menge dieser ein integriertes Data Warehouse darstellt.

Es gibt noch sehr viele weitere Unterschiede zwischen diesen beiden Ansätzen. Beispielsweise verwendet Inmon einen Modellierungsansatz, der eher Themen- und Datenorientiert ist, während Kimball eher Prozessorientiert modelliert [Breslin, 2004]. Eine umfangreichere Ausführung der Unterschiede gibt es in dem Paper von Breslin [2004]. Beide Ansätze haben ihre eigenen Vor- und Nachteile und natürlich muss man sich nicht für einen entscheiden und diesen strikt befolgen. Es gibt auch Hybride, die versuchen, die Vorteile von beiden Ansätzen zu kombinieren. Es gibt auch neuere Ansätze, die weiter von den bisher erwähnten Abweichen. Einige hiervon werden in dem letzten Unterabschnitt erwähnt.

In der Evaluierung dieser Thesis wird das semi-virtuelle Data Warehouse mit einem dimensionalen Datenwürfel nach dem Ansatz von Kimball verglichen. Die Entscheidung hierfür wurde basierend auf dem bereits erwähnten Use-Case eines Getränkemartkes getroffen. Der Use Case beinhaltet nur einen zu analysierenden Geschäftsprozess, der sich sehr gut durch einen dimensionalen Datenwürfel repräsentieren lässt. Hierdurch konnte schnell ein effizientes Data Warehouse implementiert werden, welches für den Vergleich verwendet werden kann. Zusätzlich ist eine Repräsentation der Präsentationsschicht durch einen dimensionalen Datenwürfel im Sternschema weit verbreitet, bietet eine gute Abfrageeffizienz und stellt ein Best Practice dar [Gluchowski und Chamoni, 2016].

### 2.2.3 Komponenten der Architektur

Nachfolgend werden die grundlegenden Komponenten eines Data Warehouses und deren Funktionen zusammenfassend erläutert. Diese Komponenten und Prozesse können bei verschiedenen Ansätzen unterschiedlich umgesetzt werden, die Kernidee hinter diesen bleibt jedoch gleich.

**Quellen**
Die Basis für jedes Data Warehouse bilden die Quellen. Sie liefern die Daten und damit den Inhalt, auf dem die Analysen durchgeführt werden. Ohne Quellen mit wertvollen Daten erzeugt ein Data Warehouse keinen Nutzen. Diese Quellsysteme liegen außerhalb des Data Warehouses und sind in der Praxis häufig relationale Datenbanken die OLTP-Transaktionen verarbeiten. Die Daten werden in diesem Fall hauptsächlich durch Geschäftstransaktionen generiert. Dadurch hat das Data Warehouse wenig oder keine Kontrolle über den Inhalt oder das Format der Daten in den Quellen [Kimball und Ross, 2013]. Ein Beispiel hierfür wäre wieder der Getränkemarkt. Jede Filiale besitzt ihre eigene relationale Datenbank um ihre Geschäftstransaktionen zu verwalten. Es fallen Beispielsweise Transaktionen an, wenn Waren verkauft werden oder wenn neue Waren geliefert und im Lager zwischengelagert werden. Es können aber auch viele andere Systeme als Quelle verwendet werden, wichtig ist nur eine Schnittstelle für den Zugriff auf die benötigten Daten. Beispiele wären eine Website die durch eine REST-API Zugriff auf interne Daten gewährt, Daten innerhalb eines Hadoop Clusters oder eine NoSQL Datenbank.

Eines der Probleme welches durch ein Data Warehouse gelöst werden muss, ist die Zusammenführung der Daten aus allen Quellen. Dies ist aufgrund der potentiellen Heterogenität der Quellen eine große Herausforderung. Allein durch die Möglichkeit der autonomen Entscheidung über zu verwendende Datenmodelle ergeben sich verschiedene Möglichkeiten, ein und dasselbe Problem auf verschiedene Arten darzustellen. Zusätzlich können innerhalb eines Datenmodells Ideen unterschiedlich modelliert werden. Auch das verwendete System zur Speicherung kann sich unterscheiden (SQL, NoSQL, ...) und es können verschiedene Datentypen verwendet werden. Diese Entwurfsautonomie führt zu divergenten Modellen [Köppen u. a., 2012]. Das heißt, Quellen die für ähnliche Probleme konzipiert sind, können trotzdem sehr heterogen aufgebaut sein. Ein weiteres Problem ist das Quellen häufig keine historischen Daten sammeln. Um Analysen über verschiedene Zeitperioden durchführen zu können, muss das Data Warehouse diese Aufgabe übernehmen. Zusätzlich sind Quellen häufig durch die alltäglichen operativen Tätigkeiten bereits ausgelastet [Köppen u. a., 2012]. Das Bedeutet, dass das Data Warehouse seine Aufgaben so erfüllen muss, dass die Quellen nicht belastet werden.

**ETL-Prozesse**
Die Hauptaufgabe der ETL-Prozesse ist es die Daten aus den Quellen zu extrahieren und diese für Analysen vorzubereiten. Das Data Warehouse soll kontinuierlich mit frischen Daten versorgt werden. Dieser Prozess kann in drei Phasen unterteilt werden: die Datenextraktion, die Datentransformation und das Laden der Daten in die Data Marts oder das EDW. Dabei stehen sich zwei Anforderungen an die ETL-Prozesse gegenüber: Die Minimierung der Sperrzeiten des Data Warehouses und die Bereitstellung von hochqualitativen Daten für das Data Warehouse [Köppen

u. a., 2012]. Mit umfangreicheren Prozessen kann eine höhere Datenqualität erreicht werden, gleichzeitig kann dadurch das Data Warehouse davon abgehalten werden Analysen durchzuführen.

In der ersten Phase werden die Rohdaten aus den Quellen *extrahiert* und in das Data Warehouse zur Weiterverarbeitung übertragen. Es gibt verschiedene Möglichkeiten das Data Warehouse darauf hinzuweisen, dass neue Daten in einer Quelle vorliegen und die Extraktion gestartet werden soll. Man unterscheidet grob zwischen zwei Varianten: der synchronen und der asynchronen Benachrichtigung [Köppen u. a., 2012]. Bei der synchronen Benachrichtigung propagieren die Quellen jede Änderung an den Daten direkt an das Data Warehouse weiter. Durch diese Methode sind die Daten stets aktuell, die Belastung für die Quellen und das Data Warehouse ist aber hoch. Bei der asynchronen Benachrichtigung kann das Data Warehouse periodisch, ereignisgesteuert oder anfragegesteuert über Änderungen informiert werden. Anschließend werden die Daten in das Data Warehouse extrahiert. Beispiele hierfür wären die wöchentliche Extraktion (Periodisch), die Extraktion bei X Änderungen an einer Quelle (Ereignisgesteuert) oder bei einer konkreten Anfrage durch einen Nutzer.

In der nächsten Phase steht nun die *transformation* der Daten an, um die Qualität und Kompatibilität mit dem Data Warehouse zu gewährleisten. In dieser Phase können sehr viele Arten der Verarbeitung stattfinden, dazu gehören die Homogenisierung (Daten in ein einheitliches Format überführen), Integration (Historische/aktuelle Daten aus verschiedenen Quellen zusammenführen) und Bereinigung (Fehlerhafte Daten Reparieren, Deduplizieren, ...) der Daten. Eine genauere Ausführung findet sich in Kimball und Ross [2013], Köppen u. a. [2012].

Die letzte Phase ist nun dafür zuständig die fertig transformierten Daten in das Enterprise Data Warehouse (nach Inmon) oder die dimensionalen Datenwürfel (nach Kimball) zu *laden* und diese damit für Analysen freizugeben [Kimball und Ross, 2013]. Hierbei werden Methoden verwendet, die diesen Prozess möglichst effizient abschließen um das Data Warehouse nicht unnötig lang zu blockieren (Bulk Loading) [Köppen u. a., 2012]. Es können Sicherheitschecks wie die Tests für die Integrität von Verweisen deaktiviert werden, um die Effizienz weiter zu erhöhen. Es wird schließlich davon ausgegangen, dass die transformierten und bereinigten Daten bereits korrekt vorliegen. Nach Abschluss der ETL-Prozesse liegen die aktuellsten Daten im Data Warehouse für Analysen bereit.

**Präsentationsschicht**
Die Präsentationsschicht stellt den Teil der Architektur dar, in dem die Daten für Anfragen vorbereitet vorliegen. Der Aufbau dieses Bereichs orientiert sich an den Analysebedürfnissen [Köppen u. a., 2012]. Es gibt verschiedene Möglichkeiten die Daten zu speichern. Dazu gehören die folgenden:

- *ROLAP*

  Beim ROLAP-Ansatz (relationales OLAP) werden die Daten in einer relationalen Datenbank für analytische Anfragen bereitgestellt. Ein Beispiel hierfür wäre das Sternschema nach Kimball. Aufgrund der Verfügbarkeit und Performanz von RDBMS Systemen ist dies der verbreitetste Ansatz [Köppen u. a., 2012].

- *MOLAP*

  Beim MOLAP-Ansatz (multidimensionales OLAP) wird die multidimensionale Modellierung nicht nur bei der Konzeption, sondern auch bei der Implementierung angewendet [Köppen u. a., 2012]. Hierbei wird der Datenwürfel intern als multidimensionales Array dargestellt.

- *HOLAP*

  Der HOLAP-Ansatz versucht die Vor- und Nachteile der beiden anderen Ansätze durch eine Kombination auszugleichen. Hierbei wird ein Teil der Daten in einer relationalen Datenbank und ein anderer in einem multidimensionalen Array gespeichert.

In der Architektur nach Inmon wird die Präsentationsschicht durch die einzelnen Data Marts umgesetzt. Bei Kimball stellen die multidimensionalen Datenwürfel diese Schicht dar.

**BI Application**
Die "Business Intelligence Application" Schicht stellt die Menge an Möglichkeiten dar, mit denen Analysen auf den in der Präsentationsschicht vorliegenden Daten durchgeführt werden können.

Hierbei können die verschiedensten Arten von Anfragen generiert werden, aus denen *Standard Reports*, *Ad Hoc Queries* oder auch komplexe *Data Mining* Anwendungen umgesetzt werden können [Kimball und Ross, 2013]. Die Erstellung von Reports und Analysen kann durch externe Tools unterstützt werden, wenn das Data Warehouse eine Schnittstelle für den externen Zugriff bereitstellt. Ein Beispiel für solch eine Schnittstelle wäre die ODBC API, welche ein standardisiertes Interface für den Zugriff auf Datenbanksysteme darstellt. Ein Beispiel für ein externes Tool wäre Tableau, welches die Erstellung von Anfragen und die Visualisierung der Ergebnisse vereinfacht.

### 2.2.4 Neuere Architekturen

Die beiden vorgestellten Architekturen nach Kimball und Inmon haben den entscheidenden Nachteil, dass Änderungen an den Quellen oder an den Anforderungen an die Analysen in einem hohen Entwicklungsaufwand resultieren. Sowohl das Enterprise Data Warehouse nach Inmon, als auch die Data Warehouse Bus Architecture nach Kimball werden durch solche Änderungen betroffen und müssen angepasst werden. Der Grund hierfür liegt in dem Aufbau dieser zentralen Datenlager, welche auch *Core Data Warehouse* genannt werden [Gluchowski und Chamoni, 2016].

Diese Architekturen haben über die Zeit Entwicklungen erfahren, welche unter anderem die Flexibilität von diesen erhöhen. Eine dieser Entwicklungen bildet der *Data Vault*. Der Data Vault bietet eine Methode zur Modellierung von Data Warehouse Systemen und damit eine Alternative zur Bus-Architecture und zum Enterprise Data Warehouse. Durch den Data Vault wird es möglich, das Core Data Warehouse auf eine Art und Weise zu modellieren, durch welche es leichter und schneller angepasst werden kann und damit flexibler ist. Im Gegensatz zu den bisher erwähnten Ansätzen speichert der Data Vault die Schlüssel, deskriptive Attribute und

Beziehungsinformationen in getrennten Tabellen [Gluchowski und Chamoni, 2016]. Hierdurch ist eine flexible Persistierung in einem zentralen Core Data Warehouse möglich. Für analytische Anfragen ist dieses Core Data Warehouse ähnlich wie das Enterprise Data Warehouse weniger geeignet. Für diese werden Data Marts aus dem Data Vault abgeleitet. Ein tieferer Einblick in diesen Ansatz wird von Linstedt und Olschimke [2015] gegeben.

Diese Entwicklungen sind für die Evaluierung dieser Arbeit nicht relevant, weil die für die Präsentationsschicht verwendeten Data Marts letztendlich immer noch auf dem Sternschema basieren, welches in dieser Arbeit für die Evaluierung verwendet wird. Für die Verarbeitungszeit von analytischen Anfragen macht es keinen Unterschied, ob das Core Data Warehouse ein Data Vault, ein Enterprise Data Warehouse ist oder gar nicht existiert. Hier zählt nur die Effizienz der Data Marts in der Präsentationsschicht. Und für diese ist das in dieser Arbeit untersuchte Sternschema immer noch ein Best Practice [Gluchowski und Chamoni, 2016]. Neuerungen in der Präsentationsschicht sind an dieser Stelle jedoch relevant.

Eine dieser neuen Entwicklungen stellt das bereits erwähnte virtuelle Data Warehouse [Gluchowski und Chamoni, 2016, van der Lans, 2012] oder semi-virtuelle Data Warehouse [Gluchowski und Chamoni, 2016] dar, welche durch neue In-Memory-Datenbanken an Relevanz gewinnen. Bei diesen Ansätzen werden die Schichten des Data Warehouses mehr oder weniger durch virtuelle Schichten repräsentiert, welche rein logisch modelliert sind und nur optional persistiert werden können (Caching). Das semi-virtuelle Data Warehouse wird in der Evaluierung dieser Arbeit mit einem nach dem Sternschema modellierten Data Mart verglichen. Bei diesem Ansatz werden nur die Ergebnisse der Extraktionsphase persistiert, alles andere wird virtualisiert [Gluchowski und Chamoni, 2016]. Die Verwendung eines virtuellen Data Warehouses in Kombination mit einer In-Memory-Datenbank wäre ein weiteres interessantes zu untersuchendes Szenario.

## 2.3 Virtuelles Data Warehouse

Der folgende Abschnitt präsentiert eine mögliche Architektur für ein (semi-)virtuelles Data Warehouse. Zunächst wird die Motivation für die Verwendung solch eines Ansatzes anhand einer Studie verdeutlicht und die erforderlichen Konzepte der Virtualisierung und Datenvirtualisierung geklärt. Anschließend wird die Architektur, deren Komponenten und die Zusammenhänge erläutert. Zum Schluss werden einige relevante Optimierungen vorgestellt, um den Nachteil der aus der Architektur resultierenden reduzierten Performanz zu minimieren.

### 2.3.1 Einführung Virtualisierung

Die Kernidee eines virtuellen Data Warehouses baut auf dem Konzept der Virtualisierung auf. Dieses wird in diesem Unterabschnitt definiert und anhand von einigen Beispielen erläutert. Außerdem wird die Motivation für die Einführung von virtuellen Data Warehouses ausgeführt.

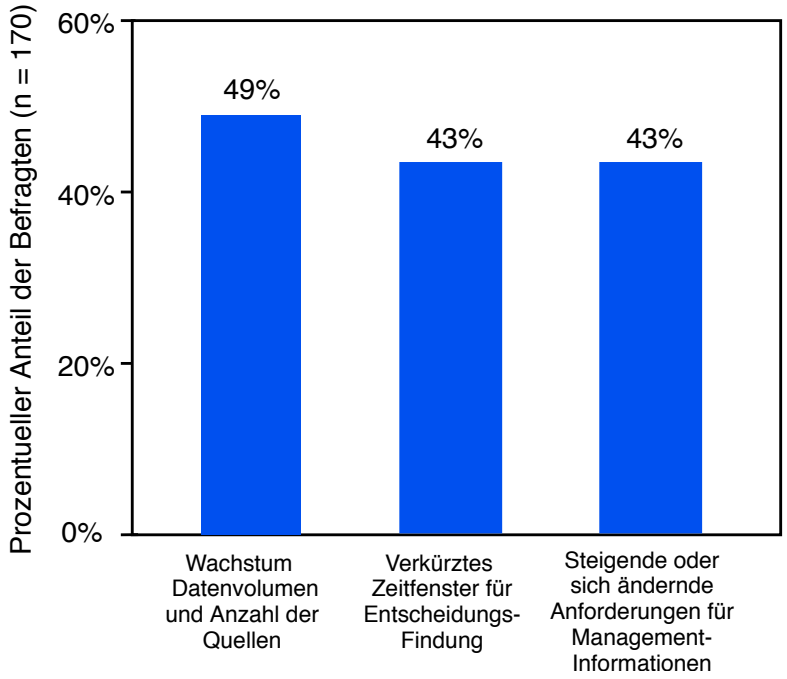



Abbildung 2.5: Studie der Aberdeen Gruppe. Teilnehmer berichten von verkürzten Zeitfenstern für Entscheidungen und sich ändernde Nachfragen an Management-Informationen

Quelle: [The Aberdeen Group, 2011]

**Motivation**

Seit der Veröffentlichung des ersten Buches über Data Warehouses von Bill Inmon im Jahre 1990 hat sich einiges getan, vor allem in der Welt der IT. Die Möglichkeiten durch Technologien wurden erweitert, gleichzeitig werden Business Intelligence Systeme vor neue Herausforderungen und Anforderungen gestellt. Aufgrund dessen wurden neue Ansätze entwickelt um sich diesen neuen Situationen zu stellen. Hierzu gehört unter anderem das virtuelle Data Warehouse.

BI Systeme generieren Nutzen indem sie das Management dabei unterstützen Entscheidungen zu treffen. Eine der neuen Anforderungen ergibt sich durch die zunehmende Schnellebigkeit in der heutigen Zeit. Die Umgebungen verändern sich, Unternehmen müssen schneller reagieren können und damit auch zügiger Entscheidungen treffen können [van der Lans, 2012]. Eine Studie der Aberdeen Group aus dem Jahre 2011 zeigte bereits, dass 43% der Unternehmen es schwerer finden rechtzeitig Entscheidungen zu treffen. Manager sagen, sie hätten immer weniger Zeit um auf Ereignisse auf dem Markt zu reagieren [The Aberdeen Group, 2011]. Eine mögliche Ursache hierfür wären Mega-Trends wie Mobile Geräte, Cloud- und Logistik-Plattformen und Open-Source Software die die Substitution von bisherigen Produkten und Dienstleistungen einfacher machen [van der Lans, 2012]. In jedem Fall ergeben sich hieraus Anforderungen an BI Systeme. Reports müssen schneller erstellt und abgeändert werden können. Hierfür ist eine flexible Gestaltung der darunter liegenden Daten und Datenstrukturen erforderlich.

Eine weitere Herausforderung die BI Systeme angehen müssen, sind sich mehrende Quellen in unterschiedlichen Formaten. Auch Datenquellen außerhalb eines Unternehmens (Webseiten, Soziale Medien, verschiedenste APIs) können wertvolle Informationen enthalten und für Analysen verwendet werden. Hierbei müssen die Daten nicht immer aus Datenbanken kommen, es können Beispielsweise auch JSON-Daten, Weblogs oder rohe Sensordaten angebunden werden. Hieraus ergibt sich das Business Intelligence Systeme bei der Anbindung von Datenquellen flexibel sein sollten [van der Lans, 2012].

Zusätzlich ergeben sich durch neue Technologien weitere Möglichkeiten bei der Verarbeitung von Daten. Es ist von Vorteil, wenn ein BI System die Integration von neuen Technologien mit weniger Aufwand ermöglicht [van der Lans, 2012]. Dies kann zum Beispiel durch Abstraktion erreicht werden. Beispiele für solche Technologien wären In-Memory Datenbanken oder Solid State Drives(SSDs), durch welche die Verarbeitungsgeschwindigkeit von Daten beschleunigt werden kann.

Zusammengefasst ergibt sich durch die Schnellebigkeit der Gesellschaft, durch die steigende Anzahl an Datenquellen und durch neue wertvolle Technologien ein erhöhter Anspruch an die Flexibilität eines Data Warehouses. Aufgrund der mehrschichtigen Architekturen von traditionellen Data Warehouses (nach Kimball und Inmon) ist diese Flexibilität nur bedingt gegeben. Es wird eine Architektur benötigt, die aus weniger Komponenten besteht. Je weniger persistierte Schichten vorhanden sind, desto simpler und flexibler ist die resultierende Architektur.

**Virtualisierung**

*Virtualisierung* ist kein neues Konzept in der IT, bereits im Jahre 1960 wurde der virtuelle Arbeitsspeicher eingeführt. Hierdurch wurde es möglich, mehr Arbeitsspeicher zu verwenden als physikalisch vorhanden ist. Mit Virtualisierung in der IT ist die Abkapselung von Ressourcen von ihren technischen Details gemeint um eine vereinfachte Verwendung zu gewährleisten [van der Lans, 2012].

Beispielsweise kann der Arbeitsspeicher, wie bereits erwähnt, virtualisiert werden. Dadurch kann dieser verwendet werden, ohne sich Gedanken darüber machen zu müssen, wo der Arbeitsspeicher liegt oder wie viel davon vorhanden ist. Wenn der physische Arbeitsspeicher aufgebraucht ist, kann die virtualisierte Schicht auf die

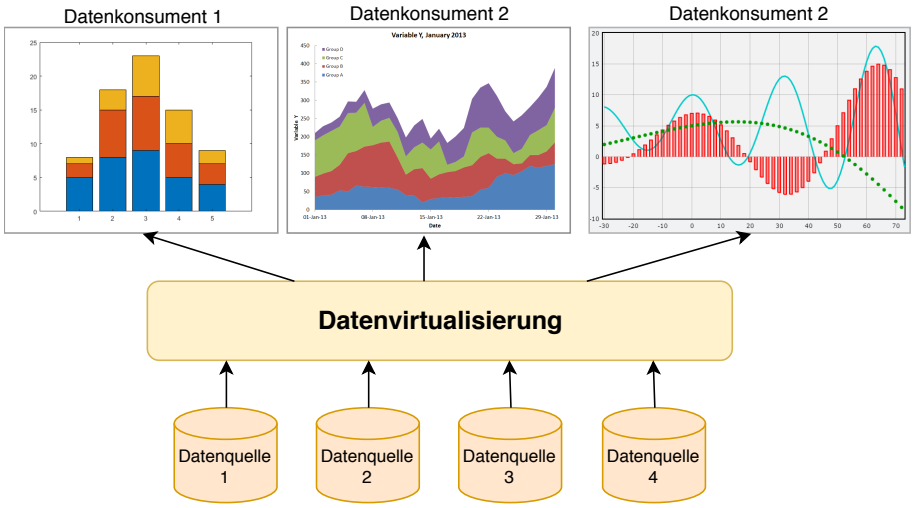



Abbildung 2.6: Virtualisierungsschicht zwischen Anwender und Datenquelle
Quelle: In Anlehnung an van der Lans [2012]

Festplatte zugreifen und diesen als Arbeitsspeicher anbieten. Ein anderes Beispiel wäre die Virtualisierung von Festplattenspeicher. Dadurch muss sich der Anwender keine Gedanken mehr um die einzelnen physikalischen Geräte machen und kann alle Dateien in einem Dateisystem verwalten. Gleichzeitig kann im Backend zum Beispiel eine automatische Replikation der Daten durchgeführt werden, um die Sicherheit vor Ausfällen zu erhöhen. Der Anwender muss hiervon gar nichts mitbekommen. Mittlerweile können sehr viele Komponenten in der IT virtualisiert werden [van der Lans, 2012, Wang und Ng, 2010]. Weitere Beispiele wären die Virtualisierung von Betriebssystemen, Netzwerkverbindungen und ganzen Rechenzentren.

*Datenvirtualisierung* ist eine Form der Virtualisierung, bei der wie der Name schon sagt, Daten abgekapselt werden. Hierbei wird eine Schicht zwischen Anwender und Datenquelle erzeugt. Diese Schicht verbirgt das Wissen darüber, wo und wie die Daten gespeichert werden [van der Lans, 2012]. Hierbei ergibt sich eine vereinfachte Verwendung für den Endanwender. Die Daten können durch eine einheitliche Schnittstelle erreicht werden. Informationen darüber wo die Daten wirklich liegen, welche Datenbanken verwendet werden und durch welche APIs der Zugriff gewährt wird, sind für die Verwendung nicht erforderlich. Das Konzept wird in Abbildung 2.6 dargestellt.

Dieses Konzept der Virtualisierung wird vom virtuellen Data Warehouse verwendet um die oben genannten Probleme zu überwinden und eine flexible Business Intelligence Lösung bereitzustellen.

### 2.3.2 Architektur (semi-)virtuelles Data Warehouse

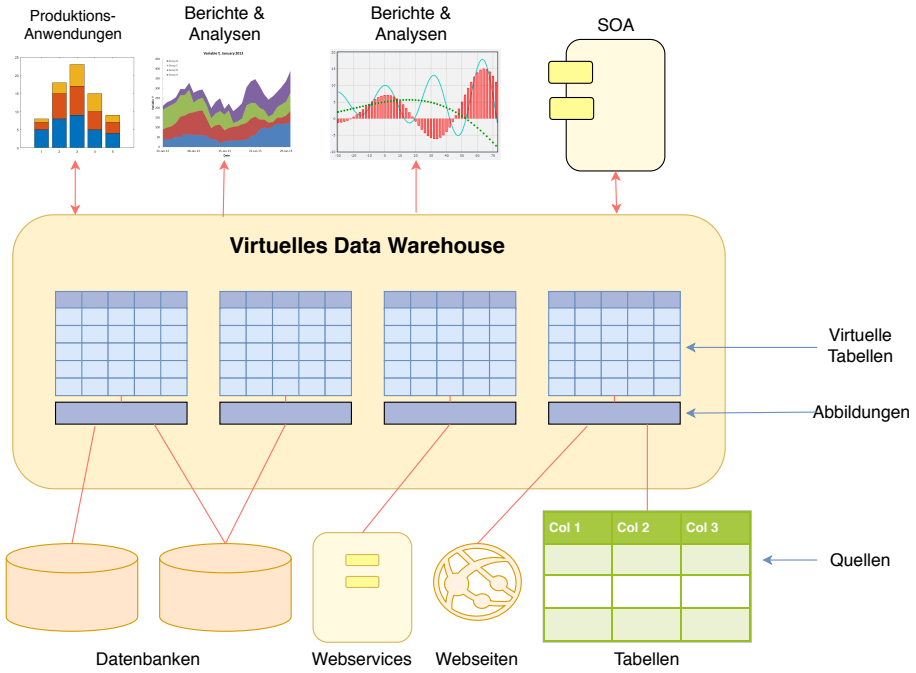



Abbildung 2.7: Grobe Architektur eines virtuellen Data Warehouses
Quelle: In Anlehnung an van der Lans [2012]

Nachfolgend wird eine mögliche Architektur für ein virtuelles Data Warehouse vorgestellt. Die vorhandenen Komponenten, ihre Funktionsweisen und Zusammenhänge werden erläutert. Durch den Grad der Virtualisierungen gibt es viele Variationen dieses Ansatzes. Im letzten Unterabschnitt wird das semi-virtuelle Data Warehouse vorgestellt, welches bis auf die Extraktionsphase alles Virtualisiert und in der Evaluierung verwendet wird.

#### 2.3.2.1 Überblick

Die Architektur eines virtuellen Data Warehouses nach van der Lans [2012] besteht aus drei Schichten wie in Abbildung 2.7 dargestellt: dem Anwender, dem virtuellen Data Warehouse und den Quellsystemen. Die Anwender nutzen eine einheitliche Schnittstelle um auf die Daten des Warehouses zuzugreifen. Es sieht von außen so aus, als würde dieses aus einer einzigen riesigen Datenbank bestehen. Es werden Anfragen an die virtuelle Schicht gesendet und Ergebnisse zurückgeliefert. Es ist nichts darüber bekannt, wo und wie die darunterliegenden Daten gespeichert sind. Dies ist das Ergebnis der bereits erwähnten *Datenvirtualisierung*.

Die virtuelle Schicht besteht aus einer Hierarchie von *virtuellen Tabellen*. Eine virtuelle Tabelle ist genau dasselbe wie eine relationale Tabelle, mit dem einzigen Unterschied, dass die Daten nicht direkt im Speicher vorliegen, sondern durch Abbildungen aus anderen Tabellen beschrieben werden. Wenn diese virtuelle Tabelle abgefragt wird, dann werden die Daten in Echtzeit neu berechnet [Gluchowski und Chamoni, 2016]. Für den Anwender sieht solch eine Tabelle wie eine ganz normale relationale Tabelle aus. *Caching* und *Query Optimization* sind weitere wichtige Bestandteile der virtuellen Schicht und dienen der Verbesserung der Abfragegeschwindigkeit.

Häufig ist eine einzige persistierte Schicht für historische Daten vorhanden, die genaue Ausprägung hiervon ist sehr variabel [van der Lans, 2012]. Auf die Elemente einer virtuellen Schicht wird bald nachfolgend ausführlicher eingegangen.

Die letzte Schicht besteht aus den Quellsystemen, die letztendlich die Daten für die Analysen liefern. Diese Schicht ist equivalent zu den Quellschichten der bereits erwähnten traditionellen Data Warehouses. Unterschiedlich ist der Zugriff auf diese Quellsysteme. Während traditionelle Data Warehouses meistens mit Batchverfahren periodisch neue Daten extrahieren, kann ein virtuelles Data Warehouse bei Bedarf direkt auf die Quellen zugreifen, um die aktuellsten Daten verwenden zu können. Hierbei wird der Datentransfer minimiert, indem Selektionen und Bedingungen in die Quellen geschoben und dort ausgeführt werden. Diese Art der Optimierung nennt man *Predicate Pushdown* [Köppen u. a., 2012, van der Lans, 2012]. Eine zusätzliche äußerst wichtige Optimierung ist das Caching, hierbei werden Zwischenergebnisse im Arbeitsspeicher oder auf der Festplatte für die Wiederverwendung gesichert. Dadurch werden die Quellsysteme vor hohen Belastungen geschützt. Weitere Details hierzu folgen im nächsten Unterabschnitt.

Es können verschiedene Schnittstellen für den Zugriff auf das Data Warehouse bereitgestellt werden. Einige Möglichkeiten wären JDBC/SQL, MDX oder SOAP [van der Lans, 2012]. Jeder Nutzer kann eine andere API verwenden, je nach Anwendungsfall können verschiedene Schnittstellen unterschiedlich praktisch sein. Im Hintergrund werden die selben Daten verwendet, nur die Repräsentation nach außen ist unterschiedlich.

#### 2.3.2.2 Quellen und Wrapper

Nachfolgend wird der Aufbau der virtuellen Schicht etwas ausführlicher erläutert, es wird hierbei von unten nach oben erklärt. Angefangen bei den Quellen bis zu der Schnittstelle zu den Anwendern.

Damit ein Quellsystem als Quelle für eine virtuelle Tabelle dienen kann, muss dieses *importiert* werden. Beim importieren verbindet sich das virtuelle Data Warehouse mit dem Quellsystem und extrahiert alle relevanten Metadaten [van der Lans, 2012]. Dabei wird unter anderem entschieden, ob Datentransformationen notwendig sind (z.B.: VARCHAR(20) zu STRING). Das Ergebnis dieses Vorgangs ist eine *Wrappertabelle*, welche alle Metadaten enthält, die notwendig sind, um Daten aus der Quelle zu extrahieren (Verbindung, Download, ...) und in ein standardisiertes Format zu bringen (einfache Transformationen) [van der Lans, 2012]. Diese Metadaten unterscheiden sich von Quelle zu Quelle, zum Beispiel würde eine relationale Datenbank andere Verbindungsdaten benötigen als eine REST-API. Der Zugriff auf eine Wrappertabelle unterscheidet sich nicht von dem Zugriff auf eine normale relationale Tabelle, von außen lassen sich diese nicht unterscheiden.

Wenn nun Beispielsweise eine relationale Datenbank als Quelle eingebunden werden soll, dann wird für jede benötigte Tabelle aus dieser Datenbank eine Wrappertabelle erzeugt. Beim Zugriff auf diese Wrappertabellen wird im Hintergrund die relationale Datenbank angefragt und die benötigten Daten geholt. Im Falle von Quellen, die keine relationale Datenbank sind, werden zusätzlich Informationen über Transformationen benötigt, um die Daten in ein Tabellenformat für die Wrappertabelle zu

überführen [van der Lans, 2012]. Die Metadaten einer Wrappertabelle können unter anderem folgendes beinhalten:

- Netzwerkadresse des Quellsystems
- Informationen für Datenbankverbindung
- Name, Besitzer und Erstellungsdatum der Quelltabelle
- Struktur der Quelltabelle (Spalten und Namen)
- Für jede Spalte: Datentyp und weitere Eigenschaften(NOT NULL,UNIQUE,...)
- Notwendige Datentyptransformationen (in das Schema des Data Warehouses)
- Anzahl der Zeilen und Werteverteilungen für jede Spalte (Queryoptimization)

#### 2.3.2.3 Virtuelle Tabellen und Abbildungen

Die Nutzer eines Data Warehouses wollen selten direkt auf die Daten aus den Quellen zugreifen. Meistens werden aggregierte, transformierte und zusammengeführte Daten aus mehreren Quellen benötigt. Hierfür werden virtuelle Tabellen benötigt, diese bauen auf den Wrappertabellen oder anderen virtuellen Tabellen auf und können so verschiedene Sichten auf die Daten erzeugen. Diese virtuellen Tabellen haben eine sehr hohe Ähnlichkeit zu Sichten aus Datenbanksystemen.

Die Art und Weise, wie die Quelle einer virtuellen Tabelle transformiert werden muss, wird *Abbildung* genannt. Diese Abbildung definiert den Aufbau der virtuellen Tabelle durch die Transformationen der Quellen. Um eine virtuelle Tabelle zu erzeugen, muss folglich eine Abbildung definiert werden [van der Lans, 2012]. Die Komplexität dieser Abbildungen kann hierbei stak variieren. Es können Daten aus verschiedenen Tabellen verbunden werden, Daten können aggregiert werden oder die Werte von bestimmten Spalten können transformiert werden. Die möglichen Operationen sind nicht auf die gewöhnlichen SQL-Operatoren beschränkt, die meisten virtuellen Data Warehouses unterstützen externe Funktionen. Hierdurch können beliebige Funktionen definiert werden, die anschließend in SQL-Queries verwendet werden können. Funktionen wie *Deduplication* oder *Fuzzy Joins* werden häufig von Haus aus angeboten und können in den Definitionen der Mappings für virtuelle Tabellen verwendet werden [van der Lans, 2012]. Durch einen hierarchischen Aufbau dieser Tabellen können sehr komplexe Transformationen und ganze ETL-Prozesse aufgebaut werden.

Der Unterschied zwischen Wrappern und virtuellen Tabellen ist, dass Wrapper nur auf Quellsysteme zeigen dürfen und diese nur durch simple Transformationen wie Typumwandlungen modifizieren dürfen [van der Lans, 2012]. Virtuelle Tabellen hingegen dürfen nur auf Wrapper und andere virtuelle Tabellen verweisen und beliebig komplexe Transformationen anwenden. Die bisher erwähnten Elemente (Quellsysteme, Abbildungen und virtuelle Tabellen) bilden die Bausteine eines virtuellen Data Warehouses und werden in Abbildung 2.8 zusammenfassend dargestellt.

Durch diesen Hierarchischen Aufbau von virtuellen Tabellen können Metadaten geteilt und wiederverwendet werden. Wenn Beispielsweise ein Teil der Transformationen von zwei Tabellen gleich ist, können diese identischen Transformationen in eine zusätzliche virtuelle Tabelle ausgelagert werden. Anschließend können die beiden bisherigen Tabellen auf diese neue Tabelle verweisen und die sich unterscheidenden Abbildungen ergänzen. Hierdurch werden Änderungen an der neuen Tabelle an beide darauf verweisenden Tabellen automatisch weitergegeben. Dies ist gewollt, da diese beiden Tabellen identische Anforderungen an die gemeinsamen Transformationen haben. Diese Möglichkeit Metadaten zu teilen ist sehr praktisch, vor allem bei umfangreichen aufeinander aufbauenden ETL-Prozessen steigt hierdurch die Entwicklungseffizienz und die Wartbarkeit des Gesamtsystems.

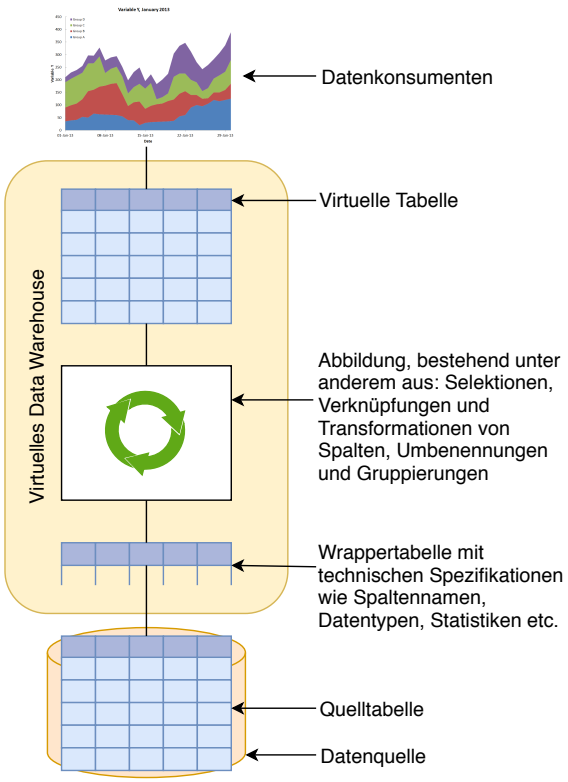



Abbildung 2.8: Zusammenhang aus Quelle, Wrappertabelle und virtueller Tabelle
Quelle: In Anlehnung an van der Lans [2012]

Nun fehlt nur noch der Zugriff des Endanwenders auf diese virtuellen Tabellen um Abfragen durchführen zu können. Um den Zugriff zu gewähren kann für jede Tabelle eine Menge von Schnittstellen gewählt werden, durch welche diese erreichbar sein soll. Diesen Schritt nennt man die *Veröffentlichung* einer virtuellen Tabelle [van der Lans, 2012]. Hierdurch kann eine Tabelle Beispielsweise von einem Nutzer durch JDBC/SQL[4] angesprochen werden und von jemand anderem durch REST[5]. Dadurch können verschiedene Anwendungen ein und dieselbe virtuelle Tabelle abfragen und hierbei ihre bevorzugten Schnittstellen verwenden. Der Vorteil hierbei liegt darin, dass auch Anwendungen, die ihren Fokus nicht auf Analysen haben, durch andere Schnittstellen auf das Data Warehouse zugreifen können [van der Lans, 2012].

#### 2.3.2.4 Semi-virtuelles Data Warehouse

Der Grad, in dem die ETL-Prozesse virtualisiert werden, ist variabel. Häufig wird zumindest eine persistierte Schicht für die Sicherung von historischen Daten angelegt [Gluchowski und Chamoni, 2016, van der Lans, 2012]. Je höher der Grad der Virtualisierung ist, desto ausgeprägter sind die Vor- und Nachteile dieses Ansatzes.

[4]https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/, abgerufen am 06.08.2018
[5]https://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer, abgerufen am 06.08.2018

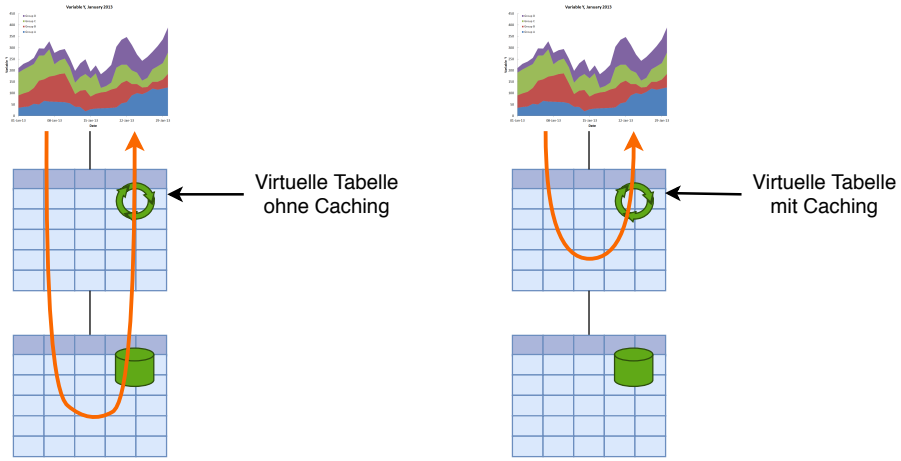



Abbildung 2.9: Query mit und ohne Caching
Quelle: In Anlehnung an van der Lans [2012]

Ein virtuelles Data Warehouse, bei dem bis auf die Ergebnisse der Extraktionsphase alles virtualisiert ist, wird nachfolgend als semi-virtuelles Data Warehouse bezeichnet und im praktischen Teil der Arbeit umgesetzt [Gluchowski und Chamoni, 2016]. Der Grund liegt darin, dass hierdurch historische Analysen möglich sind bei gleichzeitiger Maximierung des Virtualisierungsgrades. Auch bei diesem Ansatz ist es möglich virtuelle Tabellen zu definieren, die nicht auf historisierte lokalen Daten zeigen, sondern ihre Daten direkt aus den Quellsystemen extrahieren.

Die Umsetzung eines reinen virtuellen Data Warehouses, bei dem keinerlei Persistierung angewendet wird, ist weniger praktikabel, weil historische Analysen nicht sicher gewährleistet werden können. Die Aufgabe der historischen Datenhaltung muss hierbei auf die Quellen abgegeben werden, was sich in der Praxis als schwierig herausstellt [van der Lans, 2012].

### 2.3.3 Optimierung

Durch die virtuelle Schicht ergeben sich auch Nachteile, der größte hiervon ist eine Reduzierung der Abfragegeschwindigkeit. Durch das spontane Abfragen von Daten und die spontane Berechnung der Ergebnisse und Zwischenergebnisse ergibt sich ein Overhead bei den Anfragen an die virtuellen Tabellen. Um diesen Overhead zu reduzieren werden in der Praxis verschiedene Techniken verwendet, diese lassen sich in folgende Kategorien einteilen [van der Lans, 2012]:

- Caching
- Query Optimization

Eines der wichtigsten Werkzeuge zur Reduzierung der Belastung der Quellsysteme und zur Optimierung der Abfragegeschwindigkeit ist das Caching [van der Lans, 2012]. Beim Caching werden Daten im Arbeitsspeicher oder auf der Festplatte für einen schnelleren Zugriff zwischengespeichert. Im Falle des virtuellen Data Warehouses werden virtuelle Tabellen gecached, diese nennt man dann *materialized virtual*

*tables*. Beim Zugriff auf diese Tabellen müssen keine Quellen abgefragt und keine Berechnungen durchgeführt werden, es werden einfach die zwischengespeicherten Daten wiedergegeben. Ein Beispiel hierfür ist in Abbildung Abbildung 2.9 zu sehen. Im Falle eines virtuellen Data Warehouses existieren viele virtuelle Tabellen, die hierarchisch aufgebaut sind. Wenn eine Tabelle materialisiert wird verwenden alle darauf verweisenden Tabellen automatisch den Cache.

Ein Nachteil hiervon ist, dass die Daten nicht auf dem aktuellsten Stand sind. In den meisten Fällen wird eine Periode für jede dieser Tabelle angegeben, innerhalb derer der Cache automatisch aktualisiert wird. Wenn bei einer Abfrage unbedingt die aktuellsten Daten benötigt werden, dann kann das Caching auch umgangen werden und die Daten neu berechnet werden.

Query Optimization ist eine andere Herangehensweise um die Effizienz eines virtuellen Data Warehouses zu verbessern. Dadurch das deklarative Sprachen wie SQL und MDX nur spezifizieren welche Daten benötigt werden und nicht wie diese beschafft werden sollen, ergibt sich hier eine hohe Vielfalt an Möglichkeiten [van der Lans, 2012]. Der *Optimizer* ist dafür zuständig eine optimale Strategie für diese Aufgabe zu finden. Hierbei helfen Informationen über die jeweiligen Tabellen wie Zeilenanzahl, Verteilung der Werte in den Spalten und ob Daten gecached sind oder nicht. Zu den möglichen Optimierungstechniken gehören unter anderem Query Substitution, Query Predicate Pushdown und verteilte Joins [van der Lans, 2012].

Predicate Pushdown ist vor allem im Falle eines virtuellen Data Warehouses eine sehr wertvolle Optimierungstechnik. Aus Platzgründen wird an dieser Stelle nur auf diese Technik kurz eingegangen. Einer der aufwendigsten Teilschritte beim Verarbeiten von Anfragen ist das Beschaffen der Daten [van der Lans, 2012]. Dieser Schritt kann vor allem beim reinen virtuellen Data Warehouse durch die externen Quellen für einen hohen Netzwerktransfer sorgen. Um diese Problem zu minimieren, werden so viele Teile der Abfrage wie möglich nach unten zu den Quellen geschoben. Wenn eine Anfrage zum Beispiel nur Daten aus dem Jahr 2017 benötigt, dann kann diese Bedingung in die Quellen geschoben werden, sodass nur die wirklich benötigten Daten zurückgegeben werden. Das reduziert den Datentransfer bei der Beschaffung und den Rechenaufwand.

### 2.3.4 Abgrenzung zu traditionellem Data Warehouse

Nachfolgend werden die Gemeinsamkeiten, Unterschiede und die daraus resultierenden Vor- und Nachteile zwischen traditionellen und virtuellen Ansätzen zusammengefasst.

**Gemeinsamkeiten**
Bei allen Ansätzen handelt es sich im Kern um ein Data Warehouse, sie müssen somit alle den selben Zweck erfüllen und sich ähnlichen Anforderungen stellen. Sie müssen für den Endanwender eine einheitliche Schnittstelle auf bereinigte, standardisierte und integrierte Daten gewähren. Diese Schnittstelle muss für den Endanwender leicht verständlich sein und konsistente Ergebnisse liefern, um das Management bei der Entscheidungsfindung unterstützen zu können [van der Lans, 2012]. Gleichzeitig müssen Analysen auf historischen Daten möglich sein, die Quellsysteme dürfen

nicht überlastet werden und die gesamte Architektur muss anpassbar bleiben [Köppen u. a., 2012].

**Unterschiede und Konsequenzen**
Der zentrale Unterschied zwischen einem virtuellen und einem traditionellen Ansatz ist die Art und Weise, wie die ETL-Prozesse umgesetzt werden. Während bei traditionellen Ansätzen die Ergebnisse dieser Prozesse für eine effiziente Abfrage redundant gesichert werden, führen virtuelle Ansätze diese Prozesse mehr oder weniger erst bei Abfrage der Daten durch. Die Vorteile eines virtuellen Ansatzes ergeben sich durch eine meist simplere Architektur und reduzierte Datenredundanzen [van der Lans, 2012]:

- Schnellere und günstigere Entwicklung durch eine simplere Architektur
- Erhöhte Flexibilität
- Datenkonsistenz kann einfacher gewährleistet werden

Der sich durch die Architektur resultierende entscheidende Nachteil ist eine geringere Leistung bei der Verarbeitung von analytischen Anfragen. Da dies ein wichtiger Faktor für den Erfolg eines Data Warehouses ist, muss sichergestellt werden, dass diese Anforderungen erfüllt werden können. Durch eine Variation des Virtualisierungsgrades können die Vor- und Nachteile abgewägt werden.

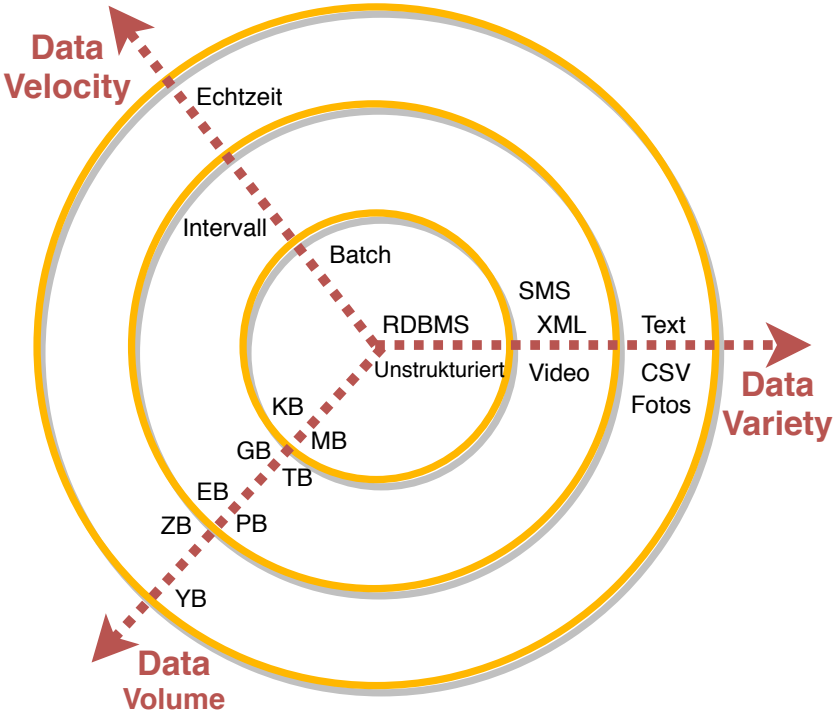



Abbildung 2.10: Die drei Dimensionen entlang derer Big Data definiert werden kann
Quelle: Angelehnt an "Big Data – What is Big Data" von Pinal Dave[6]

## 2.4 Big Data

In dem folgenden Abschnitt wird zunächst geklärt was mit Big Data gemeint ist, welche Anforderungen an Big Data Systeme gestellt werden und wie dieser Begriff mit Data Warehouses in Relation steht. Anschließend wird auf das Software Framework Hadoop eingegangen, welches entwickelt wurde um Big Data Infrastrukturen zu entwickeln. Es wird geklärt inwiefern es möglich ist ein Data Warehouse mit Hadoop umzusetzen und wieso die Entscheidung getroffen wurde, dies im praktischen Teil der Arbeit zu tun. Zum Schluss werden die wichtigsten verwendeten Hadoop-Komponenten beschrieben und geklärt, welche Komponente für die SQL-Query-Engine verwendet wurde und wieso.

### 2.4.1 Definition und Überschneidung mit Data Warehouses

Der Begriff Big Data ist mittlerweile weit verbreitet und allgegenwärtig. Er stammt zu einigen Teilen aus der Wissenschaft, der Industrie und den Medien. Deswegen gibt es bisher keine allgemeingültige und einheitliche Definition [Ward und Barker, 2013]. Nachfolgend wird die sehr weit verbreitete Definition aus dem Gartner-Report verwendet [Laney, 2001].

Diese Definition besteht aus drei Teilen, welche die Eigenschaften von Datenmengen beschreiben, die in die Definition passen. Diese drei Dimensionen werden in Abbildung 2.10 zusammenfassend dargestellt. Die Dimension "Volume" beschreibt den Umfang der Datenmengen, "Velocity" beschreibt die Geschwindigkeit, in der

[6]https://blog.sqlauthority.com, Abgerufen: 9.10.2018

neue Daten generiert werden. Und "Variety" beschreibt die Menge an Formaten, durch welche die Daten repräsentiert werden. Je höher das "Volume", die "Velocity" und die "Variety" für eine Datenmenge sind, desto eher kann die Datenmenge in die Kategorie Big Data eingeordnet werden. Es wurden keine konkreten Messgrößen für diese Definition spezifiziert [Ward und Barker, 2013]. In einigen weiteren Definitionen spielen die verwendeten Technologien eine Rolle. Zum Beispiel besagt die Definition nach NIST, dass Big Data die Kapazitäten von herkömmlichen Methoden und Systemen überschreitet [NIST]. Es werden somit neue Methoden und Technologien benötigt, hierzu gehören zum Beispiel NoSQL[7] und MapReduce [Dean und Ghemawat, 2004]. Diese Methoden und Technologien müssen somit gewisse Anforderungen erfüllen. Sie müssen in der Lage sein, mit Datenmengen umzugehen, die ein hohes Volumen, eine hohe Velocity und Variety besitzen.

Data Warehouses sind nun eine Art von Systemen, auf die die Definition von Gartner äußerst gut zutrifft. Ihre Kernkompetenz besteht darin Datenmengen zu verarbeiten die ein sehr hohes Volumen aufweisen, kontinuierlich wachsen und deren Repräsentation variieren kann. Die Dimension "Variety" nimmt durch immer mehr Möglichkeiten verschiedene Quellsysteme anzubinden auch zu. Sie müssen somit in der Lage sein mit diesen Anforderungen umzugehen. Hierbei spielt es keine Rolle, um was für eine Art von Data Warehouse es sich handelt. Sowohl konventionelle Data Warehouse mit einer umfangreichen ETL-Schicht, als auch modernere virtuelle Data Warehouses müssen sich diesem Problem stellen. Auch wenn das virtuelle Data Warehouse eine Menge an zusätzlichen Vorteilen verspricht, bringt dies alles nichts, wenn es nicht in der Lage ist Analysen auf den Datenmengen durchzuführen. Das Hauptziel dieser Arbeit ist es somit die Effizienz und Skalierbarkeit des virtuellen Data Warehouses bei steigendem Volumen zu messen und mit einem traditionellen Ansatz zu vergleichen.

### 2.4.2 Hadoop

Das Software Framework Hadoop stellt eine Art Ökosystem bereit, welches für verteilt arbeitende und skalierende Systeme vorgesehen ist. Es beinhaltet viele verschiedene Technologien welche unterschiedliche Probleme im Bereich Big Data lösen. Hadoop kann auf unterschiedlicher Hardware betrieben werden und gehört zu den wichtigsten Apache-Projekten[8]. Die folgenden Module sind inbegriffen:

- *Hadoop Common*

  Stellt Grundfunktionen und Tools für die weiteren Bausteine bereit

- *Hadoop Distributed File System (HDFS)*

  Ein verteiltes Dateisystem welches einen hohen Datendurchsatz und eine hohe Verlässlichkeit gewährt

- *YARN*

  Verwaltet die Ressourcen und die zeitliche Planung der Aufgaben

[7] Definition: https://www.searchenterprisesoftware.de/definition/NoSQL
[8] http://hadoop.apache.org/

- *MapReduce*

  Ein auf YARN aufbauendes Modul, welches die parallele Verarbeitung von großen Datenmengen ermöglicht

**Data Warehouses und Hadoop**

In klassischen Data Warehouse Systemen wurde Hadoop, wie bereits erwähnt, höchstens als Unterstützung verwendet. Für sehr aufwendige Aufgaben, bei denen eine geringe Latenz keine Rolle spielt, ist es eine kostengünstige und effiziente Lösung. Zum Beispiel kann es verwendet werden, um ETL-Prozesse durchzuführen. Mittlerweile gibt es auch Projekte, die eine effiziente Verarbeitung von Daten mit geringer Latenz versprechen. Hierzu gehören unter anderem Apache Impala[9], Apache Drill[10] und Apache Spark[11]. Sie alle basieren nicht auf MapReduce [Dean und Ghemawat, 2004], sondern verwenden ihre eigenen Methoden um eine geringe Latenz zu erreichen. Apache Impala und Drill basieren zum Beispiel auf Dremel. Dremel ist ein System, welches durch Ausführungsbäume und eine spaltenorientierte Datenspeicherung eine effiziente Ausführung von Analysen auf geschachtelten Daten erlaubt. Dieser Ansatz wurde von Google in einem Paper veröffentlicht [Melnik u. a., 2010]. Zusätzlich stellen die oben genannten Projekte eine SQL-Query-Engine bereit, mit der Analysen in der sehr verbreiteten Datenbanksprache SQL definiert werden können.

Klassischerweise befindet sich im Kern eines Data Warehouses eine relationale Datenbank oder ein multidimensionales Array. Diese sind für die effiziente Verarbeitung von analytischen Anfragen verantwortlich. Dieser Kern kann nun mehr oder weniger durch eine auf Hadoop basierende SQL-Query-Engine mit einer geringen Latenz umgesetzt werden [Grover u. a., 2015, Kornacker u. a., 2015]. Dies ist einer der Gründe, weshalb die Entscheidung getroffen wurde, die Data Warehouses mittels Hadoop umzusetzen. Dafür sprachen die zum einen frei verfügbaren Projekte, welche die gestellten Anforderungen an das Data Warehouse erfüllen können. Zusätzlich konnten alle Komponenten des Data Warehouses innerhalb von Hadoop umgesetzt werden, wodurch es einfacher war, die Arbeit im Zeitrahmen abzuschließen. Sowohl die ETL-Prozesse, die Daten im Sternschema, die SQL-Query-Engine, als auch eine Schnittstelle für Business Intelligence-Anwendungen konnten in einem Framework realisiert werden. Die verwendeten Technologien werden im folgenden Abschnitt genauer vorgestellt.

[9] https://impala.apache.org/
[10] https://drill.apache.org/
[11] http://spark.apache.org/

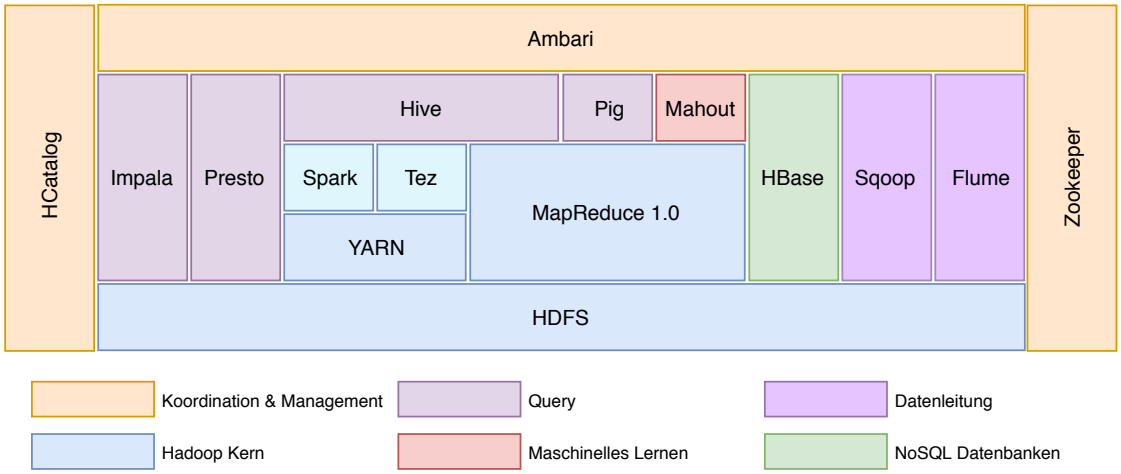



Abbildung 2.11: Ein Überblick über das Hadoop Ökosystem
Quelle: Angelehnt an "Overview of the Hadoop Ecosystem" von Dayong Du[12]

### 2.4.3 Verwendete Hadoop-Projekte

In dem folgenden Unterabschnitt wird ein Teil der im praktischen Teil verwendeten Hadoop-Komponenten kurz erläutert. Abbildung 2.11 stellt diese übersichtlich dar. Die Komponenten bauen teilweise von unten nach oben aufeinander auf, in dieser Reihenfolge werden sie nachfolgend auch erläutert.

**HDFS**
Das Hadoop Distributed Filesystem bildet das Fundament von Hadoop. Es ist ein verteiltes Dateisystem welches einen hohen Datendurchsatz und hohe Datensicherheit bereitstellt. Die im HDFS gespeicherten Dateien werden automatisch auf die einzelnen Knoten im Cluster verteilt, für den Anwender sieht dies wie ein normales Dateisystem aus.

Im Kern befindet sich eine Master-Slave-Architektur welche durch einen *NameNode* und ein oder mehrere *DataNodes* repräsentiert wird. Der NameNode übernimmt hier eine Managerrolle. Er weiß zu jeder Datei in welche Blöcke diese aufgeteilt wurde, auf welchen Cluster-Knoten diese liegen, und welche Kopien der einzelnen Blöcke es gibt. Die DataNodes befinden sich auf jedem Cluster-Knoten, auf dem HDFS-Dateien abgelegt werden sollen. Diese sind dafür da, die ihnen zugewiesenen Blöcke auf der Festplatte zu sichern. Sie haben keine Informationen darüber welche Blöcke zu welchen Dateien gehören. Wenn nun ein Anwender eine Datei lesen möchte, dann sendet er eine Anfrage an den NameNode. Dieser gibt die Informationen darüber wo sich diese Datei befindet an den Anwender wieder. Nun kann der Anwender eine direkte Verbindung zu den relevanten DataNodes aufnehmen und die Daten lesen. Hierdurch soll eine Überlastung des NameNodes verhindert werden.

Zusätzlich wird eine Ausfallsicherheit durch eine Replikation der Blöcke geboten. Jeder Block wird automatisch um einen spezifizierbaren Replikationsfaktor mehrfach auf dem Cluster gesichert. Da jegliche Kommunikation über den NameNode initiiert wird, stellt dieser einen "Single Point of Failure" dar. Um Korruption und Fehler

[12]Dayong Du: "Overview of the Hadoop Ecosystem", https://www.oreilly.com/library/view/apache-hive-essentials/9781783558575/ch01s05.html, Abgerufen: 9.10.2018

zu vermeiden wird diesem ein *SecondaryNameNode* beigestellt. Er protokolliert alle Änderungen im EditLog.

**YARN**
*Yet Another Resource Negotiator* ist ein weiteres zentrales Modul in Hadoop. Es ist für das dynamische verwalten von Ressourcen (CPU, RAM, GPU und FPGA) und die zeitliche Verteilung und Planung von Aufgaben verantwortlich. Es weist somit den Anwendungen die benötigten Systemressourcen zu.

Durch YARN wird MapReduce von der Ressourcenverwaltung abgekoppelt. Hierdurch ist es möglich, unterschiedliche Verarbeitungs- und Prozessarten auszuführen. Zum Beispiel können nun parallel zu MapReduce-Batchjobs auch Echtzeit-Analysen und interaktive Abfragen durch andere Query-Engines ausgeführt werden.

**MapReduce**
MapReduce bildet die zentrale Verarbeitungskomponente innerhalb von Hadoop. Es ist ein Batchverfahren, welches ein paralleles Verarbeiten von Daten auf vielen Rechnern gleichzeitig ermöglicht. Zusätzlich ist es fehlertolerant. Bei einer fehlgeschlagenen Berechnung wird diese auf einem anderen Rechner neugestartet. Zunächst war innerhalb von Hadoop nur die Verarbeitung von Daten mittels MapReduce möglich. Wie bereits erwähnt, wurde diese Einschränkung durch YARN aufgehoben.

Im Kern von MapReduce befinden sich die Map- und Reducefunktionen. Ein Problem wird in viele kleine parallelisierbar Teile aufgespaltet, diese werden von den Mapfunktionen verteilt verarbeitet. Anschließend werden die Ergebnisse mittels der Reducefunktionen aggregiert.

Der Hauptvorteil dieses Ansatzes ist die sehr hohe Skalierbarkeit und gleichzeitige Fehlertoleranz, wodurch eine günstige parallele Verarbeitung auf handelsüblichen Rechnern ermöglicht wird. Ein entscheidender Nachteil ist die sehr hohe Latenz bei der Ausführung.

**Hive**
Apache Hive ist eine Erweiterung von Hadoop, durch welche die Definition von MapReduce-Jobs durch eine SQL-ähnliche Sprache ermöglicht wird. Es können somit sowohl strukturierte als auch unstrukturierte Daten mit SQL abgefragt werden. Durch die weite Verbreitung dieser Sprache wird das System universeller einsetzbar. Intern werden die SQL-Statements zu MapReduce-Jobs transformiert und ausgeführt. Im praktischen Teil der Arbeit wurde Hive für die ETL-Prozesse des traditionellen Ansatzes verwendet.

**Impala**
Apache Impala ist eine verteilte SQL-Query-Engine welche in das Hadoop Ökosystem integriert ist. Es kombiniert einige der Vorteile von traditionellen Datenbanksystemen wie SQL und Multi-User Support mit der Flexibilität und Skalierbarkeit von Hadoop. Es erlaubt das Definieren und Ausführen von SQL-Anfragen auf Daten im HDFS und anderen Speicherformaten. Beim Design wurden bewusst Ähnlichkeiten und Parallelen zu Hive geschaffen. Impala verwendet dieselben Metadaten (*Hive Metastore*) und dieselbe SQL-Syntax, auch die ODBC Treiber und das User Interface innerhalb von *Hue* (Web Interface für Hadoop) sind fast identisch. Zusätzlich können Impala Anfragen auf durch Hive erstellten Tabellen ausgeführt werden. Es ist somit

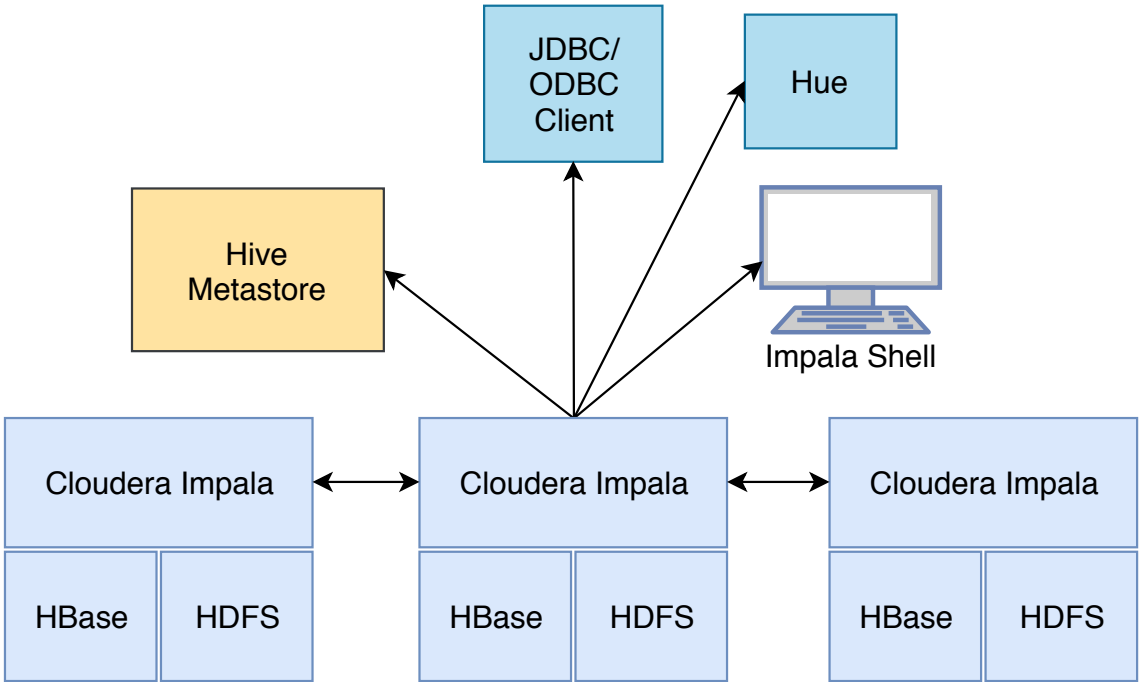



Abbildung 2.12: Ein Überblick über die Architektur von Impala
Quelle: "Apache Impala Overview"[13]

einfach zwischen Impala und Hive zu wechseln. Apache Impala wurde nicht dazu entworfen MapReduce zu ersetzen, vielmehr ist es eine Ergänzung um analytische Anfragen effizient und mit geringer Latenz ausführen zu können.

Eine sehr übersichtliche Darstellung der Architektur und der Schnittstelle zu den restlichen Hadoop Komponenten wird in Abbildung 2.12 präsentiert. Die folgenden Kernkomponenten existieren:

- *Clients*

  Die Anwender von Impala können auf verschiedene Wege mit Impala interagieren. Zur Auswahl steht eine interaktive Shell, eine Web-UI über Hue und eine generelle JDBC / ODBC Schnittstelle bereit. Durch die Schnittstellen können auch Analyse- und Visualisierungstools wie Tableau an Impala angebunden werden.

- *Hive Metastore*

  Der Hive Metastore wird von Hive bereitgestellt und fasst alle Meta-Informationen über vorhandene Tabellen, deren Aufbau und weiteres zusammen. Impala liest und aktualisiert diesen Metastore und kann hierdurch Problemlos Hive-Tabellen mitverwenden.

- *Impala*

  Die Impala-Daemons bilden den Kern von Impala und sitzen auf allen Data-Nodes. Jedem Knoten mit HDFS-Daten wird ein Impala-Daemon zugewiese und jeder dieser Daemons kann als Koordinator fungieren und SQL-Anfragen

entgegennehmen. Es wird in diesem Fall die Anfrage analysiert und ein optimierter Ablaufplan erstellt, anschließend werden die anfallenden Arbeiten auf alle Daemons verteilt. Diese greifen auf die Daten im HDFS zu und erledigen ihre jeweiligen Aufgaben. Die Ergebnisse werden anschließend zu dem Koordinator-Daemon zurückgegeben.

Impala bietet somit eine parallelisierte SQL-Query-Engine, welche mit sehr hohen Datenmengen umgehen kann und eine geringe Latenz aufweist. Zusätzlich existiert eine sehr gute Integration mit anderen Hadoop-Projekten. Bei bestimmten Auslastungen kann Impala bereits traditionelle analytische RDBMS ersetzen [Kornacker u. a., 2015]. Ein genauerer Einblick in die Funktionsweise und die Performanz von Impala verglichen mit anderen Hadoop- und traditionellen SQL-Query-Engines wird von Kornacker u. a. [2015] und Floratou u. a. [2014] gegeben.

**Verwendete Query-Engine**
Aufgrund der soeben erwähnten Eigenschaften wurde die Entscheidung getroffen, Impala als SQL-Query-Engine für das traditionelle und virtuelle Data Warehouse im praktischen Teil dieser Arbeit zu verwenden. Der Hauptgrund lag in der vergleichsweise sehr geringen Latenz. Alternativen wären unter anderem Apache Hive, SparkSQL und Apache Drill.

[13] "Apache Impala Overview", https://www.cloudera.com/documentation/enterprise/5-15-x/topics/impala_intro.html, Abgerufen: 9.10.2018

# 3. Implementierung

Es wurden zwei Data Warehouses für die finale Evaluierung umgesetzt, ein traditionelles und ein semi-virtuelles Data Warehouse. In diesem Kapitel werden die Implementierungen, die Designentscheidungen und die hieraus resultierenden Konsequenzen erläutert. Auch Maßnahmen, die getroffen wurden um die Vergleichbarkeit sicherzustellen, werden geklärt. Aufgrund des Umfangs der Implementierungen können diese nur auf einem abstrakten Level vorgestellt werden. Details, wie das aufsetzen und konfigurieren des Hadoop-Clusters, werden ausgelassen. Das Kapitel ist folgendermaßen strukturiert:

Zunächst wird die Umsetzung der Quellsysteme beschrieben. Diese bilden das Fundament, auf das sich beide Implementierungen stützen. Es wird auf die Generierung der Datenmengen und deren Verteilung auf die Quellsysteme eingegangen.

Anschließend wird erläutert, welche Maßnahmen getroffen wurden um eine Vergleichbarkeit zwischen den beiden Systemen zu sichern. Diese Maßnahmen und die Grenzen hiervon werden beschrieben.

Der dritte Abschnitt befasst sich mit der Implementierung des traditionellen Data Warehouses. Es wird auf die durchgeführten ETL-Prozesse, die angewendeten Optimierungen und die Art und Weise, auf die die Abfragen definiert werden können, eingegangen.

Im letzten Abschnitt wird die Architektur des semi-virtuellen Data Warehouses vorgestellt. Auf einem abstrakten Level wird der Aufbau der Virtuellen Tabellen und der Wrappertabellen beschrieben, der detaillierte Aufbau wird anhand von einigen konkreten Beispielen demonstriert. Zum Schluss wird wie beim traditionellen Ansatz auf die angewendeten Optimierungen und die Abfragen eingegangen.

## 3.1 Quellen

Die Quellen liegen außerhalb der beiden Data Warehouse Architekturen, in der Praxis sind sie aber immer ein fester Bestandteil von Data Warehouses. Die in dem folgenden Abschnitt vorgestellten Quellen wurden einmal implementiert und anschließend von beiden Data Warehouses verwendet. Für die Erzeugung der Quellen wurde zunächst ein auf den Use-Case zugeschnittenes Datenset generiert, welches anschließend auf eine dynamische Menge von Quellsystemen verteilt werden konnte. Diese Quellen wurde in der Evaluierung als Stellschraube verwendet um die Menge der zu verarbeitenden Datenmengen zu regulieren. So konnte die Skalierbarkeit der Implementierungen untersucht werden.

### 3.1.1 Quelldatengenerierung

Wie bereits in der Einführung erwähnt, wurde die Evaluierung auf einem Datensatz durchgeführt, der auf den Verkaufsdaten eines Getränkeeinzelhandels beruht. In diesem Kapitel wird der Aufbau und die Generierung dieses Datensatzes erläutert. Bei der Datengenerierung wurden Maßnahmen getroffen um die Daten möglichst realistisch zu erzeugen, damit bei der Evaluierung die Abfragen auf realistischen Produkten und Kunden beruhen und nicht auf zufälligen Zeichenabfolgen. Die Verkaufsdaten beruhen auf dem Dunnhumby-Datensatz[1] für Retail-Sales, dies ist ein generierter Datensatz der Muster aus dem echten Einzelhandel simuliert. Die selbst generierten Daten und der Datensatz von Dunnhumby wurden kombiniert und in ein Datenbankschema transformiert, welches anschließend in operative Datenbanken geladen werden konnte. Diese operativen Datenbanken repräsentieren die Quellen für das Data Warehouse. Um sich auf den Aufbau des Data Warehouses konzentrieren zu können, den Aufwand für den Aufbau der Quellsysteme zu reduzieren und die resultierenden Ergebnisse vergleichbarer zu halten, wurde für alle Quellsysteme dasselbe Datenbankschema verwendet. Weitere zu berücksichtigende Faktoren waren die Datengrößen der Datensätze und die Anzahl der Quellen. Es wurde die Entscheidung getroffen, einen großen Datensatz (100GB) zu erzeugen, aus dem im Nachhinein kleinere Datensätze und Quellen extrahiert werden konnten. Hierdurch war es leichter sicherzustellen, dass die Daten bei allen Tests exakt die selben sind.

**Dunnhumby**
Das Fundament für die generierten Daten bildete der Datensatz von Dunnhumby. Dieser Datensatz simuliert Transaktionen aus dem Einzelhandel und umfasst eine Gesamtmenge von 40,7GB an Daten. Diese Transaktionen sind über 117 Wochen verteilt mit 2,6 Millionen Transaktionen pro Woche und insgesamt 47 Millionen Warenkörben. Das Datenset wird durch 117 CSV Dateien ausgeliefert die jeweils eine Woche an Transaktionen enthalten. Für jede Transaktion existieren Fremdschlüssel für die Produkte / Kunden / etc. Diese können aber nicht verwendet werden, weil keine Kunden- oder Produktdaten mitgeliefert werden. Aus diesem und weiteren Gründen mussten die Daten transformiert werden um verwendet werden zu können.

[1] https://www.dunnhumby.com/sourcefiles

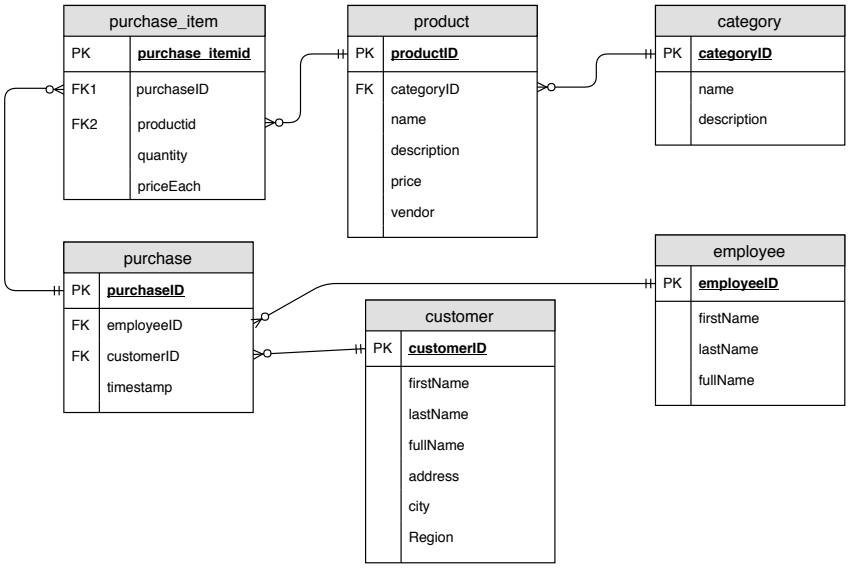



Abbildung 3.1: Datenbankschema der Quellsysteme
Quelle: Eigene Darstellung

**Weitere Tabellen und Datenbankschema**
Allein die Transaktionsdaten reichen für die Fragestellung nicht aus, es werden noch Informationen über Produkte, Kunden und den Standort der Filialen benötigt. Diese Daten wurden selbst generiert. Die Produkt und Filialdaten wurden per Hand generiert und in eine .csv Tabelle transformiert. Für die Kundendaten wurden Datensätze für insgesamt ∼500.000 Kunden erzeugt, diese passten mit den Kunden-IDs aus dem Dunnhumby Datensatz zusammen und konnten so kombiniert werden. Um realistisch wirkende Daten (Namen, Adressen, etc.) zu erzeugen, wurde eine Random User Generator[2] API verwendet. Diese API generiert Kundendaten bei Berücksichtigung von bestimmten Parametern und liefert diese im JSON-Format zurück. Die geladenen JSON-Kundendaten wurden anschließend bereinigt und in eine .csv Tabelle mit zu Dunnhumby passenden ID's transformiert.

Bei der Weiterverarbeitung des Dunnhumby Datensatzes mussten weitere Transformationen angewendet werden. Unter anderem mussten die ID's für Produkte, Kunden und Warenkörbe angepasst werden, Datentyptransformationen und Konvertierungen für die Zeiträume durchgeführt werden. Schlussendlich mussten die Daten in kleinere Tabellen aufgeteilt werden, um dem Datenbankschemata zu entsprechen. Dieses Datenbankschemata ist in Abbildung 3.1 dargestellt und wurde anhand des Use-Cases definiert. Am Ende aller Transformationen lagen die Daten in diesem Schema vor, nur die Employee Tabelle wurde bei den nachfolgenden Arbeiten nicht berücksichtigt. Der Umfang des Datensatzes betrug nach sämtlichen Transformationen noch ∼2,5GB. Um auf die gewünschten 100GB zu kommen, musste dieses also noch erweitert werden.

[2]https://randomuser.me

**Erweiterung des Datensatzes**
Die Erweiterung der Daten wurde in zwei Stufen durchgeführt, bei denen die Daten zunächst verzehnfacht und anschließend vervierfacht wurden (2,5GB * 10 * 4 = 100GB):

1. *Verzehnfachung der Daten durch Erweiterung der Warenkörbe:*

   In der ersten Stufe wurde die Anzahl der Transaktionen für jeden Warenkorb um das zehnfache erhöht. Das heißt, wenn ein Kunde in seinem Warenkorb zwei Gegenstände hatte, wurde dieser Warenkorb auf zwanzig Gegenstände erhöht. Durch diesen Ansatz konnte sichergestellt werden, dass die Verteilung der Transaktionen über alle Dimensionen (Produkt, Zeit und Kunde) hinweg gleich blieb. Ein Nebeneffekt ist, dass es keine Warenkörbe mit weniger als zehn Gegenständen gibt. Dies ist im weiteren Verlauf aber nicht relevant.

2. *Vervierfachung durch Erweiterung des Zeitraums* (117 Wochen → 468 Wochen)

   Anschließend wurden die Daten um das vierfache erweitert, indem der komplette bisherige Datensatz für neue Zeiträume repliziert wurde. Die Verteilung der Daten erstreckte sich auf 117 Wochen, durch die Transformation wurde der Zeitraum um das vierfache auf 468 Wochen erhöht. In jedem dieser vier Zeiträume ist die Verteilung der Daten identisch. Um dies zu erreichen, mussten die ID's und die Zeitangaben innerhalb der Datensätze angepasst werden. Auch hierdurch konnte sichergestellt werden, dass die Daten über alle Dimensionen hinweg gleichmäßig erhöht wurden.

### 3.1.2 Quellsysteme

Die Quellsysteme dienen sowohl für das traditionelle als auch für das virtuelle Data Warehouse als Startpunkt. Aus diesen werden sämtliche Daten extrahiert und in die Data Warehouses geladen. Bereits bei der Generierung der Quelldaten war klar, dass für die Evaluierung eine variable Anzahl von Quellen mit variabler Datengröße erforderlich sein wird, um die Effizienz der verschiedenen Ansätze in unterschiedlichen Situationen untersuchen zu können. Es müssen also schnell und einfach beliebig viele Quellen mit beliebigen Daten erzeugt werden können.

**Erste Idee für die Umsetzung der Quellsysteme**
Der erste Ansatz bestand darin die Quellen durch MySQL-Datenbanken[3] darzustellen. Es wurden Skripte geschrieben, um automatisiert eine beliebige Anzahl an MySQL-Servern auf einem Rechner aufzusetzen und die Tabellen für das Datenbankschema zu erzeugen. Dieses Skript könnte leicht erweitert werden, um die Aufgaben stattdessen per SSH[4] auf einem anderen Rechner auszuführen und so die Server auf verschiedenen Instanzen laufen zu lassen. Gleichzeitig wurde ein Skript geschrieben, um diese Server anschließend mit einem beliebigen Anteil des gesamten Datensatzes zu befüllen. Um diesen Vorgang zu beschleunigen, wurden die Daten nach ihren ID's vorsortiert und im Batchverfahren geladen.

[3] https://www.mysql.com
[4] https://www.ssh.com

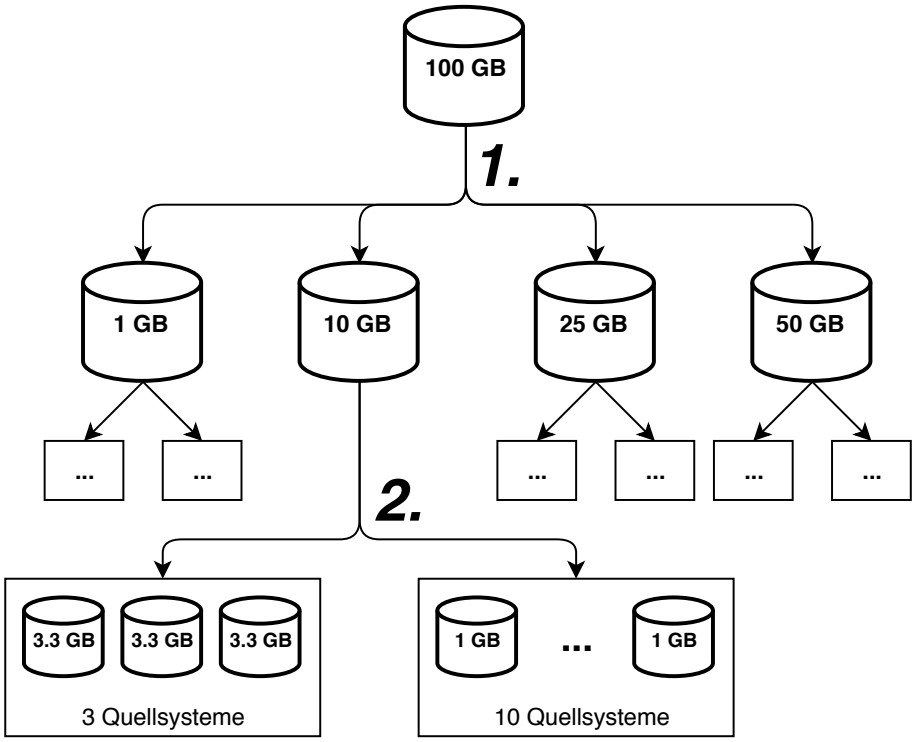



Abbildung 3.2: Extraktion der Quellen aus dem Gesamtdatensatz
Quelle: Eigene Darstellung

An dieser Stelle wurde festgestellt, dass diese Methode vermutlich bei der Evaluierung für einen enormen Zeitaufwand durch das langsame befüllen und anschließende extrahieren der Daten sorgen würde. Trotz der angewendeten Optimierungen dauerte allein die Befüllung einer Quelle mit einem Teil der Daten mehrere Minuten. In der praktischen Evaluierung würde dieser Prozess häufig, auf vielen Quellen, und auf größeren Datensets ausgeführt werden müssen. Es wurde die Entscheidung getroffen die Daten "vorextrahiert" in die Staging Area zu legen. Bei der Untersuchung mit ausschließlichem Blick auf die Anforderung "Volume" macht es keinen Unterschied, ob die Daten extrahiert werden müssen oder nicht. Dies gilt sowohl für den dimensionalen Datenwürfel als auch für das semi-virtuelle Data Warehouse.

**Zweiter letztendlich umgesetzter Ansatz**
Der gesamte Datensatz wurde in das Hadoop Cluster geladen, wo er anschließend weiterverwendet werden konnte. Im HDFS dieses Clusters wurde ein Ordner für die unverarbeiteten Quelldaten angelegt ('/etl/sources'). Anschließend wurden Skripte geschrieben, mit denen beliebig große Datensätze für beliebig viele Quellen aus dem Gesamtdatensatz extrahiert werden konnten. Diese werden automatisch in den 'sources' Ordner für die Quelldaten geladen. Eine Quelle wird hierbei einfach durch einen Ordner im HDFS repräsentiert. Dieser beinhalten weitere Ordner für die jeweiligen Tabellen und deren Daten. Die Skripte werden durch Abbildung 3.2 zusammengefasst. Der Vorgang besteht aus zwei Schritten.

1. *Extraktion einer beliebigen Datenmenge aus dem gesamten Datenset*

   Um eine möglichst gleichmäßige Extraktion der Daten aus dem Gesamtdatensatz zu erhalten, wurden die Warenkorb-ID's anhand des modulo-Operators gefiltert. Wenn aus dem Datensatz beispielsweise 25GB extrahiert werden sollen, dann kann dies dadurch erreicht werden, dass nur jeder vierte[5] Warenkorb in das neue Datenset übernommen wird. Es müssen also sowohl die Warenkörbe, als auch sämtliche Transaktionen anhand der Warenkorb ID gefiltert werden. Die Dimensionstabellen können anhand des resultierenden Datensets modifiziert werden (Beispielsweise wenn bestimmte ProduktID's nicht mehr verwendet werden). In der Praxis wurden, wie in der Abbildung dargestellt, vier Datensets extrahiert mit den Größen 1GB, 10GB, 25GB und 50GB.

2. *Aufteilung dieser Datenmenge auf eine beliebige Menge an Quellsystemen*

   Anschließend musste die resultierende Datenmenge auf die Quellen verteilt werden. Wenn Beispielsweise ein Test auf einem Data Warehouse mit 25GB an Daten und zehn Quellsystemen durchgeführt werden soll, dann müssen die 25GB gleichmäßig auf zehn verschiedene Quellen verteilt werden. Auch hier wurde der modulo Operator verwendet um diese gleichmäßige Verteilung sicherzustellen. Die erste Quelle erhält jeden zehnten Warenkorb, angefangen beim allerersten. Die zweite Quelle erhält jeden zehnten Warenkorb, angefangen beim zweiten, und so weiter. In der Praxis wurden die Tests auf Data Warehouses durchgeführt die einmal auf drei und einmal auf zehn Quellen basierten. Aus diesem Grunde mussten für alle Datensets (1GB, 10GB, 25GB, 50GB) die Daten für drei und für zehn Quellen extrahiert werden, wie in Abbildung 3.2 dargestellt.

[5] Filterung anhand der Warenkorb-ID: (ID % 4) == 0

## 3.2 Vergleichbarkeit der Implementierungen

Es wurde versucht die Ansätze vergleichbar zu halten, indem ähnliche Optimierungen und Technologien verwendet wurden. Aufgrund der sehr unterschiedlichen Ansätze war dies aber nur bis zu einem bestimmten Grad möglich.

**Ähnlichkeiten**
Nachfolgend werden einige Punkte erläutert, bei denen beide Architekturen ähnliche Ansätze verwenden:

- *Identische Analysen und Datensätze*

  Mit beiden Ansätzen wurden identische Analysen auf identischen Datensätzen durchgeführt und es wurde sichergestellt, dass die gelieferten Ergebnisse auch gleich sind. Die verwendeten SQL-Abfragen unterscheiden sich etwas, weil das virtuelle Data Warehouse kein Sternschema verwendet.

- *Ähnliche Technologien*

  Zusätzlich wurden für die Umsetzung beider Data Warehouses ähnliche Technologien verwendet. Die Daten liegen bei beiden Ansätzen im spaltenorientierten Tabellenformat (Parquet) im HDFS vor. Der Zugriff auf die Daten wird bei beiden durch Impala durchgeführt.

- *Ähnliche Optimierungen*

  Auch wurde versucht, die Menge an angewendeten Optimierungen bei beiden Ansätzen ähnlich zu halten. Dies war aufgrund des unterschiedlichen Aufbaus der darunterliegenden Daten nur bedingt machbar und sinnvoll. Die Tabellen werden bei beiden Ansätzen wie bereits erwähnt spaltenbasiert im Parquet-Format gespeichert. Die großen Tabellen werden bei beiden Data Warehouses durch eine Kompression mit Snappy um einen ähnlichen Faktor (10x) komprimiert. Das Caching wird auch bei beiden Ansätzen für alle kleinen Tabellen angewendet. Der Replikationsfaktor wurde so gewählt, dass alle Arbeiterknoten im Cluster alle kleinen Tabellen im Speicher halten. Hier ergibt sich aber ein großer Nachteil für das virtuelle Data Warehouse, der Grund wird etwas später erläutert. Auch die Partitionierung wurde bei beiden Ansätzen nach Jahr und Monat durchgeführt. Hieraus ergibt sich aber an dieser Stelle ein Unterschied bei beiden Ansätzen. Weil beim semi-virtuellen Ansatz die Daten noch in der rohen Schicht vorliegen, ergibt sich eine viel feinere Aufteilung der Datenmengen. Schließlich wird die Partitionierung nicht einmal auf den gesamten Datensatz angewendet, sondern jeweils einmal für jede Quelle. Ein mögliches Resultat wäre, dass die Partitionierung hierdurch zu tief geht und der generierte Nutzen sich reduziert. In der Evaluierung werden die Ergebnisse dieser Effekte (mit/ohne Partitionierung) ausführlich erläutert.

**Unterschiede**
Diese Idee, die beiden Technologien möglichst ähnlich zu gestalten, kann nur bis zu einer Grenze angewendet werden. Schließlich unterscheiden sich die beiden Ansätze ziemlich stark, woraus sich auch ergibt, dass sich ein optimaler Aufbau beider Architekturen sehr unterscheidet. Ein fairer Vergleich ist also nur möglich, wenn beide

Ansätze für sich selbst möglichst ähnlich weit optimiert werden. Nachfolgend ein paar Beispiele, an denen beide Ansätze unterschiedliche Wege gehen müssen.

- *Datenbankschema*

  Das Datenbankschema im Sternschema zu entwerfen ist eine sehr gute Optimierung für ein traditionelles Data Warehouse. Es bildet einen guten Kompromiss zwischen komplett normalisierten und denormalisierten Tabellen und liefert in der Praxis eine sehr gute Performanz. Für ein virtuelles Data Warehouse ist dieser Weg aber nicht praktikabel. Der Aufbau des Sternschemas ist mit einem Aufwand verbunden, den das virtuelle Data Warehouse regelmäßig durchführen müsste. Dieser Aufwand zahlt sich nicht aus, weil die Ergebnisse schlecht wiederverwendet werden können. Der Aufbau der Transformationen des virtuellen Data Warehouses werden weiter unten erläutert.

- *Partitionierung*

  Ein weiteres Beispiel ist die Partitionierung. Wie bereits erwähnt, sehen die abzufragenden Daten bei beiden Ansätzen unterschiedlich aus (Sternschema vs. Rohdatenschicht). Dadurch, dass in der Rohdatenschicht die Quellen noch nicht integriert sind, würde eine Partitionierung nach Jahr/Monat beim virtuellen Data Warehouse in einer viel tieferen Unterteilung der Daten resultieren. Eine Unterteilung in sehr kleine Dateien ist aufgrund des Overheads beim scannen suboptimal, vor allem bei geringen Datenmengen.

- *Caching*

  Das letzte Beispiel ist das Caching von Tabellen. Beim traditionellen Data Warehouse kann ein großer Nutzen durch das Caching von Dimensionstabellen generiert werden. Dadurch müssen diese nicht erst gelesen oder über das Netzwerk geschickt werden um beim Verbund mit der Faktentabelle verwendet werden zu können. Beim virtuellen Data Warehouse ist dies nicht möglich, auch das Caching von kleinen Tabellen aus den Quellen bringt in der Praxis nicht viel. Der Grund hierfür ist, dass Transformationen und Verbunde häufig auf den Ergebnissen von anderen Transformationen beruhen. Die Zwischenergebnisse, welche an dieser Stelle einen großen Teil der Datenmenge ausmachen können, müssen also trotzdem verteilt und verarbeitet werden.

  In der Praxis werden bei virtuellen Data Warehouses nicht die Quelltabellen, sondern die fertig transformieren virtuellen Tabellen gecached. Dies ist eine der wichtigsten Optimierungen für virtuelle Data Warehouses [van der Lans, 2012]. Aufgrund der großen Datenmengen von transformierten und gejointen Zwischenergebnissen werden diese häufig nicht im Arbeitsspeicher, sondern auf der Festplatte gecached. Die Wahl kann hier für jede Tabelle separat getroffen werden. Leider wurde erst recht spät entdeckt, dass Impala ein caching von virtuellen Tabellen nicht erlaubt, deshalb kann dieser Aspekt bei der Evaluierung nicht beleuchtet werden.

Alles in allem wurden die Architekturen für sich optimiert. Wo es geht wurde versucht, die Optimierungen und Ansätze ähnlich zu gestalten.

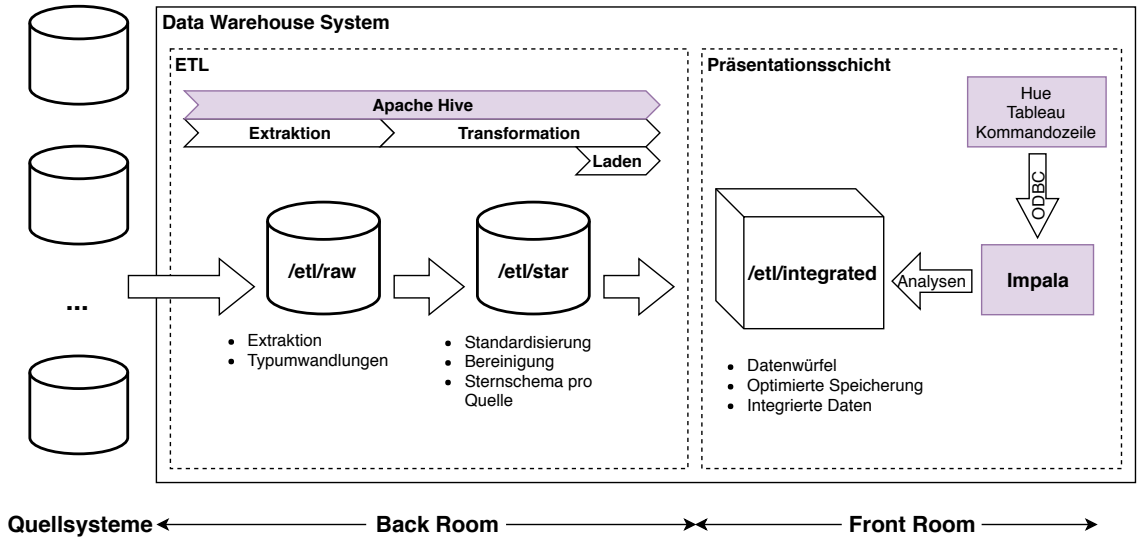



Abbildung 3.3: Implementierung traditionelles Data Warehouse
Quelle: Eigene Darstellung

## 3.3 Traditionelles Data Warehouse

Bei der Implementierung des traditionellen Data Warehouses wurde der Ansatz nach Kimball verfolgt. Das heißt, es wird ausgehend von einem Geschäftsprozess ein Datenwürfel gebaut, der für analytische Anfragen optimiert ist. Die Pfadangaben (wie '/etl/raw') beziehen sich auf das HDFS und enthalten Ergebnisse / Zwischenergebnisse der ETL-Prozesse. Bei der Beschreibung der einzelnen Komponenten werden der Aufbau und die verwendeten Technologien sehr grob beschrieben. Um auch einen detaillierteren Einblick in den Aufbau der Transformationen zu bieten, wird ein kleiner Ausschnitt der Transformationen (Teil der Integration) durch Codeausschnitte erläutert. Zum Ende dieses Abschnitts werden noch verwendete Optimierungen und die Art und Weise, wie Anfragen durchgeführt werden, erläutert. Die Architektur wird in Abbildung 3.3 zusammengefasst. Lila hinterlegte Elemente repräsentieren die für die jeweiligen Aufgaben verwendeten Technologien.

### 3.3.1 Quellen und Extraktion

Wie bereits in der vorherigen Sektion erwähnt wurden die Quellen für die Evaluierung nicht durch echte Systeme repräsentiert. Stattdessen liegen die Daten im Format der Quellsysteme bereits in der Staging Area ('/etl/sources') des Data Warehouses. Dies ändert nichts an den (E)TL-Prozessen, die durchgeführt werden müssen. Nur die Extraktion der Daten über das Netzwerk fällt weg.

Fast sämtliche Transformationen in der ETL-Phase werden über Apache Hive durchgeführt. Das heißt, die Transformationen werden durch SQL-Anfragen beschrieben. Die Extract-Phase des ETL-Prozesses ist normalerweise dafür da, die Daten aus den Quellsystemen zu extrahieren und leichte Transformationen wie Datentyptransformationen durchzuführen. In dieser Implementierung findet nur eine Transformation der Datentypen in das Format des Data Warehouses statt (z.B.: VARCHAR(20) zu STRING). Die Daten werden hierbei von der '/etl/sources' Schicht in die '/etl/raw' Schicht überführt. Diese enthält bis auf die Datentypen ein identisches Abbild

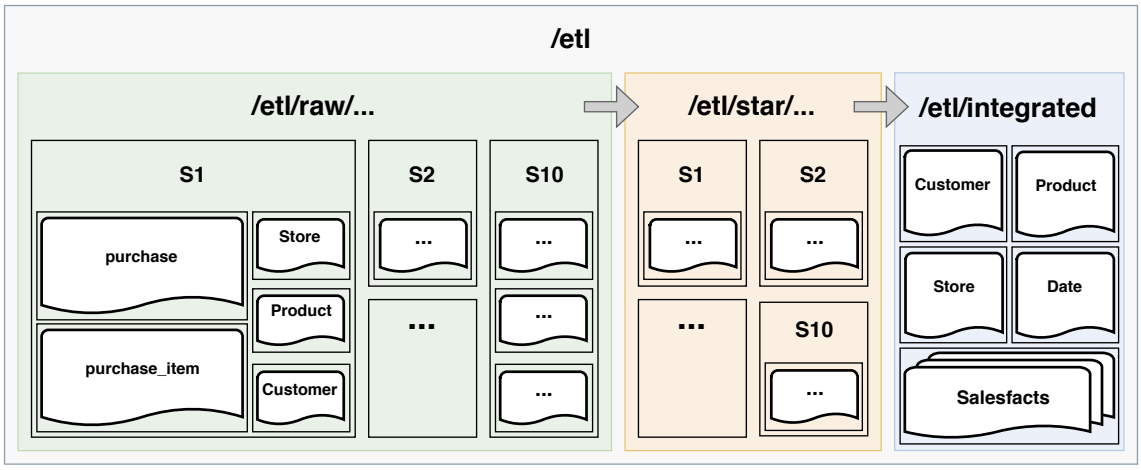



Abbildung 3.4: Ordnerstruktur HDFS /etl
Quelle: Eigene Darstellung

der Daten aus den Quellen und hat den Aufbau des grün hinterlegten Bereichs in Abbildung 3.4. S1 bis S10 stellen hierbei die Quellen dar, welche nach dem Datenbankschema aus Abbildung 3.1 modelliert wurden. Allgemein stellt die Abbildung die persistierten Schichten des ETL-Prozesses dar. In '/etl/star' wird jede bereinigte und standardisierte Quelle im Sternschema zwischengespeichert. Das finale integrierte Sternschema wird für Anfragen optimiert in '/etl/integrated' gesichert.

### 3.3.2 Transformation und Laden

Die Transformations- und Ladephase sind bei dieser Implementierung vermischt, weil sowohl die Transformationen als auch die Analysen im HDFS des Clusters durchgeführt werden. Nach der letzten Transformationsphase liegen die Daten quasi fertig geladen im Data Warehouse, sie müssen nicht zusätzlich in eine externe Datenbank exportiert werden. Die gesamte Transformationsphase besteht hierbei aus zwei Schritten.

Im ersten Schritt werden die Daten für jede Quelle separat bereinigt und in ein einheitliches Format überführt. Die aufgeführten Arten von Transformationen sind beim traditionellen und virtuellen Data Warehouse durch unterschiedliche Umsetzungen vorhanden. Die erwähnten Optimierungen werden beim virtuellen Data Warehouse auf die rohe Schicht direkt beim Extrahieren angewendet, das traditionelle Data Warehouse wendet diese bei der Erstellung der Tabellen für das Sternschema an.

- Textersetzungen: 5
- Konkatenation von Spalten: 1
- Datumstransformationen: 6
- Standardisierung von Spalten: 4
- Verbunde

- Vereinigungen
- Optimierungen:
  - Partitionierung
  - Kompression
  - Zusammenführung kleiner Dateien

Die Wahl der Transformationen wurde basierend auf dem Use-Case getroffen. Es wurden Transformationen gewählt, die auch in realen Projekten vorkommen könnten, wie das ersetzen von Ländernamen durch deren Abkürzungen (Textersetzung) oder die Transformation der Adressdaten in ein einheitliches Format (Standardisierung). Die Anzahl der Verbunde und Vereinigungen wurde nicht aufgelistet, weil diese sich beim traditionellen und virtuellen Ansatz unterscheiden. Beim virtuellen Ansatz hängen diese von der virtuellen Tabelle und der Anzahl der Quellen ab.

Der Aufbau der Ordnerstruktur nach diesem Schritt ist sehr ähnlich zu dem Aufbau nach der Extraktionsphase, für jede Quelle existiert ein Ordner für dessen Daten. Der größte Unterschied besteht darin, dass die Daten in das finale Sternschema transformiert wurden. Nach diesem Schritt liegen die Daten im HDFS-Pfad '/etl/star' und die Größe des gesamten Datensatzes ist durch die Kompression um ein zehnfaches gesunken.

Im zweiten Schritt findet nun die Integration sämtlicher Daten aller Quellen in das finale Sternschema des Data Warehouses statt. Durch die Vorarbeiten im ersten Schritt ist dies ohne großen Aufwand möglich, wichtig ist nur, dass die Fremdschlüssel der Faktentabellen nach der Integration auf die richtigen Werte in den Dimensionstabellen zeigen. Zunächst werden sämtliche Daten aus der ersten Quelle in den finalen HDFS-Ordner ('/etl/integrated') kopiert. Anschließend werden alle verbleibenden Quellen Schritt für Schritt in diesen Ordner / Tabelle integriert. Um einen Einblick in den Aufbau der Transformationen zu gewähren, werden nachfolgend Teile der Integration anhand von Codeausschnitten erläutert. Diese Integration kann in folgende Teilschritte unterteilt werden:

1. Feststellen welche Werte in der zu integrierenden Tabelle neu sind
2. Anhängen dieser neuen Werte mit modifizierten ID's
3. Erstellen einer Abbildung für alle Zeilen der zu integrierenden Dimensionstabelle: alte ID → neue ID in der integrierten Dimensionstabelle

Anschließend kann die Faktentabelle integriert werden. Hierbei müssen alle Fremdschlüssel für die Dimensionstabellen anhand der vorher erstellten Abbildungen angepasst werden. Im folgenden wird dieser Integrationsprozess anhand der Produktdimension verdeutlicht. Zunächst wird die letzte ProductID aus der integrierten Tabelle extrahiert:

```
maxProductID=$(
SELECT max(productID)
FROM $integrateddbName.product;)
```

Anschließend werden die Dimensionstabellen zusammengeführt indem alle neuen Produkte hinzugefügt werden. Die neuen ID's beginnen mit *maxProducctID*+1 und werden inkrementell erhöht. Durch die Kombination von "left outer join" und der "where-Bedingung" werden nur die neuen Produkte hinzugefügt:

```
INSERT INTO $integrateddbName.product
SELECT ($maxProductID + 1) + ROW_NUMBER() OVER (ORDER BY p2.categoryID)
AS productID, p2.categoryID, p2.name, p2.description, p2.price,
p2.vendor, p2.categoryname, p2.categorydesc
FROM $otherdbName.product AS p2
LEFT OUTER JOIN $integrateddbName.product AS pintegrated
ON (p2.name = pintegrated.name AND p2.vendor = pintegrated.vendor)
WHERE pintegrated.productID IS NULL;
```

Zum Schluss findet die Integration der Faktentabellen statt. In der inneren Abfrage wird die Abbildung von den Produkt-ID's aus der alten Produkttabelle zu den ID's der neuen integrierten Produkttabelle erzeugt (*potherID* $\rightarrow$ *pintegratedID*). Anschließend wird diese Abbildung unter Verwendung der alten ProduktID (*potherID*) mit der Faktentabelle gejoint. Die neue ID (*pintegratedID*) wird anschließend als finaler Produktschlüssel in dieser Faktentabelle verwendet. Dadurch wird sichergestellt, dass die Fremdschlüssel in der Faktentabelle korrekt bleiben. Die restlichen Dimensionen wurden hierbei wegen Platzmangel ausgeblendet:

```
INSERT INTO $integrateddbName.salesfacts
SELECT pmapping.pintegratedID AS productKey,
smapping.sintegratedID AS storeKey, sf.dateKey,
cmapping.cintegratedID AS customerKey, sf.quantity, sf.priceeach,
sf.day, sf.year, sf.month
FROM $otherdbName.salesfacts as sf
INNER JOIN
(
SELECT pintegrated.productID AS pintegratedID,
pother.productID AS potherID
FROM $integrateddbName.product pintegrated
INNER JOIN $otherdbName.product pother
ON (pother.name = pintegrated.name
AND pother.vendor = pintegrated.vendor)
) AS pmapping
ON (sf.productKey = pmapping.potherID)

INNER JOIN
[...]
```

Diese Integrationen werden für alle Quellen nacheinander durchgeführt bis der finale Datenwürfel steht. Die Tabellen dieses Würfels sind nach dem Schema in Abbildung Abbildung 3.5 modelliert.

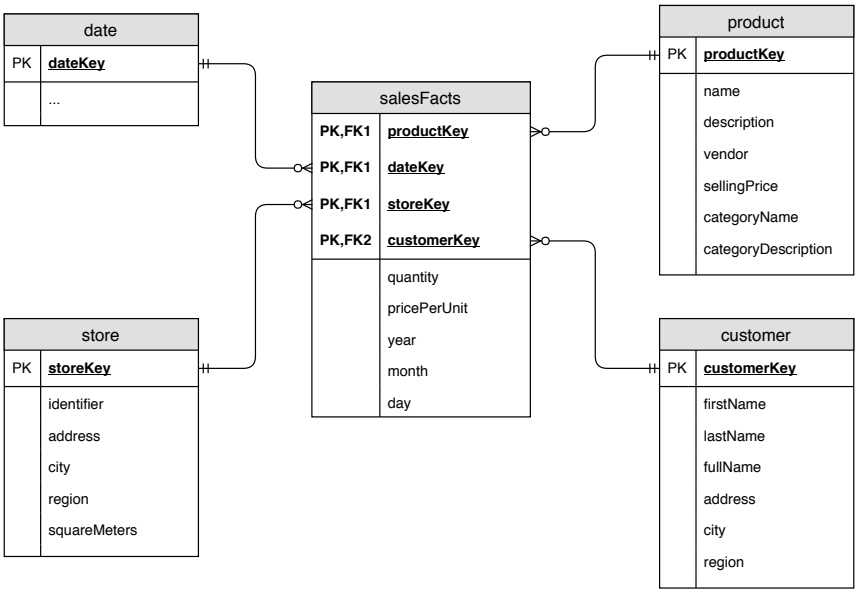



Abbildung 3.5: Sternschema der Präsentationsschicht
Quelle: Eigene Darstellung

### 3.3.3 Optimierung

Die Transformation in das Sternschema ist nicht die einzige Optimierung, die für das traditionelle Data Warehouse durchgeführt wurde. Nachfolgend werden weitere angewendete Techniken kurz aufgeführt.

- *Spaltenorientierte Tabellenspeicherung*:

  Das Parquet-Format ist ein spaltenorientiertes Speicherformat für das Hadoop Ecosytem.[6] Es ist für große analytische Anfragen optimiert und erlaubt durch die spaltenorientierte Speicherung hohe Kompressionsraten, wodurch Daten effizienter übertragen werden können. Außerdem müssen bei Abfragen von nur einzelnen Spalten auch wirklich nur diese gelesen werden, dies ist vor allem bei analytischen Anfragen wichtig.

- *Kompression*:

  Durch die Kompression von Tabellen können die Daten schneller von der Festplatte gelesen werden und können schneller zwischen den Rechenknoten im Cluster verteilt werden. Als Kompressionsalgorithmus wurde Snappy[7] verwendet. Dieser bietet einen guten Mittelwert zwischen Kompressionsintensität und Kompressionsgeschwindigkeit. Im Durchschnitt wurden die Datenmengen hierdurch um das zehnfache reduziert. Unterstützt wird die Kompression durch das bereits erwähnte Spaltenbasierte Parquet-Format.

- *Partitionierung*:

  Um nach Zeiträumen selektierende Anfragen zu optimieren wurde die Faktentabelle (Salesfacts) nach Jahr und Monat partitioniert. Hierdurch wird im

[6] https://parquet.apache.org
[7] https://github.com/google/snappy

HDFS für jedes Jahr und jeden Monat ein separater Ordner angelegt, der die jeweiligen Daten enthält. Bei einer selektierenden Anfrage nach Zeit werden dann nur die wirklich benötigten Daten geladen und verarbeitet.

- *Zusammenführung kleiner Dateien*:

  Durch die MapReduce ETL-Prozesse entstehen viele kleine Dateien, vor allem wenn die Dateien zusätzlich noch partitioniert werden. Damit die einzelnen Parquet-Dateien auf eine durchschnittliche Größe von 128 MB kommen, werden die Dateien nach den Transformationen noch zusammengeführt.

- *Caching*:

  Beim Caching werden häufig benötigte Daten zwischengespeichert, um die Performanz beim Laden zu verbessern. In dieser Implementierung wurden alle Dimensionstabellen auf alle Arbeiterknoten verteilt und im Arbeitsspeicher gehalten. Hierdurch müssen so gut wie keine Daten mehr über das Netzwerk geschickt werden, weil jeder Knoten alle benötigten Daten (Dimensionstabellen + Faktentabellen) bereits vorliegen hat. Jeder Arbeiterknoten muss nur noch die für ihn bestimmten Teile der Faktentabelle von seiner Festplatte laden.

### 3.3.4 Abfragen

Die im Sternschema vorliegenden Tabellen können nun zum Beispiel durch Hive oder Impala über SQL abgefragt werden. Impala ist hierbei eine Technologie, die nicht auf MapReduce baut sondern, auf eine kurze Latenz bei analytischen Anfragen setzt. Aus den im theoretischen Kapitel bereits erwähnten Gründen wird sowohl beim traditionellen als auch beim virtuellen Data Warehouse Impala für die Verarbeitung von analytischen Abfragen verwendet.

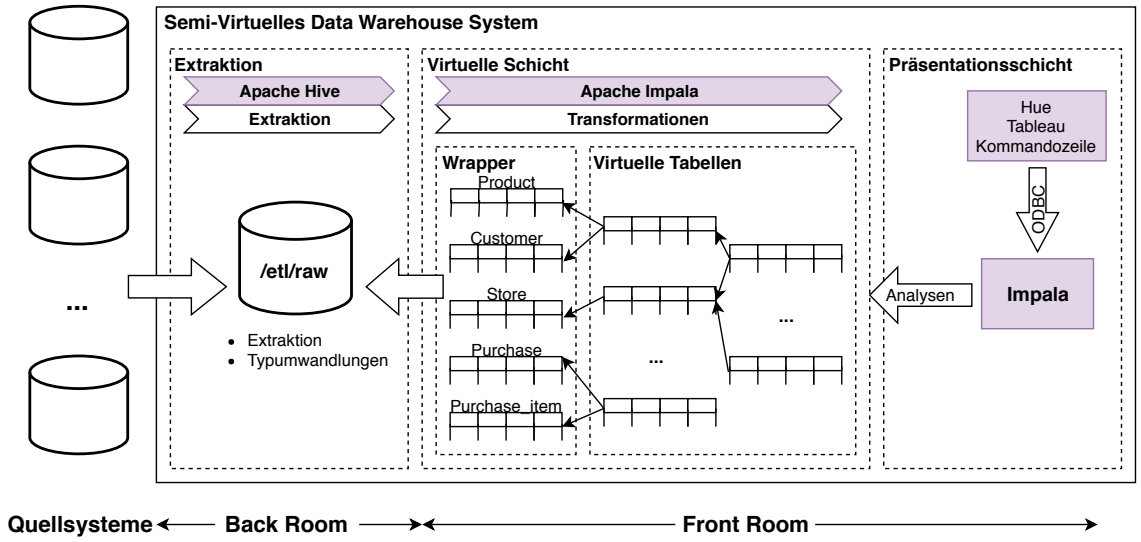



Abbildung 3.6: Implementierung semi-virtuelles Data Warehouse
Quelle: Eigene Darstellung

## 3.4 Semi-virtuelles Data Warehouse

Das semi-virtuelle Data Warehouse ist das Vergleichsobjekt, mit dem der dimensionale Datenwürfel nach Kimball in der Evaluierung verglichen werden soll. Dieser Ansatz ist ein Hybrid, weil die Quellen nicht virtualisiert werden. Diese werden wie bei einem traditionellen Data Warehouse extrahiert und historisiert abgespeichert. Die Virtualisierung wird auf den gesamten Bereich nach der Extraktion angewendet, also auf den gesamten (E)TL-Prozess bis zu den SQL-Abfragen der Nutzer. Die einzigen auf die Quellen angewendeten Transformationen sind Datentyptransformationen und Optimierungen (Partitionierung, Kompression), um die Kompatibilität mit dem Data Warehouse herzustellen. Ansonsten liegen die Daten komplett unverarbeitet im HDFS vor. Die finale Implementierung enthält abhängig von der Anzahl der Quellen die folgende Menge an Wrapper- und Virtuellen Tabellen:

| | Wrappertabellen | Virtuelle Tabellen | Insgesamt |
|---|---|---|---|
| 3 Sources | 15 | 31 | 46 |
| 10 Sources | 50 | 94 | 144 |

### 3.4.1 Quellen und Wrapper

Nachfolgend wird die Umsetzung der einzelnen Bausteine der Architektur grob erläutert. Da es sich hierbei um eine semi-virtuelle Umsetzung handelt, liegen die Quelldaten bereits historisiert im HDFS vor. Auf diesen Daten werden Wrappertabellen aufgesetzt, die die Daten exakt wiedergeben. Es werden hierbei höchstens Datentyptransformationen durchgeführt. Ein Beispiel für eine Wrappertabelle wird in Abbildung 3.7 dargestellt. Es wird eine Wrappertabelle auf im Parquet-Format vorliegende Produktdaten erzeugt, die Preisspalte wird vom Typ STRING zum Typ Dezimal umgewandelt. Diese und weitere Wrappertabellen übernehmen die Daten aus den Quellen und führen lediglich simple Datentyptransformationen wie oben

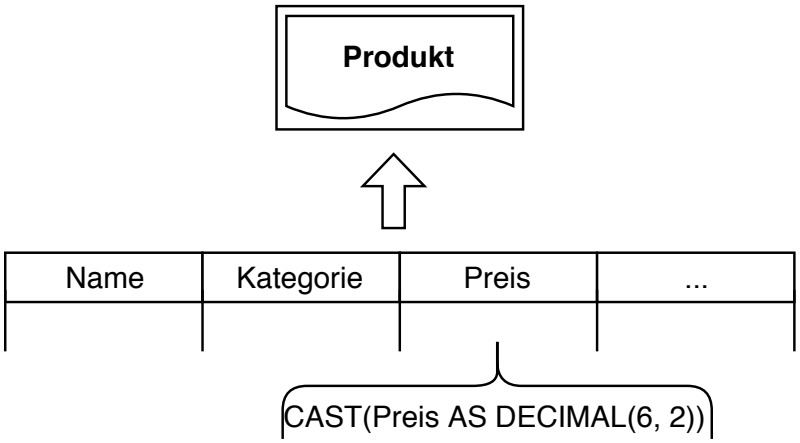



Abbildung 3.7: Beispiel Wrappertabelle
Quelle: Eigene Darstellung

gezeigt durch. Wenn nun ein Data Warehouse auf zehn Quellen basiert und jede davon fünf Tabellen hat, müssen bereits fünfzig Wrappertabellen erzeugt werden. Die Erstellung aller Wrappertabellen wurde durch Skripte automatisiert, welche die Tabellen abhängig von der Anzahl der vorhandenen Quellen erzeugen können.

### 3.4.2 Virtuelle Tabellen

Auf den Wrappertabellen wurde eine Hierarchie von virtuellen Tabellen aufgebaut. Diese geben dieselben Transformationen wieder, wie das traditionelle Data Warehouse, sie sind nur anders strukturiert. Diese Hierarchie ist in Schichten aufgeteilt und wird in Abbildung 3.8 dargestellt, dies ist eine detaillierte Darstellung der virtuellen Schicht aus Abbildung 3.6. Die einzelnen Schichten werden nachfolgend erläutert, wobei die nullte Schicht die Wrappertabellen repräsentiert und zuvor bereits geklärt wurde.

Die erste Schicht ist für Transformationen zuständig, die pro Tabelle angewendet werden. Hierzu gehören unter anderem Bereinigungen, Standardisierungen und Normalisierungen, es finden noch keine Verbunde statt. Nachfolgend werden beispielhaft die Transformationen für die Kundentabelle aus der Wrappertabelle dargestellt:

```
CREATE VIEW customer AS
SELECT customerid, firstname, lastname,
CONCAT(lastname, '␣', firstname) AS fullname,
CONCAT(substr(address, 6), '␣', substr(address, 1, 4)) AS address,city,
CASE region
        WHEN 'niedersachsen' THEN 'NI'
        WHEN 'saarland' THEN 'SL'
        WHEN 'nordrhein-westfalen' THEN 'NW'
        WHEN 'hessen' THEN 'HE'
         [...]
ELSE region
END AS region
FROM wcustomer;
```

Die zweite Schicht ist nun dafür da, Verbunde von verschiedenen Tabellen innerhalb einer Quelle bereitzustellen. Abhängig von den Verbunden können auch Transfor-

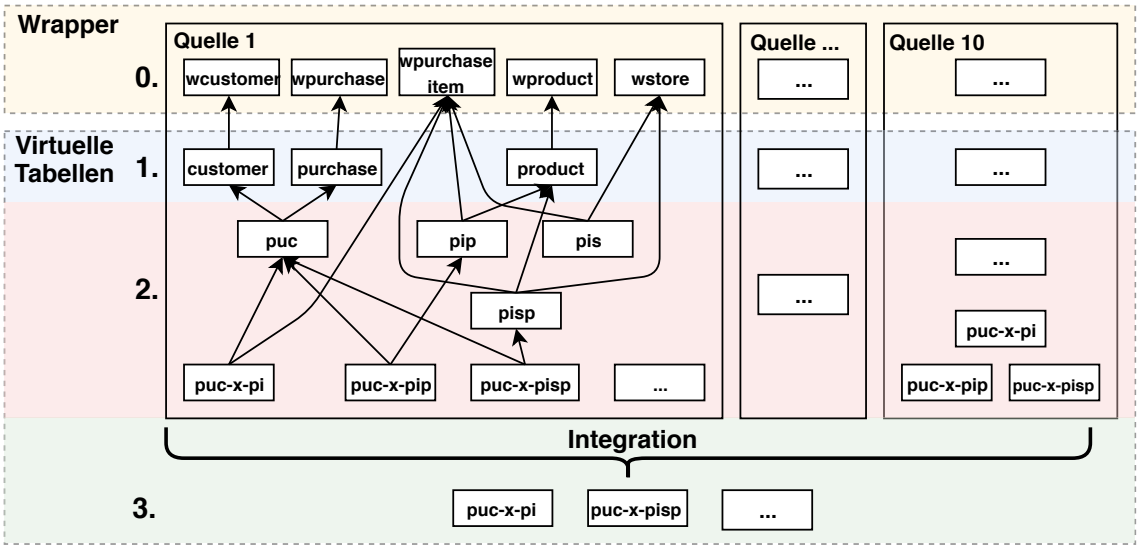



Abbildung 3.8: Aufbau der virtuellen Schicht
Quelle: Eigene Darstellung

mationen durchgeführt werden, die Informationen aus mehreren Tabellen benötigen. Diese Schicht ist hauptsächlich dafür da, die Definitionen von virtuellen Tabellen in der dritten Schicht zu vereinfachen. Es werden nur sehr wenige echte Transformationen durchgeführt. Die Abkürzungen in der Abbildung spiegeln die durchgeführten Verbunde wieder, zum Beispiel steht ”puc” für ”purchase” verbunden mit ”customer”. ”pip” steht für ”purchase_item” verbunden mit ”product”. Es wurden verschiedene Kombinationen bereitgestellt, weil nicht alle analytischen Anfragen Daten aus allen Tabellen benötigen. Nachfolgend die Definition der Tabelle ”puc”:

```
CREATE VIEW puc AS
SELECT purchaseid, c.firstname AS firstname, c.lastname AS lastname,
c.fullname AS fullname, c.address AS address, c.city AS city,
c.region AS region, date_part('year', 'date') AS year,
date_part('month', 'date') AS month, date_part('day', 'date') AS day
FROM purchase p
JOIN customer c ON (p.customerid = c.customerid);
```

Der Zweck der dritten Schicht ist es, Sichten für integrierte Daten bereitzustellen. Es werden also Tabellen über verschiedene Quellen hinweg zusammengeführt, hierbei finden auch nur wenige oder keine richtigen Transformationen statt. Dadurch, dass es für alle Quellen normalisierte Repräsentationen der Daten gibt, werden die meisten Integrationen anhand von Vereinigungen durchgeführt. Ein weiterer Zweck ist es, virtuelle Tabellen bereitzustellen, die vordefinierte analytische Anfragen repräsentieren. Dies kann für häufig angewendete und sich selten ändernde Anfragen durchgeführt werden. In der Praxis werden fast ausschließlich virtuelle Tabellen aus dieser Schicht durch die Anwender abgefragt. Es können aber natürlich auch Tabellen aus den anderen Schichten abgefragt werden und per Hand zusammengeführt werden. Da es mehrere Variationen der Data Warehouses gibt (z.B.: mit sich ändernder Anzahl der Quellen), wurde die Erstellung der virtuellen Tabellen durch dynamische Skripte realisiert. Bei den vorherigen SQL-Statements wurde alles, was nicht SQL

ist, entfernt. Nachfolgend wird beispielhaft ein Bash-Skript[8] dargestellt, welches alle Daten aus der Wrapper-Tabelle "purchase_item" aggregiert. Im Endeffekt ist es nur eine Vereinigung über alle "purchase_item"-Tabellen. Die SQL-Statements wurden markiert:

```
dbCount = $input
impalascript="$impalascript CREATE VIEW $todbName.pi AS"
for ((i=1;i<dbCount;++i)); do
        fromdbName="v$i"
        impalascript="$impalascript
        SELECT quantity, priceeach, year, month
        FROM $fromdbName.wpurchase_item
        UNION ALL"
done

fromdbName="v$dbCount"
impalascript="$impalascript
SELECT quantity, priceeach, year, month
FROM $fromdbName.wpurchase_item;"
```

### 3.4.3 Optimierungen

Es wurden einige Optimierungen durchgeführt, die denen des traditionellen Data Warehouses ähneln. An dieser Stelle werden nur die stark abweichenden Optimierungen kurz erläutert.

- *Aufbau der virtuellen Schicht*:

  Die Definitionen der virtuellen Tabellen bergen ein enormes Potential für Optimierungen. Allein die Definition einer SQL-Anfrage kann auf unterschiedliche Weisen durchgeführt werden, welche mit verschiedenen Laufzeiten verbunden sein können. Wenn man nun Anfragen auf Anfragen aufbaut, erhöht sich das Potential an dieser Stelle noch weiter. Der Aufbau in Abbildung 3.8 versucht die Menge an nötigen Verbunden zu minimieren und vereinfacht die Propagation von Bedingungen in geschachtelten Anfragen. Zunächst wurde ein anderer Ansatz (Sternschema) verfolgt, dieser wurde aber aufgrund einer sehr schlechten Performanz komplett überarbeitet. In dem verwendeten Aufbau gibt es sicher noch einige Optimierungsmöglichkeiten.

- *Caching*:

  Das Caching von virtuellen Tabellen ist in virtuellen Data Warehouses ein essentielles Feature um eine gute Performanz liefern zu können. Hierbei werden Zwischenergebnisse im Arbeitsspeicher oder auf der Festplatte periodisch gesichert. Leider wurde erst sehr spät herausgefunden, dass die verwendete Query-Engine (Impala) ein Caching von virtuellen Tabellen nicht ermöglicht. Es wird nur das Caching von "echten" Tabellen angeboten. Der Einfluss dieser Optimierung wird bei der Evaluierung somit nicht berücksichtigt, dies wäre jedoch ein interessanter Punkt für ein aufbauendes Thema.

[8] https://www.gnu.org/software/bash/

### 3.4.4 Abfragen

Auch bei diesem Ansatz werden die Anfragen durch die Impala Query-Engine durchgeführt. Die SQL-Anfragen sehen anders aus als beim traditionellen Data Warehouse, weil intern kein Sternschema modelliert wurde. Trotzdem werden dieselben analytischen Ergebnisse berechnet. Für alle Anfragen wurden die generischen virtuellen Tabellen aus der dritten Schicht (Abbildung 3.8) verwendet.

# 4. Evaluierung

In dem folgenden Kapitel werden die beiden vorgestellten Ansätze innerhalb von festgelegten Rahmenbedingungen evaluiert. Der Hauptfokus liegt darin herauszufinden, wie gut sie steigende Datenmengen verkraften können. Die Performanz wird anhand von drei Kriterien gemessen die am Anfang des Kapitels vorgestellt werden. Anhand dieser werden die beiden Ansätze miteinander verglichen, hierdurch soll der Einfluss der Virtualisierung auf die Performanz herausgefiltert werden. Zusätzlich werden die Auswirkungen von drei Parametern (Partitionierung, Anzahl der Quellen und Arten von Anfragen) auf die Leistungen gemessen. Dadurch wird untersucht, ob die beiden Ansätze aufgrund der unterschiedlichen Architekturen anders auf die Einflüsse der Parameter reagieren. Die Beobachtungen bei der Evaluierung werden diskutiert und es werden Schlussfolgerungen gezogen. Diese können letztendlich verwendet werden um die gestellten Forschungsfragen zu beantworten.

Bevor es mit der Evaluierung losgehen kann, müssen die Rahmenbedingungen festgelegt werden, innerhalb derer die Ergebnisse gültig sind. Hierfür ist der erste Abschnitt des Kapitels zuständig. Es werden die untersuchten Analysen und Parameter erläutert. Es werden Grenzen definiert, innerhalb derer die Ergebnisse gültig sind. Und es werden das verwendete Testsystem und die verwendeten Methoden zur Messung der Leistung beschrieben.

Anschließend werden die beiden Architekturen evaluiert. Hierbei werden die Ergebnisse zunächst bei beiden für sich betrachtet und dann übergreifend untersucht. Diese übergreifende Evaluierung beinhaltet einen Vergleich der generellen Leistung und eine Abwägung der Auswirkungen der Parameter.

Zum Schluss werden die zuvor untersuchten Ergebnisse diskutiert und Schlussfolgerungen gezogen. Es werden die Vor- und Nachteile der beiden Ansätze innerhalb der Rahmenbedingungen verdeutlicht und das Wissen zusammengefasst, welches für die Beantwortung der Forschungsfragen verwendet wird.

## 4.1 Überblick

In diesem Abschnitt werden generelle Informationen präsentiert, die für das Verständnis der darauffolgenden Evaluierung notwendig sind. In dem ersten Unterabschnitt findet eine Beschreibung der Situationen und Anfragen statt, in dessen Rahmen die Data Warehouses evaluiert werden. Zusätzlich werden die Abläufe während der Testdurchführung kurz beschrieben. Anschließend werden die Struktur und die vorhandenen Ressourcen des Testsystems erläutert. Daraufhin werden die Metriken vorgestellt, mit deren Hilfe die Leistungen der Systeme verglichen werden. Der letzte Abschnitt definiert die Grenzen und Rahmenbedingungen, innerhalb derer die Ergebnisse der Arbeit gültig sind.

### 4.1.1 Szenarien, Abfragen und Testdurchführung

Es gibt sehr viele Parameter, die einen Einfluss auf die Performanz eines Data Warehouses haben können. Dazu gehören viele optionale Optimierungen. Leider konnte für die praktische Evaluierung aufgrund des Aufwandes nur ein kleiner Teil dieser betrachtet werden. Zu den untersuchten Parametern gehören die folgenden:

- *Partitionierung*

  Dies ist eine sehr verbreitete Optimierung für Datenbanken allgemein. Für die Evaluierung wurde eine Partitionierung entlang der Zeit-Dimension verwendet. Das heißt, dass Anfragen die entlang dieser Dimension selektieren von der Partitionierung profitieren sollten. Die Partitionierung wurde als Parameter aufgenommen, weil es eine sehr verbreitete Optimierung ist die von vielen Datenbanken unterstützt wird.

- *Anzahl der Quellsysteme*

  Als zweiter Parameter wurde die Anzahl der vorhandenen Quellsysteme verwendet. Dies ist ein Punkt, der in der Praxis wenig oder kaum beeinflusst werden kann, im Falle des virtuellen Data Warehouses aber einen Einfluss auf die Leistung haben kann. Denn im Gegensatz zu traditionellen Ansätzen muss das virtuelle Data Warehouse die Transformation und Integration der Quellsysteme spontan durchführen. Es wird angenommen, dass sich dieser Aufwand bei vielen Quellsystemen in der Verarbeitungszeit wiederspiegelt. Konkret wurden die Leistungen bei drei und zehn Quellsystemen untersucht.

- *Datenmenge*

  Der offensichtliche letzte Parameter ist die zu verarbeitende Datenmenge. Unter anderem soll schließlich auch die Skalierbarkeit beider Ansätze untersucht werden. Die Tests wurden für die folgenden Datenmengen durchgeführt:
  1GB, 10GB, 25GB und 50GB

Eine Kombination dieser Parameter repräsentiert ein Szenario, es ergeben sich also 2 (mit/ohne Partitionierung) * 2 (3 / 10 Quellsysteme) * 4 (Datenmengen) = 16 Szenarien, unter denen die beiden Ansätze verglichen werden. Wenn einer dieser Parameter geändert wird, dann muss sowohl das traditionelle als auch das virtuelle

Data Warehouse neu aufgebaut werden. Bei den Parametern Partitionierung und Anzahl der Quellsysteme muss die Struktur der Data Warehouses angepasst werden. Es ändert sich zum Beispiel fast der gesamte ETL-Prozess beim traditionellen Data Warehouse und fast alle virtuellen Schichten beim virtuellen Data Warehouse, wenn die Anzahl der Quellen geändert wird. Schließlich müssen neue Quellen hinzugefügt und integriert werden, oder eben entfernt werden. Das selbe gilt für die Partitionierung. Wenn die Datenmenge sich ändert, dann muss die Struktur der Data Warehouses nicht angepasst werden. Das traditionelle Data Warehouse muss aber mit den neuen Daten beladen werden. Um alle Szenarien abzudecken mussten also insgesamt 32 Data Warehouses gebaut oder neu beladen werden. Wenn für ein spezifisches Szenario das traditionelle und das semi-virtuelle Data Warehouse erstellt wurden, dann konnten die zu testenden Anfragen durchgeführt werden. Diese werden nachfolgend erläutert.

**Analytische Anfragen**
Für jedes dieser Szenarien wurden anschließend unterschiedliche SQL-Anfragen durchgeführt, die Laufzeiten dieser Anfragen wurden gemessen. Es handelt sich hierbei ausschließlich um analytische Anfragen. Das heißt, es wurden Aggregationen über Fakten durchgeführt und Verbunde mit dimensionalen Tabellen geschlossen um an kontextuelle Informationen zu kommen und Selektionen durchzuführen.

Es wurden insgesamt neun Arten von Anfragen definiert. Die konkreten Ausprägungen dieser Anfragen unterscheiden sich beim traditionellen und virtuellen Data Warehouse aufgrund des unterschiedlichen Aufbaus dieser, es werden jedoch identische Ergebnisse berechnet. Es gibt eine riesige Menge an möglichen analytischen Anfragen die verwendet werden könnten. Damit die letztendliche Wahl nicht willkürlich geschieht, wurden die Anfragen anhand von zwei klassifizierenden Dimensionen definiert:

- *Volumen*

  Anhand dieser Dimension wird die zu aggregierende Datenmenge definiert. Praktisch wird diese Dimension durch unterschiedlich restriktive Selektionen umgesetzt. Die folgenden drei Abstufungen wurden verwendet:

  - Hoch: Keine Selektion; Es werden 100% der Daten verarbeitet
  - Mittel: Selektion entlang der Zeit-Dimension; 8% der Daten
  - Gering: Selektion entlang der Zeit- und Kunden-Dimensionen; 0,02% der Daten

- *Komplexität*

  Diese Dimension beschreibt den Aufwand der Berechnungen, die für das Umsetzten der Anfrage notwendig sind. Auch hier wurden drei Abstufungen verwendet:

  - Gering: Es wird bloß eine Aggregation der selektierten Daten durchgeführt. Im Detail wird eine Summe für die Anzahl der verkauften Produkte und der Profit berechnet.

  - Mittel: Zusätzlich zu den Aggregationen wird die Faktentabelle mit allen Dimensionstabellen verbunden
  - Hoch: Zusätzlich zu den anderen Berechnungen werden die Ergebnisse am Ende anhand des Namens der gekauften Produkte gruppiert und nach dem Profit sortiert.

Wenn alle Abstufungen kombiniert werden ergeben sich insgesamt neun Arten von Anfragen. Alles in allem wurden also 18 konkrete SQL-Anfragen definiert, neun für den traditionellen und virtuellen Ansatz. Wenn diese Anfragen mit den Szenarien kombiniert werden, ergeben sich insgesamt 288 zu untersuchende Datenpunkte.

Um Verzerrungen durch Ausreißer zu reduzieren wurden all diese Anfragen dreimal ausgeführt und der Durchschnitt für diese gebildet. Leider konnte eine gewünschte höhere Wiederholungszahl aus zeitlichen Gründen nicht angewendet werden. Um die Auswirkungen von Ausreißern auf die Ergebnisse einschätzen zu können, wurde die Varianz über diese Durchläufe untersucht. Diese stellte sich als relativ gering heraus. Auf die konkrete Untersuchung dieser und die Einflüsse auf die Ergebnisse der Arbeit wird in Abschnitt 4.4.2 eingegangen.

**Testdurchführung**
Die gesamte Testphase kann folgendermaßen zusammengefasst werden. Für jede der oben genannten Szenarien wurden die folgenden Schritte ausgeführt:

1. *Erstellen der Data Warehouses*

   Abhängig von dem aktuellen Szenario werden die Daten der Quellsysteme importiert, die beiden Data Warehouses gebaut, und das traditionelle Data Warehouse beladen.

2. *Durchführung der Tests*

   Anschließend werden sequentiell alle SQL-Anfragen über Impala auf beiden Data Warehouses ausgeführt und die benötigten Verarbeitungszeiten automatisch dokumentiert.

3. *Download benötigter Informationen*

   Zum Schluss werden alle benötigten Informationen für die Analysen heruntergeladen. Konkret bedeutet das, dass sämtliche Profiling-Informationen für jede Anfrage heruntergeladen werden. Im Falle von Impala werden diese Informationen in Form einer Website bereitgestellt.

### 4.1.2 Testsystem

Die Tests wurden auf einem Cluster innerhalb der firmeninternen Cloudumgebung durchgeführt, es wurden die folgenden Ressourcen reserviert:

- 64 vCPUs
- 128GB RAM
- 1TB Speicher

Für das automatische Setup der Infrastruktur und das Deployment von Cloudera wurden Terraform[1] und Ansible[2] verwendet. Aus Platzgründen und der fehlenden Relevanz wird jedoch nur auf eine Übersichtliche Verteilung der Ressourcen und Aufgaben auf den Hosts eingegangen.

**Ressourcen- und Aufgabenverteilung**
Die insgesamt zur Verfügung stehenden Ressourcen wurden auf 11 Hosts verteilt, diese hatten die folgenden Aufgaben:

- *Jump Host*

  Für den Jump Host wurden die wenigsten Ressourcen reserviert. Dessen Aufgabe bestand darin, einen Einstiegspunkt für Verbindungen aus dem Internet bereitzustellen. Jeglicher Zugriff auf das Cluster war aus Sicherheitsgründen nur über den Jump Host möglich.

- *Edge Node*

  Der Edge Node[3] diente als Schnittstelle zwischen Hadoop und dem Anwender. Auf ihm waren alle Gateway-Services für alle verwendeten Hadoop-Komponenten installiert. Von hier werden sämtliche Arbeiten an die jeweiligen anderen Nodes weitergegeben.

- *Master Nodes (3x)*

  Es wurden drei Master Nodes angelegt, diese beinhalteten die Manager für die einzelnen Services. Unter anderem die folgenden:

  - Cloudera Manager: Management und Analyse sämtlicher anderer Komponenten innerhalb des Clusters
  - NameNode: Verwaltet sämtliche Daten auf dem HDFS
  - Resource Manager: Verplant Zeit und Ressourcen für Aufgaben (YARN)
  - Journal Node: Synchronisiert Änderungen im HDFS zwischen NameNode und SecondNode
  - ...

[1] https://www.terraform.io/
[2] https://www.ansible.com/
[3] Manchmale auch *Gateway Node* genannt

- *Worker Nodes (6x)*

  Die Worker Nodes waren für die Speicherung sämtlicher Daten im HDFS und für sämtliche auf diesen durchgeführten Arbeiten verantwortlich. Sie führten letztendlich die relevante Arbeit aus und reservieren die meisten Ressourcen. Für jeden dieser Arbeiterknoten waren 8vCPUs und 16GB an Arbeitsspeicher reserviert, ein kleiner Teil hiervon musste für das Betriebssystem abgegeben werden. Insgesamt standen also ∼48vCPUs und ∼96GB an RAM für die Analysen zur Verfügung.

### 4.1.3 Performanz Metriken

Um die Leistungen der beiden Ansätze untereinander und miteinander vergleichen zu können, wurden die folgenden drei Metriken festgelegt:

- *Verarbeitungszeit*

  Die Verarbeitungszeit ist die simpelste der drei Metriken. Es wird gemessen, wie lang eine Implementierung für die Verarbeitung einer Anfrage benötigt. Startpunkt ist das Senden einer Anfrage, Endpunkt ist die Antwort mit den Ergebnissen. Die Messung dieser Zeit wird automatisch durch Impala übernommen. Bei der Durchführung einer Anfrage muss nur die Zeit aus dem zurückkommenden Ergebnis gefiltert werden. Die Verarbeitung jeder Anfrage kann unter anderem in die folgenden Teile aufgebrochen werden: Kompilation der Anfrage (Optimierungen, Erstellung des Ablaufplans, ...), Ausführung der Anfrage und Ausgabe der Ergebnisse. Zur Vereinfachung werden all diese Zeiten für die Verarbeitungszeit zusammengefasst.

- *Stabilität*

  Mit Stabilität ist an dieser Stelle die Stabilität der Verarbeitungszeiten gemeint. Je stabiler diese Zeiten sind, desto verlässlicher ist das gesamte System. Hierbei werden die folgenden zwei Arten von Stabilität unterschieden:

  – Über eine einzelne Anfrage

    Das System verhält sich stabil, wenn ein und dieselbe Anfrage bei mehreren Durchläufen immer ungefähr die selbe Zeit zum verarbeiten benötigt.

  – Über unterschiedliche Anfragen hinweg

    Das System verhält sich stabil, wenn ähnliche Anfragen ähnlich viel Zeit zum verarbeiten benötigen. Nachfolgend ein Beispiel für ein instabiles System nach diesem Kriterium: Eine SQL-Abfrage benötigt bei einem Gigabyte an Daten 3 Sekunden, bei 10GB 300 Sekunden und bei 25GB an Daten 60 Sekunden. Stabilität ist durch den Ausreißer bei 10GB an Daten nicht vorhanden.

- *Skalierung*

  Die letzte untersuchte Metrik ist die Skalierung. Mit dieser wird gemessen, wie gut ein System mit steigenden Datenmengen umgehen kann. Da vor allem Data Warehouses enorme Mengen an Daten verarbeiten müssen, ist dieser

Punkt relevant. Konkret wird hierbei gemessen, wie sich die Verarbeitungszeit verändert wenn die Datenmengen erhöht werden. Wenn zum Beispiel die Daten in den Quellsystemen verdoppelt werden und eine Anfrage hierdurch doppelt so lange zur Verarbeitung benötigt, dann skaliert das System linear.

### 4.1.4 Grenzen der Untersuchungen

Die nachfolgenden Ergebnisse und Schlussfolgerungen sind nur unter den gegebenen Rahmenbedingungen gültig. Diese werden nachfolgend kurz erläutert.

**Traditioneller Ansatz mit modernen Technologien**
Die neuesten Entwicklungen im Bereich Hadoop erlauben mittlerweile eine Echtzeitverarbeitung von riesigen Datenbeständen. Zusätzlich werden Analysen durch eine SQL-Syntax und die Anbindung von Analyse-Tools ermöglicht. Trotzdem kann der verwendete Technologie-Stack nicht als traditionell bezeichnet werden. Die umgesetzte Architektur (Stern-Schema) ist traditionell, die zur Umsetzung verwendeten Technologien aber nicht. Eine wirklich traditionelle Implementierung würde im Kern eine relationale Datenbank oder ein multidimensionales Array beinhalten. Die gefundenen Ergebnisse sind somit hauptsächlich in dem nachfolgend erwähnten Technologiestack gültig. Die Ergebnisse lassen sich aber auch mehr oder weniger auf andere verteilte SQL-Query-Engines übertragen und sind auch außerhalb von Hadoop teilweise gültig. Der durch die Virtualisierung hinzukommende Aufwand ist nämlich unabhängig von der Query-Engine. Die Art und Weise, wie eine Query-Engine mit einer Anfrage umgeht, könnte sich jedoch unterscheiden und damit die Ergebnisse beeinflussen.

**Technologiestack**
Die durchgeführten Tests wurden innerhalb einer Cloudera-Hadoop Umgebung in einer Cloud-Umgebung durchgeführt. Die im Kern verwendete SQL-Query-Engine ist Apache Impala. Diese Query-Engine ist ein relevanter Faktor, der Auswirkungen auf die Ergebnisse haben kann.

Jede Anfrage durchläuft diese Query-Engine. Es wird ein Ablaufplan erstellt, es wird optimiert und letztendlich werden die Berechnungen durchgeführt. Die Art und Weise, wie diese Query-Engine implementiert wurde, hat somit Auswirkungen auf die Laufzeiten der Anfragen. Somit sind die Ergebnisse mehr oder weniger an diese Query-Engine und Umgebung gebunden. Die Daten wurden im HDFS in einem spaltenorientierten Format komprimiert zwischengespeichert. Das verwendete Format ist Parquet und die verwendete Kompression ist Snappy.

**Verhältnis: Rechenressourcen zu Rechenaufwand**
Ein weiterer relevanter Faktor ist das Verhältnis von vorhanden Rechenressourcen zu der zu verarbeitenden Datenmenge. Zum Beispiel wäre es möglich, dass bei einer Einschränkung der Ressourcen die Kluft zwischen dem traditionellen und virtuellem Ansatz bereits bei den verwendeten Datenmengen größer geworden wäre. Das virtuelle Data Warehouse benötigt für die selben Anfragen eine deutlich höhere Menge an Ressourcen, weil wesentlich mehr Berechnungen durchgeführt werden müssen. In Abschnitt 4.1.2 wird das verwendete Testsystem detaillierter beschrieben.

Zuletzt haben auch Parameter wie die verwendete Anzahl und die Diversität der Quellsysteme, der Umfang der ETL-Prozesse, die Partitionierung und die verwendete Architektur für die Data Warehouses einen Einfluss auf die Ergebnisse. Diese Parameter wurden variiert um deren Auswirkungen zu untersuchen. Die erzielten Ergebnisse sind innerhalb der untersuchten Bereiche gültig, darüber hinaus könnten Abweichungen auftreten.

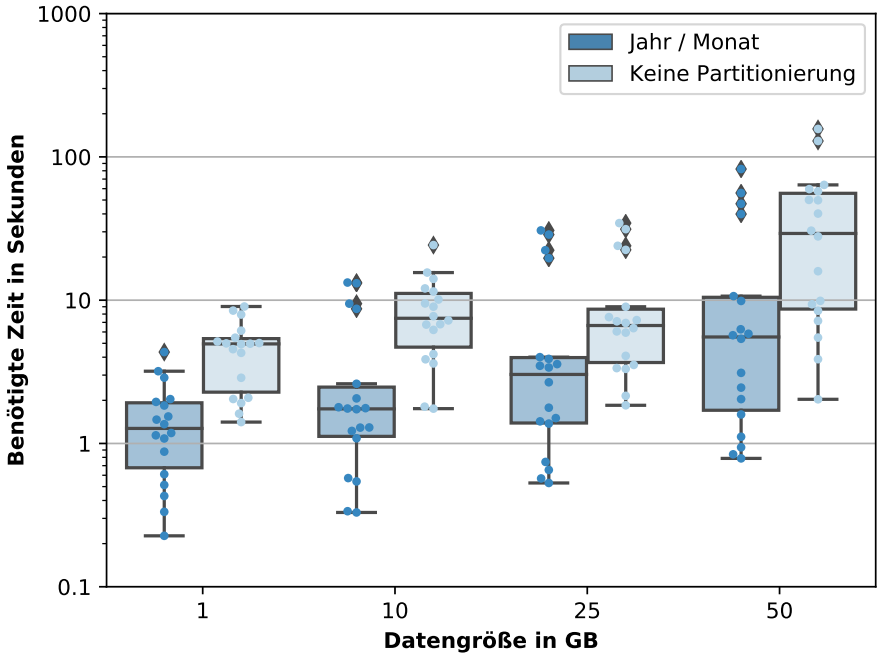



Abbildung 4.1: Anfragen auf dem traditionellen Data Warehouse

## 4.2 Traditionelles Data Warehouse

In dem nachfolgenden Abschnitt werden die Testergebnisse zu dem traditionellen Data Warehouse ausgewertet. Es wird zunächst geklärt, wie gut es sich bei dem gegebenen Use-Case behaupten kann und ob es bei steigenden Datenmengen mitskaliert. Anschließend werden die Einflüsse der Parameter auf die Leistung des Data Warehouses analysiert.

### 4.2.1 Generelle Trends und Ergebnisse

In Abbildung 4.1 werden sämtliche vom traditionellen Data Warehouse durchgeführten Anfragen in einer Abbildung übersichtlich zusammengefasst. Diese werden auf der X-Achse nach verarbeiteter Datenmenge und verwendeter Partitionierung unterteilt und jeweils in einem BoxPlot dargestellt. Zu jeder Anfrage wird auf der Y-Achse die benötigte Zeit zur Verarbeitung in Sekunden angegeben.

Auf den ersten Blick kann man erkennen, dass ein großer Unterschied zwischen den partitionierten und unpartitionierten Implementierungen besteht. Es hat sich herausgestellt, dass die unpartitionierten Lösungen sowohl beim traditionellen als auch beim virtuellen Ansatz eine äußerst schlechte Leistung liefern. Deswegen werden die unpartitionierten Lösungen bei den meisten Ergebnissen und generellen Vergleichen außen vorgelassen. Es wird nur in den jeweiligen Unterabschnitten und wenn explizit darauf hingewiesen wird auf die unpartitionierten Lösungen eingegangen. Zusätzlich fällt auf, dass die Varianz in der Abbildung sehr hoch ist. Dies liegt daran, dass alle Anfragen für eine Implementierung eines Data Warehouses in einem Boxplot zusammengefasst werden. Diese Anfragen unterscheiden sich teilweise sehr in ihrem Aufwand. Die Abbildung ist dafür da, generelle Trends zu verdeutlichen. Eine

| | 1GB | 10GB (+900%) | 25GB (+150%) | 50GB (+100%) |
|---|---|---|---|---|
| Jahr / Monat | 1,27 | 1,74 (+37%) | 3,03 (+74%) | 5,53 (+82%) |
| Keine Partitionierung | 4,95 | 7,47 (+51%) | 6,65 (-11%) | 29,21 (+439%) |

Tabelle 4.1: Median der Verarbeitungszeiten bei verschiedenen Datenmengen

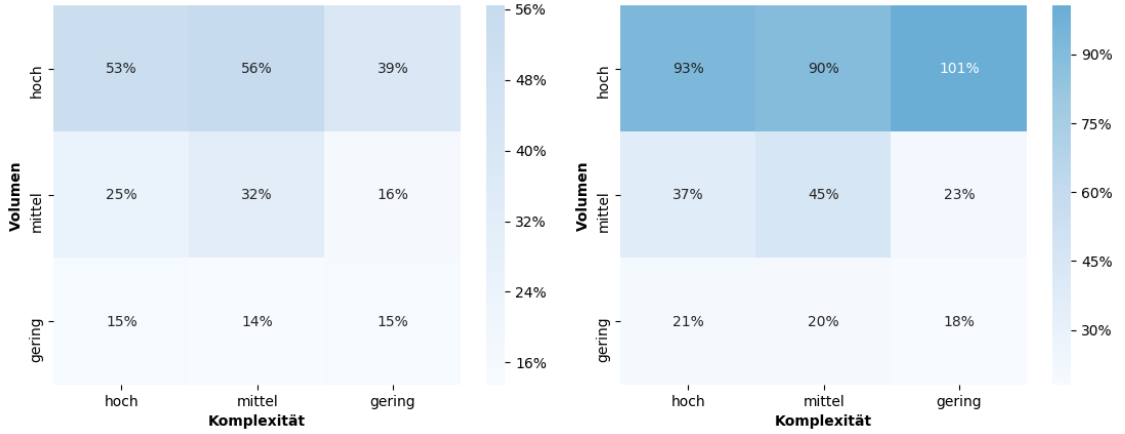
genauere Betrachtung der Tests mit Berücksichtigung der Parameter findet in den folgenden Unterabschnitten statt.

Generell ist die Leistung des traditionellen Data Warehouses sehr gut, sogar die aufwändigste aller Anfragen wird in unter 1,5 Minuten abgearbeitet. Diese beinhaltet einen Scan über 50GB an Faktendaten, welche mit allen Dimensionstabellen verbunden werden müssen. Anschließend werden die Daten nach Produkten gruppiert, summiert und sortiert. Erstaunlich ist auch, dass es bei allen Datenmengen Anfragen gibt, die in weniger als einer Sekunde abgearbeitet werden können. Dies sind bei allen Tests Anfragen, die sehr stark selektieren (geringes Volumen) und keine Verbunde anwenden (geringe Komplexität). Dies ist trotzdem bemerkenswert, weil die selektierten Daten zunächst aus der gesamten Datenmenge gefiltert werden müssen. Hierbei ist die Partitionierung vor allem bei großen Datenmengen essentiell, wie man in der Abbildung sehen kann. Bei 50GB an Daten erhöht sich der Median der Anfragezeit aller Anfragen von 5,53 Sekunden auf 29,21 Sekunden, wenn die Partitionierung nach Jahr und Monat weggelassen wird.

Bis auf einige Ausreißer können alle Anfragen vom partitionierten Data Warehouse in unter 10 Sekunden verarbeitet werden. Bei unter 50GB sogar in unter fünf Sekunden. Dies ist durch die optimierte Speicherung der Daten im Sternschema möglich.

**Ausreißer**
Die Ausreißer, die zu sehen sind, sind alles Anfragen die nicht selektieren (hohes Volumen) und die Daten mit allen Dimensionstabellen verbinden (hohe / mittlere Komplexität). Für solche Anfragen benötigt diese Implementierung bis zu 1,5 Minuten. Solche Anfragen finden in der Praxis jedoch eher selten Anwendung. Die vier Ausreißer (für jede Datengröße) mit besonders guten Zeiten bestehen aus Anfragen, die die Datenmengen filtern (geringes oder mittleres Volumen) und eine geringe Komplexität aufweisen (keine Verbunde). In solchen Fällen sind bei allen Datenmengen Laufzeiten in unter einer Sekunde möglich. Sobald Verbunde erzeugt werden steigt auch bei hoch selektierenden Anfragen die Laufzeit deutlich an.

**Skalierung**
Ein Überblick über die Skalierungen der beiden Implementierungen bei steigenden Datenmengen wird in Tabelle 4.1 gegeben. Die Tabelle stellt für beide traditionellen Implementierungen (partitioniert und unpartitioniert) den Median der Anfragezeiten für alle Datenmengen dar. In jeder Zelle wird in Klammern angeben, wie sich der Wert verglichen mit der vorherigen Spalte verändert hat. Zum Beispiel besagt die Zelle in der Zeile Jahr/Monat und der Spalte 10GB, dass das Data Warehouse im Mittel 1,74 Sekunden gebraucht hat um eine Anfrage mit diesen Parametern zu verarbeiten. Die Verarbeitungszeit hat sich verglichen mit der vorherigen Spalte um 37% verlängert, die Datenmenge hat sich jedoch um 900% erhöht. Bei der Erhöhung der Datenmengen von 1GB auf 10GB konnte das traditionelle Data Warehouse somit äußerst gut skalieren.




(a) 3 Quellen (b) 10 Quellen

Abbildung 4.2: Relative Performanz nach Einführung der Partitionierung

Generell kann das partitionierte traditionelle Data Warehouse bei der gegebenen Systemspezifikation und den gegebenen Datenmengen noch sehr gut skalieren. Man sieht aber, dass bei steigenden Datenmengen diese sich der linearen Skalierung nähert. Bei einer perfekten linearen Skalierung würde sich die Laufzeit um das doppelte erhöhen, wenn die Datenmenge verdoppelt wird. Dies ist beim Sprung von 25GB auf 50GB an Daten beinahe der Fall (Datenmenge +100%; Laufzeit +82%). Bei noch größeren Datenmengen wird eine lineare Skalierung oder noch schlechter erwartet. Was in der Tabelle nicht zu sehen ist, bei einem genaueren Blick in die Daten jedoch auffällt, ist dass das Data Warehouse bei hohem Anfragevolumen (Full Table Scan) bereits linear Skaliert (∼105%). Dieser Wert ist bei allen Komplexitäten (Hoch / Mittel / Gering) gleich.

Das unpartitionierte Data Warehouse weißt Sprünge auf, die bisher noch nicht geklärt werden konnten. Zum Beispiel verfünffacht sich der Aufwand beim Übergang von 25GB an Daten auf 50GB an Daten während sich der Aufwand sogar reduziert, wenn die Datenmengen von 10GB auf 25GB erhöht werden.

### 4.2.2 Einflüsse durch die Partitionierung

Partitionierung ist eine Optimierung, bei der die Daten so strukturiert werden, dass bei selektierenden Anfragen nicht alle Daten gelesen werden müssen. Wenn eine Tabelle beispielsweise nach Jahr und Monat partitioniert wurde und eine Anfrage gestellt wird, die nur die Daten aus dem Jahre 2017 benötigt, dann müssen auch nur diese Daten gelesen werden. Alle anderen Jahre können direkt ausgeschlossen werden.

In Abbildung 4.2 wird der Einfluss der Aktivierung einer Partitionierung nach Jahr und Monat auf die Verarbeitungszeiten dargestellt. Die beiden Abbildungen stellen diese Einflüsse einmal für ein traditionelles Data Warehouse mit drei und mit zehn Quellsystemen dar. Der Einfluss, den die Partitionierung hat, hängt sehr von der durchzuführenden Anfrage ab. Aus diesem Grunde wurde für die Darstellung eine Heatmap verwendet. Auf der X-Achse werden die Anfragen nach ihrer Komplexität

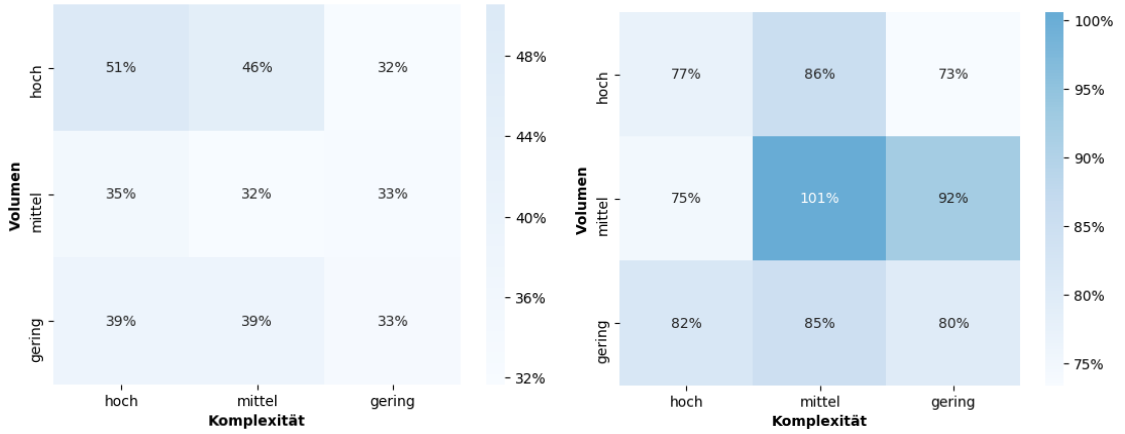
und auf der Y-Achse nach ihrem Volumen aufgeteilt. Die Zahl in einer Zelle gibt in Prozent an, wie sich die Verarbeitungszeit nach der Einführung der Partitionierung verändert hat. Zum Beispiel besagt die Zelle (geringes Volumen, hohe Komplexität) in 4.2(a), dass die Anfragen in nur 15% der benötigten Zeit der unpartitionierten Lösung bearbeitet wurden. Die Laufzeit könnte zum Beispiel von 5 Sekunden auf 0,75 Sekunden gesunken sein. Die Werte innerhalb einer Zelle wurden über alle Datenmengen (1GB bis 50GB) aggregiert, eine feinere Darstellung ist im Appendix in Abschnitt A.3 zu sehen.

Man kann direkt erkennen, dass die Partitionierung in diesem Fall einen sehr positiven Einfluss auf die Verarbeitungszeiten hat. Alle durchgeführten Anfragen profitierten mehr oder weniger davon. Dies passt mit den in Abbildung 4.1 zu sehenden Trends überein. Sowohl die Performanz, als auch die Konsistenz der Anfrageverarbeitung verbessern sich. Die Auswirkungen der Partitionierung sind teilweise enorm, abhängig von den Anfragen kann eine Verzehnfachung der Verarbeitungsgeschwindigkeit erreicht werden. Die Hauptursache hierfür sind die folgenden zwei Punkte:

**Reduzierung des Leseaufwandes:**
Dies ist die offensichtliche Ursache, die die Laufzeiten bei mittlerem und geringem Volumen verbessert. Diese Anfragen selektieren die Daten deutlich (8% bei mittlerem Volumen und 0,02% bei geringem Volumen) und durch die Partitionierung kann direkt ein großer Teil der Daten ausgeschlossen werden. Dies erklärt den in der Abbildung sichtbaren Stufeneffekt, der sich vertikal von oben nach unten abzeichnet. Je restriktiver die Selektionen, desto besser ist das partitionierte Data Warehouse verglichen mit dem unpartitionierten. Erstaunlich ist jedoch, dass die Partitionierung bei geringem Volumen nochmal deutlich besser wirkt als bei mittlerem Volumen. Beim geringen Volumen wird die Selektion nämlich durch eine unpartitionierte Dimension erweitert (Kunde).

**Erhöhte Parallelisierung:**
Ein weiterer unerwarteter Effekt ist die Verbesserung der Leistung bei hohem Volumen, schließlich müssen in diesem Fall alle Daten gelesen werden. Hierdurch kann die Partitionierung den Leseaufwand nicht reduzieren. Dieser Effekt lässt sich durch eine erhöhte Parallelisierung erklären. Durch die Partitionierung werden die Daten auf der Festplatte in kleinere Dateien aufgespalten. Dies führt dazu, dass die Größenunterschiede zwischen Dateien kleiner sind und allgemein wesentlich mehr Dateien vorliegen. Hierdurch kann Impala konsistenter und mehr parallelisieren (bis zu einem Grade).

Ein weiteres Merkmal in Abbildung 4.2 unterstützt diese Annahme. Man kann erkennen, dass die Partitionierung bei dem Data Warehouse mit nur drei Quellen deutlich besser hilft als bei dem Data Warehouse mit 10 Quellen, vor allem bei Anfragen mit hohem Volumen. Das einzige, was diese beiden Data Warehouses unterscheidet, ist eine feinere Aufteilung der Dateien der Faktentabelle bei zehn Quellen. Die Integration der Quellen läuft während des ETL-Prozesses folgendermaßen ab: Als erstes wird die erste Quelle in die finale Faktentabelle geladen. Anschließend werden alle anderen Quellen Schritt für Schritt in die finale Faktentabelle integriert. Diese Integration findet letztendlich durch das anfügen von einer oder mehreren Dateien im Ordner der Faktentabelle statt. Hierdurch ergibt sich eine feinere Aufteilung




(a) Unpartitioniert (b) Partitioniert Jahr / Monat

Abbildung 4.3: Relative Performanz nach Erhöhung der Quellanzahl

der Tabellendateien bei vielen Quellsystemen und Integrationsschritten. Das Ergebnis ist also, dass bei nur drei Quellsystemen die Faktentabelle weniger feingranular gespeichert wird als bei zehn Quellen. Die Partitionierung nach Jahr und Monat dürfte hier demnach einen größeren Effekt erzielen, was durch die Abbildung bestätigt wird. Bei zehn Quellsystemen hat die Partitionierung bei hohem Volumen kaum einen Einfluss (88% bis 94%).

Zum Schluss fällt noch auf, dass die Komplexität der Anfragen den Einfluss der Partitionierung kaum verändert. Dies macht in dieser Implementierung auch Sinn, die Partitionierung hilft hier lediglich dabei die benötigten Daten effizient zu laden. Es kann jedoch Anwendungsfälle geben, bei denen durch die Partitionierung die Durchführung von Verbunden beschleunigt wird. Wenn zwei Tabellen zum Beispiel anhand ihres Verbundschlüssels partitioniert sind, dann können die zueinander passenden Tupel durch die Partitionierung effizienter gefunden werden.

### 4.2.3 Einflüsse durch die Anzahl der Quellen

An dieser Stelle wird untersucht, welchen Einfluss die Anzahl der Quellen auf die Verarbeitungsgeschwindigkeit hat. Um die Ergebnisse zu präsentieren wird eine ähnliche Darstellung wie bereits bei der Partitionierung verwendet. In Abbildung 4.3 wird in Prozent angegeben, wie sich die Laufzeit bei der Erhöhung der Anzahl der Quellen von drei auf zehn Quellen verändert. Einmal auf einem unpartitionierten und einmal auf einem partitionierten Data Warehouse. Wenn der Wert einer Zelle zum Beispiel 33% ist, dann wurde diese Anfrage vom Data Warehouse mit zehn Quellen dreimal schneller verarbeitet als vom Data Warehouse mit drei Quellen.

Eine Annahme, die vor der Evaluierung getroffen wurde, war dass es beim traditionellen Data Warehouse keinen Unterschied machen sollte, wie viele Quellen vorhanden sind. Durch den ETL-Prozess müssten alle Unterschiede beglichen werden und am Ende ein Star-Schema vorliegen, welches unabhängig von der Anzahl der Quellen eine gute Leistung erbringt. Dadurch, dass die Menge und der Inhalt der Daten bei 3 / 10 Quellen identisch ist (nur die Verteilung ist anders), wurde erwartetet, dass die resultierenden Data Warehouse nahezu identische Leistungen liefern.

Umso erstaunlicher waren dann die Ergebnisse der Tests. Die Anzahl der Quellen macht bei dieser Implementierung doch einen Unterschied, der Grund hierfür liegt in der Struktur der Dateien, die die Tabellendaten repräsentieren. Die Daten für die Tabellen liegen im HDFS. Das bedeutet, dass Daten nicht mehr geändert werden können, nachdem sie ausgeschrieben wurden. Wenn nun neue Daten an die Faktentabelle durch die Integration von weiteren Quellen angehängt werden sollen, dann geschieht dies durch das Anhängen von neuen Dateien im HDFS-Ordner dieser Tabelle. Dadurch, dass dies beim Data Warehouse mit zehn Quellen häufiger passiert, ergibt sich auch eine feinere Aufteilung der Daten in kleinere Dateien. Diese feinere Aufteilung ist für die Leistungsunterschiede die in Abbildung 4.3 zu sehen sind, verantwortlich.

Bei der Betrachtung von 4.3(a) ist ein enormer Leistungsgewinn durch die Erhöhung der Quellsysteme zu sehen. Hierbei ist wichtig zu erwähnen, dass diese Grafik eine Aggregation über die Ergebnisse aller Datenmengen darstellt. Bei einer genaueren Betrachtung fällt auf, dass dieser Vorteil fast ausschließlich bei dem größten Datenset (50GB) auftritt, hier jedoch ziemlich intensiv. Bei den kleineren Datensets ist, wenn überhaupt, nur ein kleiner Vorteil zu sehen. Diese unaggregierten Daten können in Abschnitt A.2 eingesehen werden. Dies ist auch logisch, eine feinere Aufteilung der Dateien führt nur bis zu einem gewissen Grade zu einer höheren Parallelisierung. Bei einer zu tiefen Aufteilung erhöht sich der Aufwand durch das Öffnen / Schließen der vielen kleinen Dateien, was wiederum die Leistung senkt. Daraus ergibt sich, dass eine feinere Aufteilung der Dateien vor allem bei sehr großen Datensets zu einer Leistungserhöhung führen kann.

Ähnlich sieht es mit 4.3(b) aus, diese stellt dieselben Unterschiede beim partitioniertem Data Warehouse dar. Wie erwartet, sind die Verbesserungen der Leistung an dieser Stelle geringer als beim unpartitioniertem Data Warehouse. Das liegt daran, dass durch die Partitionierung bereits eine tiefere Aufspaltung der Dateien stattgefunden hat. Eine weitere Aufteilung erbringt wie in der Abbildung zu sehen einen geringeren Nutzen. Es konnte kein erwähnenswerter Einfluss von Parametern wie Volumen / Komplexität der Anfragen festgestellt werden

### 4.2.4 Einflüsse durch die Arten von Anfragen

Nachfolgend werden die Auswirkungen des Anfragevolumens und der Anfragekomplexität auf die Leistung betrachtet. Hierbei liegt der Fokus auf dem traditionellen und partitionierten Data Warehouse, diese Auswirkungen werden in Abbildung 4.4 zusammengefasst. Diese Abbildung stellt für jede Kombination von Anfragevolumen und -komplexität einen Boxplot dar. Jeder dieser Boxplots fasst die Daten von acht Anfragen zusammen. Diese Anfragen beinhalten die unterschiedlichen Datenmengen (1GB, 10GB, 25GB und 50GB) für jeweils drei und zehn Quellsysteme (4 * 2 = 8 verschiedene Anfragen).

Auf den ersten Blick ist eine sehr hohe Varianz zu erkennen, beim aller ersten Boxplot (ganz links) liegen das Minimum und das Maximum bei ∼3 und ∼80 Sekunden. Das liegt daran, dass diese Plots für die jeweilige Anfrage (hier hohes Volumen und hohe Komplexität) die Daten für alle Datenmengen zusammenfassen. Auf den zweiten Blick fällt jedoch auf, dass diese Varianz sehr von dem Anfragevolumen abzuhängen

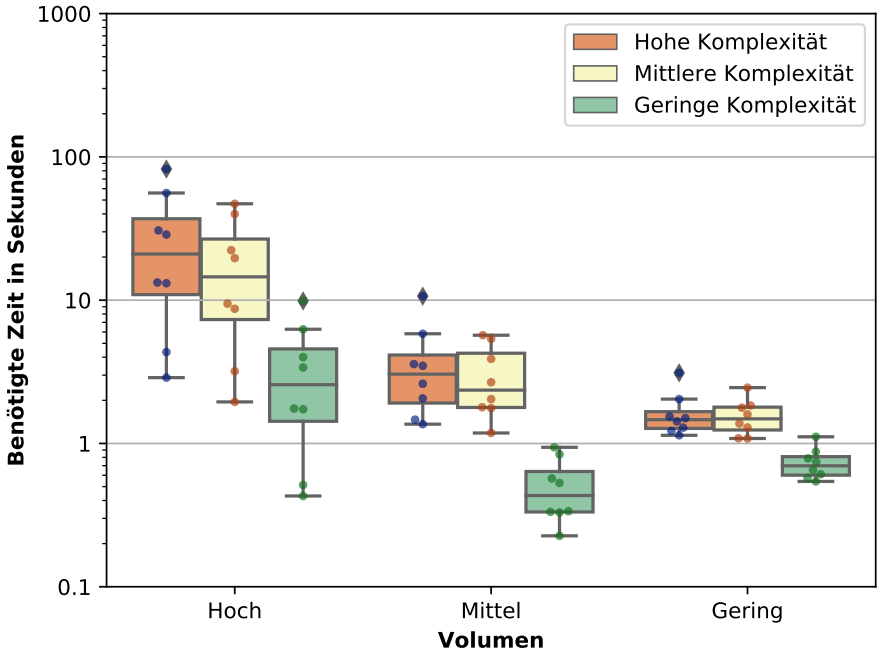



Abbildung 4.4: Einfluss des Anfragevolumens und der Anfragekomplexität auf die Leistungen des partitionierten traditionellen Data Warehouses

scheint. Bei mittlerem Volumen ist diese viel geringer und bei geringem Volumen kaum noch vorhanden. Daraus kann man schließen, dass das Datenvolumen abhängig von dem Anfragevolumen (mit anderen Worten, abhängig von der Selektivität einer Anfrage) einen unterschiedlichen Einfluss auf die Verarbeitungszeit hat. Zum Beispiel liegt bei einer Anfrage mit geringem Volumen und einer hohen Komplexität die Differenz der Laufzeit zwischen einem 1GB / 50GB Datenset bei nur ∼2 Sekunden. Bei hohem Anfragevolumen liegt diese Differenz bei ∼77 Sekunden. Das bedeutet also, dass bei Anfragen die strenger selektieren, die Datenmengen eine wesentlich geringere Rolle spielen. Sehr selektive Anfragen können auch auf enormen Datenmengen sehr effizient verarbeitet werden, zumindest wenn wie in diesem Fall eine Partitionierung durchgeführt wurde. Das Data Warehouse skaliert bei selektiven Anfragen also wesentlich besser.

Nicht nur die Varianz nimmt bei selektierenden Anfragen ab, allgemein ist eine wesentlich bessere Performanz bei mittlerem und geringem Volumen zu sehen. Vor allem der Sprung von hohem zu mittlerem Volumen ist enorm. Dies lässt sich leicht durch die Partitionierung erklären. Bei mittlerem Volumen wird eine Selektion auf der Zeit-Dimension durchgeführt, dies ist die Dimension, nach der die Faktentabelle partitioniert ist. Die Leistungsverbesserung beim Übergang vom mittlerem zu geringem Volumen ist viel geringer. Die Selektion wird zwar restriktiver, diese wird aber auf einer anderen Dimension durchgeführt nach der das Data Warehouse nicht partitioniert ist. Zusammengefasst lässt sich schließen, dass das Anfragevolumen einen enormen Einfluss auf die Verarbeitungszeit und auf die Skalierung bei unterschiedlich großen Datenmengen hat.

An dieser Stelle wird ein Blick auf die Auswirkungen der Anfragekomplexität geworfen. Auffällig ist, dass die Abweichung zwischen hoher und mittlerer Komplexität

bei allen Anfragevolumen sehr gering ist. Der Unterschied zwischen mittlerer und geringer Komplexität ist jedoch enorm. Die Unterschiede zwischen diesen drei Arten von Anfragen wurden bereits erläutert, hier nur eine kurze Zusammenfassung:

- Gering: Es wird nur eine Aggregation über alle zu verarbeitenden Daten durchgeführt
- Mittel: Zusätzlich zur Aggregation wird die Faktentabelle mit allen vorhandenen Dimensionstabellen verbunden
- Hoch: Die Daten werden zusätzlich nach dem Produktnamen gruppiert und die aggregierten Fakten nach dem Profit sortiert

Das Ergebnis ist also, dass die durchzuführenden Verbunde zwischen den Tabellen mit einem sehr hohem Aufwand verbunden sind. Ein Gruppieren oder Sortieren von Daten wurde an dieser Stelle effizient durchgeführt. Erwähnenswert ist noch, dass das "ORDER BY" nur auf die bereits gruppierten Daten angewendet wurde, die Menge an zu sortierenden Daten ist also minimal. Ein "ORDER BY" auf nicht gruppierten großen Datenmengen würde wahrscheinlich in anderen Ergebnissen resultieren.

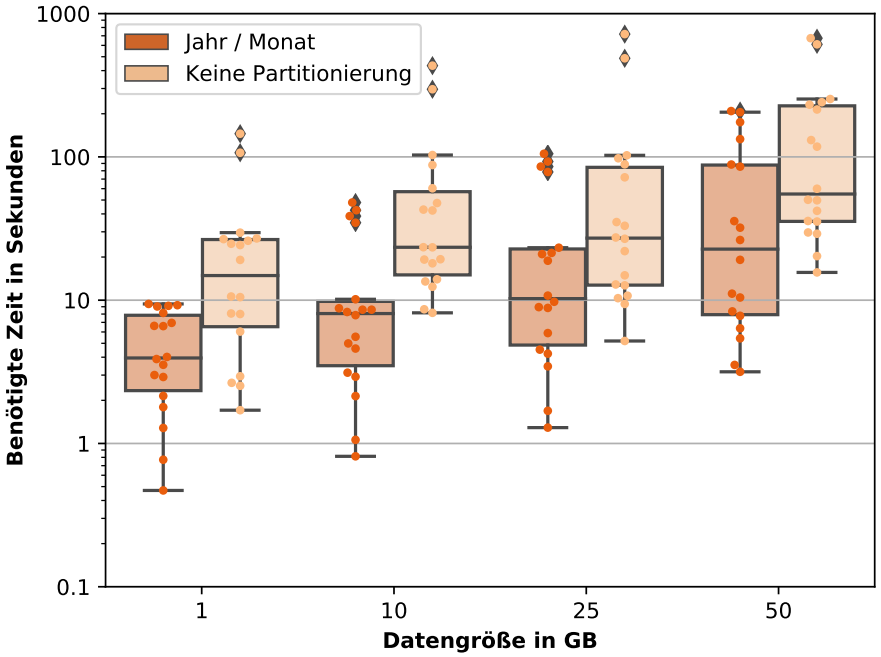



Abbildung 4.5: Anfragen auf dem virtuellen Data Warehouse

## 4.3 Virtuelles Data Warehouse

Nachfolgend werden die Testergebnisse zu dem virtuellen Data Warehouse ausgewertet. Der Aufbau des Abschnitts ist identisch zu dem vorherigen.

### 4.3.1 Generelle Trends und Ergebnisse

Wie bereits beim traditionellen Data Warehouse wird durch Abbildung 4.5 die Leistung vom virtuellen Data Warehouse zusammenfassend dargestellt. Die Verläufe sehen auf den ersten Blick ziemlich ähnlich aus, verglichen mit dem traditionellen Ansatz. Es gibt ähnliche Ausreißer, der Unterschied zwischen partitioniertem und unpartitioniertem Data Warehouse ist ähnlich groß, und sie scheinen ähnlich gut zu skalieren. Der Hauptunterschied liegt darin, dass die Laufzeiten aller Anfragen nach oben verschoben wurden, sie benötigen also wie erwartet länger zum Verarbeiten. Auch hier werden die Analysen hauptsächlich auf den deutlich besseren Ergebnissen der partitionierten Data Warehouses durchgeführt. Auf die Unterschiede zwischen den partitionierten / unpartitionierten Ansätzen wird wie schon beim traditionellen Data Warehouse in dem entsprechenden Unterabschnitten eingegangen.

Generell ist auch die Leistung vom virtuellen Data Warehouse gut, man muss aber bei ausnahmslos allen Anfragen länger warten. Die aufwändigste aller Anfragen wurde in maximal 3 Minuten und 20 Sekunden abgeschlossen, was einer Verdopplung der Verarbeitungszeit verglichen mit dem traditionellen Data Warehouse entspricht. Die einfachsten Anfragen (geringes Volumen und Komplexität) werden auch vom virtuellen Ansatz äußerst schnell verarbeitet, hier ist jedoch ein größerer Unterschied zum traditionellen Ansatz zu sehen. Die Verarbeitungszeiten liegen hier im Sekundenbereich (bis zu 1,5 Sekunden bei bis zu 25GB; 5 Sekunden bei 50GB).

| | 1GB | 10GB (+900%) | 25GB (+150%) | 50GB (+100%) |
|---|---|---|---|---|
| Jahr / Monat | 3,95 | 8,06 (+102%) | 10,25 (+26%) | 22,71 (+121%) |
| Keine Partitionierung | 14,85 | 23,4 (+57%) | 27,10 (+15%) | 55,06 (+103%) |

Tabelle 4.2: Median der Verarbeitungszeiten bei verschiedenen Datenmengen

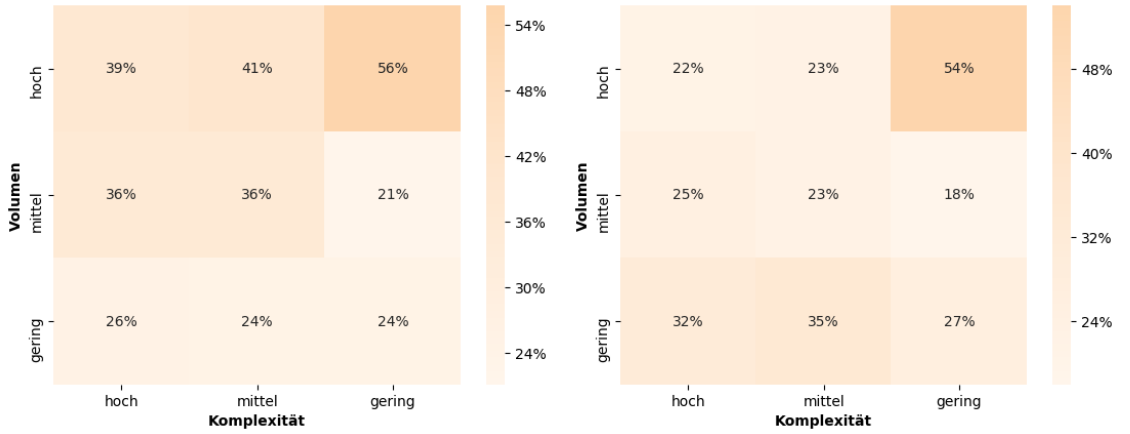
Beim traditionellen Ansatz konnten alle Anfragen dieser Art in unter einer Sekunde abgeschlossen werden.

Generell werden alle Anfragen (bis auf die vier aller aufwendigsten) in unter 100 Sekunden verarbeitet. Wenn man die Ausreißer außen vor nimmt, dann werden die meisten Anfragen in unter 30 Sekunden abgeschlossen (10 Sekunden beim traditionellen Ansatz). Es ist allgemein ein deutlicher und erwarteter Anstieg der Laufzeiten zu verzeichnen.

**Ausreißer**
Die zu sehenden Ausreißer zeigen die selben Muster wie schon beim traditionellen Ansatz. Die Ausreißer nach oben führen keine Selektionen durch und führen alle möglichen Verbunde aus (Es kann also auf alle Spalten zugegriffen werden). Für solche Anfragen werden maximal 3,3 Minuten (∼2x so lang wie beim traditionellen Ansatz) zur Verarbeitung benötigt, diese treten in der Praxis jedoch selten auf. Bei den Ausreißern mit besonders guten Zeiten sind es auch Anfragen, die die Datenmengen selektieren und keine Verbunde durchführen. Auch beim virtuellen Ansatz werden diese Arten von Anfragen in nur wenigen Sekunden verarbeitet, es ist jedoch ein deutlicherer Unterschied zum traditionellen Ansatz zu sehen als bei den Ausreißern nach oben. Auch hier gilt: die Einführung von Verbunden hat einen enormen Einfluss auf die Laufzeiten.

**Skalierung**
Tabelle 4.2 gibt wie schon beim traditionellen Data Warehouse einen Überblick über die Skalierungen der beiden Implementierungen (Partitioniert und Unpartitioniert). Für beide Implementierungen wird der Median der Verarbeitungszeiten bei verschiedenen Datenmengen dargestellt, in Klammern der Anstieg verglichen mit der vorherigen Spalte.

Das partitionierte virtuelle Data Warehouse liefert an dieser Stelle wesentlich bessere Ergebnisse als erwartet. Es skaliert bei Datenmengen bis zu 25GB äußerst gut. Zum Beispiel erhöht sich bei einer Verzehnfachung der Daten von 1GB auf 10GB die Laufzeit nur um das doppelte. Auch bei einer anschließenden Erhöhung der Daten um 150% verschlechtert sich die Laufzeit nur um 26%. Erst beim Sprung auf 50GB wird eine lineare Skalierung erreicht, die etwas schlechter ist als beim traditionellen Ansatz (+82% beim traditionellen Ansatz gegen +121% beim virtuellen). Es scheint so, als wäre der Overhead durch die Virtualisierung bei diesen Implementierungen und Datenmengen hauptsächlich ein konstanter Faktor. Dieser liegt bei allen Datenmengen ungefähr bei dem drei- bis vierfachen der Laufzeit des traditionellen Data Warehouses. Es ist bei diesen Datenmengen kein deutlicher Anstieg dieses Faktors zu sehen, beide Ansätze skalieren bei bis zu 50GB an Daten noch ungefähr gleich gut. Durch den Overhead ist das virtuelle Data Warehouse natürlich immer langsamer. Interessant wäre es an dieser Stelle zu wissen, welchen Einfluss noch größere Datenmengen auf die Skalierungen beider Ansätze hätten.




(a) 3 Quellen (b) 10 Quellen

Abbildung 4.6: Relative Performanz nach Einführung der Partitionierung

Die unpartitionierte Implementierung des virtuellen Ansatzes skaliert bei steigenden Datenmengen erstaunlicherweise nochmal besser als die partitionierte Implementierung. Aufgrund der äußerst schlechten Laufzeiten bei 1GB an Daten bringt die positive Skalierung letztendlich nicht viel, die Laufzeiten bleiben trotzdem bei allen Datenmengen schlecht.

### 4.3.2 Einflüsse durch die Partitionierung

Nachfolgend wird der Einfluss einer Partitionierung nach Jahr und Monat auf die Performanz des virtuellen Data Warehouses erläutert. Wie schon beim traditionellen Ansatz stellt Abbildung 4.6 die Leistung des Data Warehouses nach Einführung der Partitionierung dar, verglichen mit der selben Implementierung ohne Partitionierung.

Es ist wie schon beim traditionellen Data Warehouse ein äußerst positiver Einfluss durch die Partitionierung zu sehen. Alle Arten von Anfragen profitieren hierdurch, sowohl bei drei als auch bei zehn Quellsystemen. Durchschnittlich verbessert sich die Laufzeit um das zwei bis fünffache, die unpartitionierte Implementierung benötigt im Durchschnitt also zwei bis fünfmal so lang zur Verarbeitung der Anfragen.

Äußerst auffällig ist auch, dass der Stufeneffekt, der beim traditionellen Ansatz deutlich zu sehen war, hier kaum zu erkennen ist. Alle Anfragen scheinen ungefähr gleich gut zu profitieren, Volumen und Komplexität der Anfragen haben dieser Abbildung nach kaum einen Effekt auf den Einfluss der Partitionierung. Nur bei drei Quellsystemen kann man einen leichten Stufeneffekt über die Y-Achse erkennen. Dies ist merkwürdig, eigentlich sollte die Implementierung vor allem bei den selektierenden Anfragen (mittleres und geringes Volumen) deutlich von der Partitionierung profitieren und verglichen zum hohen Volumen besser abschneiden. Wie bereits im Unterkapitel zum traditionellen Ansatz erwähnt, stellt diese Abbildung eine Aggregation über mehrere Datenmengen (1GB bis 50GB) dar. Ein Blick auf die unaggregierten Daten erklärt dieses Verhalten, durch einige Ausreißer erscheint der Einfluss der Partitionierung bei hohen Datenmengen positiver als er eigentlich ist. Die unaggregierten Daten werden in Abschnitt A.3 dargestellt.

Quelltext 4.1: Ausschnitt aus der Impala Profile Page (Schritt 09:HASH JOIN)

```
HASH_JOIN_NODE (id=9):(Total: 1m16s, non-child: 8s650ms, % non-child: 11.27%)
- BuildRows: 20.54M (20538566)
[...]
EXCHANGE_NODE (id=97):(Total: 1m7s, non-child: 1m7s, % non-child: 100.00%)
[...]
- TotalBytesDequeued: 1.62 GB (1735426790)
- TotalGetBatchTime: 1m7s
- DataWaitTime: 1m6s
```

Die detaillierteren Abbildungen bestätigen, dass der Stufeneffekt doch existiert. Das virtuelle Data Warehouse profitiert bei selektierenden Anfragen genauso wie das traditionelle Data Warehouse besonders stark. Es ist jedoch noch nicht geklärt, wieso einige der Anfragen bei hohem Volumen so extrem von der Partitionierung profitieren. Das auffällige Verhalten wird an einem der Ausreißer erläutert. Es wird ein genauerer Blick auf die Anfrage mit hohem Volumen, hoher Komplexität und dem virtuellen Data Warehouse mit zehn Quellen auf 25GB an Daten geworfen. Diese Anfrage wird vom unpartitionierten Warehouse in 719 Sekunden verarbeitet, das partitionierte benötigt nur 85 Sekunden. Die partitionierte Implementierung benötigt also wie in der detaillierten Abbildung A.4 dargestellt nur 12% der Zeit, um diese Anfrage zu verarbeiten. Diese unglaubliche Verbesserung durch die Partitionierung lässt sich also durch die sehr schlechte Leistung des unpartitioniertem Data Warehouses erklären.

Um dieses Verhalten zu erklären, wird ein Blick in die Ablaufpläne der Anfragen von Impala geworfen. In Abschnitt A.1 werden diese Pläne für das partitionierte und das unpartitionierte Data Warehouse dargestellt. Aufgrund des Umfangs dieser Pläne, werden an dieser Stelle nur die Kernursachen und deren Identifizierung erläutert. Die Ablaufpläne werden von unten nach oben gelesen, das ist die Reihenfolge in der die einzelnen Phasen und Schritte ausgeführt werden. Jede Zeile repräsentiert eine Phase oder einen Schritt und fasst für diese Phase / diesen Schritt einige Informationen zusammen. Dazu gehören:

- Die Anzahl der Hosts auf denen die Arbeit verteilt wurde
- Die durchschnittlich und maximal benötigten Zeiten auf den Hosts für den Schritt
- Die Anzahl der verarbeiteten Zeilen
- Die Menge an benötigtem Arbeitsspeicher
- Die letzte Spalte gibt die durchgeführten Operationen an und auf welchen Tabellen diese ausgeführt wurden

Als Ursache für die äußerst schlechte Performanz auf dem unpartitioniertem virtuellen Data Warehouse konnte ein schlecht optimierter Ablaufplan identifiziert werden.

Aus bisher nicht erkannten Gründen erstellt der Impala-Query-Optimizer unterschiedliche Pläne für die partitionierte / unpartitionierte Implementierung. Dieser Ablaufplan ist aus den folgenden zwei Gründen sehr ungünstig:

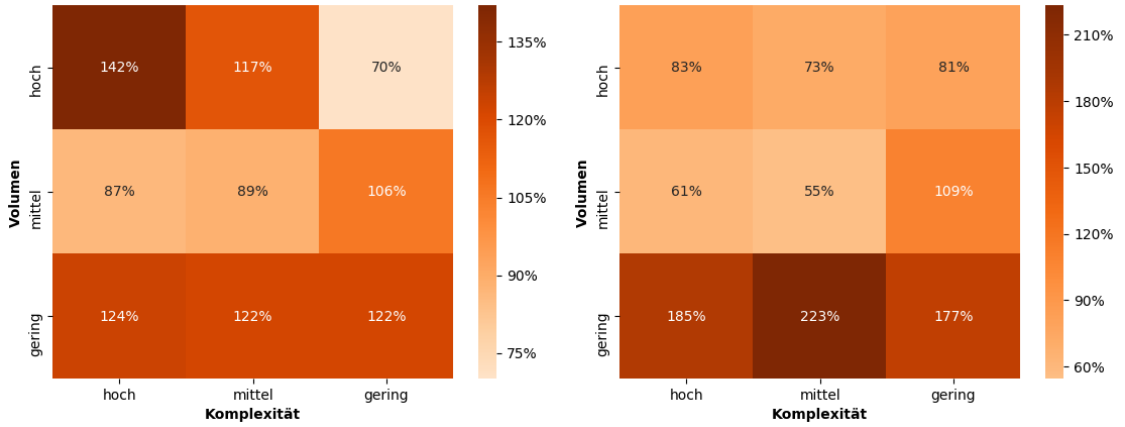
**Viel Netzwerk-Transfer**
Als Hauptursache wurde ein enormer Overhead beim transferieren von Daten über das Netzwerk festgestellt. Um dies zu verdeutlichen, wird die Berechnung der integrierten Tabelle für die erste Quelle an beiden Ablaufplänen erläutert. Diese Berechnungen werden bei beiden Abläufen am Anfang durchgeführt und sind in den beiden Ablaufplänen durch einen gräulichen Hintergrund markiert.

Das unpartitionierte Data Warehouse beginnt diesen Prozess damit, die Daten für die Tabelle *Purchase* in den Speicher zu laden (Schritt 06), anschließend wird die *Customer*-Tabelle geladen (Schritt 07) und mit der *purchase*-Tabelle verbunden (Schritt 08). Als nächstes wird die größte Tabelle geladen (*purchase_item*) und mit den Tabellen *product* und *store* verbunden (Schritte 01 bis 05). Der letzte und aufwändigste Schritt besteht darin, die komplette *purchase_item*-Tabelle mit der *purchase*-Tabelle zu verbinden, hier liegt das Hauptproblem der unpartitionierten Implementierung (Schritt 09). Dadurch, dass die *purchase*-Tabelle als erstes geladen wurde, dient es als Fundament für diesen großen Verbund. Die wesentlich größere *purchase_item*-Tabelle muss also über das Netzwerk an alle Hosts mit den bereits geladenen *purchase*-Tabellen gesendet werden. Hierbei fallen lange Wartezeiten für alle Hosts an.

Leider stellt diese Summary-Tabelle nur die Zeiten für die durchgeführten Operationen dar, I/O-Wartezeiten werden zum Beispiel nicht berücksichtigt. Der Schritt "09:HASH JOIN" benötigt nach der Summary acht Sekunden und 650 Millisekunden zur Berechnung des Verbundes. Ein Blick auf die wesentlich detailliertere Profile-Page von Impala zeigt, dass der Hash-Verbund wie in Quelltext 4.1 dargestellt insgesamt 1m16s zur Verarbeitung benötigte, 8s650ms davon wurden tatsächlich vom Verbund verbraucht. Die restlichen 1m7s wurden vom "EXCHANGE_NODE" für den Transfer von 1,62GB an unkomprimierten Daten verbraucht. Zusammengefasst kann man sagen, dass in diesem Fall fast 90% der Zeit für den Netzwerktransfer von Daten verbraucht wurde. Ein genauerer Einblick in die Profile-Page kann aus Platzgründen nicht gewährt werden. Allein für die gerade erläuterte Anfrage umfasst diese mehr als 35.000 Zeilen.

**Weniger Parallelisierung**
Die zweite entdeckte Ursache ist die geringere Parallelisierung durch Impala. Ein Blick auf die markierten Ablaufpläne in Quelltext 4.1 zeigt, dass alle Verbunde bei der unpartitionierten Implementierung wesentlich langsamer sind als bei der partitionierten (z.B.: 8s650ms vs. 1s962ms bei dem Schritt 09:HASH JOIN). Einer der Gründe hierfür ist eine geringere Parallelisierung durch eine nicht optimale Verteilung der Daten auf den Hosts. Die effiziente partitionierte Implementierung beginnt damit, die Faktentabelle in Schritt "01:SCAN HDFS" auf allen sechs Hosts verteilt in den Speicher zu laden. Damit ist der größte Teil der Daten gleichmäßig auf allen Hosts im Speicher verteilt. Anschließend werden die restlichen benötigten Tabellen Schritt für Schritt geladen und mit der *purchase_item*-Tabelle verbunden.

Die unpartitionierte Implementierung beginnt mit Schritt "06:SCAN HDFS" die wesentlich kleinere *purchase*-Tabelle auf vier Hosts verteilt in den Speicher zu laden. Die




(a) Unpartitioniert (b) Partitioniert Jahr / Monat

Abbildung 4.7: Relative Performanz nach Erhöhung der Quellanzahl

wesentlich größere *purchase_item*-Tabelle wird erst etwas später in Schritt "01:SCAN HDFS" auf nur zwei Hosts verteilt geladen. Hierdurch ist die Verteilung der Daten wesentlich schlechter, es müssen mehr Daten über das Netzwerk transferiert werden und eine optimale Parallelisierung der Arbeit ist nicht möglich. Der Grund, wieso Impala diesen Ablaufplan bei dieser Implementierung so gestaltet, konnte nicht geklärt werden.

Zusammengefasst ergibt sich, dass ohne Partitionierung die Ablaufpläne der Anfragen teilweise äußerst schlecht optimiert werden. Hierdurch können sich, wie in dem Beispiel gezeigt, sehr inkonsistente Verhaltensweisen und äußerst lange Laufzeiten ergeben. Die Ursache hierfür konnte leider nicht gefunden werden. Erstaunlich ist aber, das solch ein Verhalten bei allen partitionierten Anfragen nicht gefunden werden konnten.

### 4.3.3 Einflüsse durch die Anzahl der Quellen

An dieser Stelle wird der Einfluss der Anzahl der Quellsysteme auf die Leistung des virtuellen Data Warehouses untersucht. Ähnlich zu den vorherigen Abschnitten stellt Abbildung 4.7 die Veränderung der Laufzeiten nach Erhöhung der Anzahl der Quellsysteme von drei auf zehn Quellen dar. Beispielsweise gibt der Wert 142% bei der unpartitionierten Implementierung und hohem Volumen / Komplexität eine Verlangsamung um 42% nach Erhöhung der Anzahl der Quellsysteme wieder.

Im Gegensatz zum traditionellen Data Warehouse hat die Anzahl der Quellen bei einem virtuellen Data Warehouse einen erheblichen Einfluss auf die Art und Weise, wie die Anfragen verarbeitet werden. Alle Quellen müssen bereinigt, normalisiert und integriert werden. Dies passiert bei den traditionellen Ansätzen während des ETL-Prozesses. Der Einfluss der Erhöhung der Quellsysteme auf das virtuelle Data Warehouse wird an einem Beispiel verdeutlicht. Bei einem 10GB Datenset würde dieses im Falle von drei Quellen in drei Teile aufgeteilt sein, jede der Quellen würde ungefähr 3,33GB an Daten enthalten. Im Falle von 10 Quellen würden es 10 Teile mit jeweils 1GB an Daten sein. Die insgesamt zu verarbeitende Datenmenge ist also

gleich, bei 10 Quellsystemen wird die Arbeit jedoch weiter aufgeteilt und die Integrationsphase ist komplexer. Aufgrund dieser erhöhten Komplexität der Ablaufpläne wurde eine Verschlechterung der Leistung nach der Erhöhung der Quellsysteme erwartet. Die Praxis zeigt, dass dies nicht unbedingt der Fall ist.

Zunächst wird der Effekt dieses Parameters auf die unpartitionierte Implementierung untersucht. Nach Abbildung 4.7(a) verhält sich diese sehr instabil, ein Blick auf die unaggregierten Daten bestätigt dies. Die unaggregierten Daten können in Abschnitt A.2 eingesehen werden. Allgemein ist eher eine erwartete Verschlechterung der Leistung zu sehen. Die detaillierten Daten zeigen sehr extreme Ausreißer (von 20% bis 811%), außerdem ließen sich keine Muster aus den Daten ableiten. Zum Beispiel verbessert sich die Leistung nach Erhöhung der Quellen bei hohem Volumen / Komplexität und 1GB an Daten um das fünffache (fünf mal schneller). Bei den 10GB / 25GB Datensets verschlechtert sich bei 10 Quellen und derselben Anfrage die Leistung um das vier- bis achtfache. Beim 50GB Datenset verbessert sich die Leistung wiederum, ähnlich wie beim 1GB Datenset. Es wird vermutet, dass der Impala-Query-Optimizer diese Ausreißer durch schlecht optimierte Ablaufpläne produziert, dies ist bei der unpartitionierten Implementierung schon einige Male aufgetreten. Es konnte bisher leider keine sichere Erklärung für dieses Verhalten gefunden werden.

Die partitionierte Implementierung verhält sich wie bei allen Tests bisher wesentlich stabiler. Sie profitiert bei Anfragen mit hohem / mittlerem Volumen von der Erhöhung der Anzahl der Quellen, bei geringem Anfragevolumen ist wiederum eine deutliche Verlangsamung zu sehen. Aus den detaillierteren Abbildungen lässt sich folgendes schließen: Eine Erhöhung der Anzahl der Quellen führt bei diesem virtuellen Data Warehouse nur bei großen zu verarbeitenden Datenmengen zu einer Beschleunigung der Verarbeitungsgeschwindigkeit. Wenn dieses Datenvolumen durch ein kleineres Datenset oder durch eine selektierende Anfrage reduziert wird, dann erhöht sich der Aufwand durch die zusätzlichen Quellsysteme deutlich.

Die Unterschiede zwischen den beiden Implementierungen liegen in einer feingranulareren Aufteilung der Dateien und in unterschiedlich komplexen Ablaufplänen. Bei zehn Quellen sind die Aufgaben feiner aufgeteilt. Diese tiefe Aufteilung der Daten und Aufgaben führt bei kleinen Datenmengen zu einem deutlich sichtbaren Overhead. Wenn die Datenmengen jedoch steigen, dann kann dieser Overhead durch eine verbesserte Parallelisierung ausgeglichen werden, bei sehr hohen Datenmengen sind die Laufzeiten bei zehn Quellen sogar besser als bei drei Quellen. Letztendlich ist bei dieser Implementierung das Verhältnis von den zu verarbeitenden Datenmengen und die Anzahl der Quellen entscheidend. Ein äußerst riesiges Datenset aus nur einer Quelle würde vermutlich durch eine feine Aufteilung der Daten profitieren. Dies kann zum Beispiel durch eine Aufteilung in mehrere Quellen oder durch eine tiefere Partitionierung der Daten erreicht werden. Andererseits würde ein virtuelles Data Warehouse, welches aus sehr vielen kleinen Quellen besteht, durch eine teilweise Aggregation dieser profitieren. In der Praxis hat man jedoch kaum Einfluss auf die Anzahl und den Inhalt der Quellsysteme. Die Ergebnisse dieses Abschnitts sollen nur verdeutlichen, welchen Einfluss die unterschiedliche Verteilung von Daten auf die Leistungen von virtuellen Data Warehouses haben können.

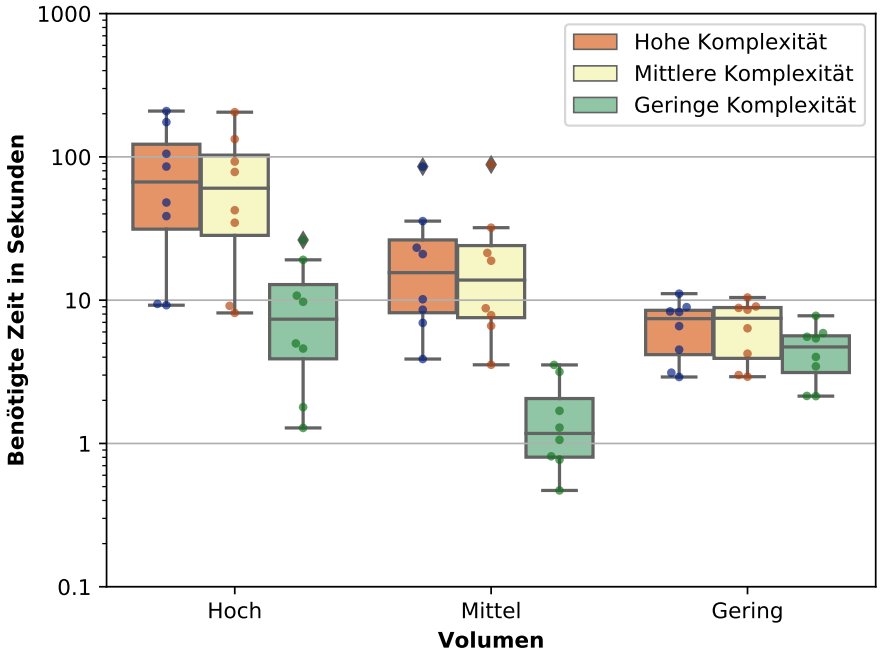



Abbildung 4.8: Einfluss des Anfragevolumens und der Anfragekomplexität auf die Leistungen des partitionierten virtuellen Data Warehouses

### 4.3.4 Einflüsse durch die Arten von Anfragen

An dieser Stelle sollen die Auswirkungen des Anfragevolumens und der Anfragekomplexität auf die Leistung des virtuellen partitionierten Data Warehouses betrachtet werden. Die Auswirkungen sind nahezu identisch zu denen des traditionellen Data Warehouses, weshalb diese nur kurz wiederholt werden:

- Je selektiver die Anfragen, desto besser skaliert das Data Warehouse (Erkennbar an der kleiner werdenden Varianz bei selektiven Anfragen)
- Das Anfragevolumen hat mit und ohne Partitionierung einen enormen Einfluss auf die Verarbeitungsgeschwindigkeiten. Mit Partitionierung ist der Vorteil aber nochmal um einiges größer
- Verbunde zwischen Tabellen resultieren in einem vervielfachen des Aufwandes bei allen Anfragevolumen (Geringe → Mittlere Komplexität)
- Ein Gruppieren und anschließendes sortieren ist mit einem kaum erkennbaren Aufwand verbunden (Mittlere → Hohe Komplexität)

Ein Boxplot sticht bei dieser Abbildung jedoch heraus, der grüne Boxplot bei mittlerem Volumen und geringer Komplexität. Obwohl diese Anfrage nicht so restriktiv ist wie die Anfrage bei geringem Volumen und geringer Komplexität, verläuft diese schneller. Dieses Merkmal widerspricht den oben erwähnten Schlussfolgerungen, weil eine Anfrage mit einer zusätzlichen Restriktion auf den Datenmengen schneller sein sollte als dieselbe Anfrage ohne diese zusätzliche Restriktion. Es gibt jedoch eine leichte Erklärung für dieses Verhalten.

Der Unterschied zwischen mittlerem und geringem Anfragevolumen ist eine zusätzliche Selektion bei geringem Volumen. Diese dazukommende Selektion findet auf der Kundendimension statt, dass heißt die Faktentabelle muss mit einer zusätzlichen Tabelle verbunden werden, um die Selektion durchführen zu können. Beim mittlerem Volumen ist diese Selektion und damit auch der Verbund mit dieser Tabelle unnötig. Das bedeutet also, dass der Aufwand des Verbundes höher ist, als der Nutzen der hierdurch ermöglichten Selektion. Bei mittlerer und hoher Komplexität ist dieser Effekt nicht zu sehen, weil bei diesen Anfragen sowieso alle Verbunde erzeugt werden. Auch beim Übergang von hohem zu mittlerem Volumen ist dieser Effekt nicht zu sehen, weil die Informationen der Zeit-Dimension in die Faktentabelle integriert sind. Beim traditionellen Data Warehouse ist das Ergebnis übrigens dasselbe, es ist nur weniger ausgeprägt. Aus diesem Grunde wurde beim virtuellen Data Warehouse darauf eingegangen.

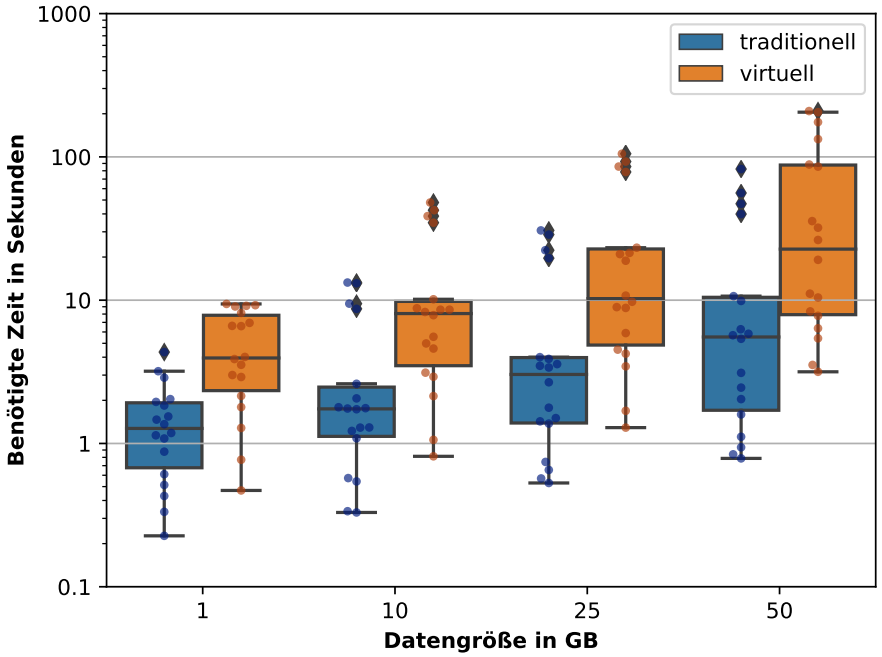



Abbildung 4.9: Alle Anfragen auf den partitionierten Data Warehouses

## 4.4 Traditionell vs. Virtuell

Der Fokus liegt in diesem Abschnitt auf dem Vergleich der beiden soeben vorgestellten Ansätze miteinander. Es wird geschaut, ob und wie die Implementierungen mit den Datenmengen und Anforderungen umgehen können. Welcher Ansatz wäre bei den gegebenen Problemen der bessere und wie gut skalieren die Ansätze bei steigenden Datenmengen? Welchen Einfluss haben die untersuchten Parameter auf die Performanz? Reagieren beide Ansätze unterschiedlich auf sich ändernde Parameter? Diese und weitere Fragen werden im Verlauf des Unterabschnitts beantwortet.

### 4.4.1 Genereller Vergleich

In Abbildung 4.9 werden alle Anfragen auf die partitionierten Versionen des traditionellen und virtuellen Data Warehouses zusammenfassend dargestellt. Aufgrund der sehr schlechten Performanz der unpartitionierten Lösungen bei beiden Ansätzen, werden diese bei den folgenden Analysen ausgelassen. Auf dieser Grafik wird nochmal deutlich, dass die beiden Ansätze sehr gleichartig auf verschiedene Parameter reagieren. Sie haben ähnliche Ausreißer, skalieren ähnlich gut und reagieren ähnlich auf verschiedene Anfragen. Der große Unterschied ist, dass das virtuelle Data Warehouse bei diesen Datenmengen um einen konstanten Faktor langsamer zu sein scheint.

Die Grafik verdeutlicht nochmal, dass beide Ansätze bei den gegebenen Szenarien und Anfragen eine gute Leistung bieten können. Auch bei den aller komplexesten analytischen Anfragen beträgt die Wartezeit maximal bis zu ca. drei Minuten. Wenn diese in der Praxis seltenen und äußerst aufwendigen Anfragen außen vor gelassen werden, dann schafft es das traditionelle Data Warehouse alle übrigen Anfragen in

| Ansatz | 1GB | 10GB | 25GB | 50GB |
|---|---|---|---|---|
| Traditionell | 1,27 | 1,74 | 3,03 | 5,53 |
| Virtuell | 3,95 | 8,06 | 10,25 | 22,71 |
| *Faktor* | 3,11x | 4,63x | 3,38x | 4,10x |

Tabelle 4.3: Mittlere Laufzeit in Sekunden bei verschiedenen Datenmengen. Die letzte Zeile gibt den Faktor an, um den der virtuelle Ansatz langsamer ist

| Ansatz | 1GB ->10GB | 10GB ->25GB | 25GB ->50GB |
|---|---|---|---|
| Traditionell | 0,04 | 0,49 | 0,82 |
| Virtuell | 0,11 | 0,17 | 1,21 |

Tabelle 4.4: Skalierungsfaktoren bei verschiedenen Datenübergängen

unter zehn Sekunden zu verarbeiten. Beim virtuellen Data Warehouse liegt diese Grenze bei 30 bis 40 Sekunden, also bei dem drei- bis vierfachen der Zeit.

Wenn man einen Blick auf die Mediane bei verschiedenen Datenmengen in Tabelle 4.3 wirft, dann sieht man, dass das virtuelle Data Warehouse bei allen Datenmengen drei- bis fünfmal länger benötigt als das traditionelle Data Warehouse. Erstaunlich ist, dass sich dieser Faktor über alle Datenmengen nicht großartig verändert. Sie skalieren somit ähnlich gut, wie in einem nachfolgenden Paragraph etwas ausführlicher erläutert wird. Etwas enttäuschend ist, dass beide Ansätze durch die Datenmengen nicht an ihre Grenzen gebracht werden konnten. Es wäre interessant zu sehen, ab welchen Datenmengen der Spalt zwischen ihnen anfangen würde größer zu werden.

*Ausreißer*
Wie erwartet sind bei beiden Ansätzen dieselben Ausreißer zu sehen. Beide Data Warehouses werden somit durch ähnliche Anfragen stark oder weniger stark ausgelastet. Durch die Virtualisierung ändert sich an diesem Verhalten also recht wenig. Die aller aufwendigsten Anfragen werden vom traditionellen Data Warehouse in knapp 1,5 Minuten verarbeitet. Das virtuelle Data Warehouse benötigt für dieselben Anfragen rund 3,3 Minuten, also etwas mehr als doppelt soviel Zeit. Dies ist erstaunlich: Zunächst wurde erwartet, dass das virtuelle Data Warehouse bei besonders aufwendigen Anfragen und besonders großen Datensets aufgrund der Architektur wesentlich mehr Zeit benötigen würde, als das traditionelle Data Warehouse. Bei den gegebenen Datenmengen konnte der virtuelle Ansatz verglichen mit dem traditionellen Ansatz eine ähnlich gute Skalierung vorweisen.

*Skalierung*
In den letzten Abschnitten wurde bereits etwas deutlich, dass beide Ansätze relativ ähnlich skalieren. An dieser Stelle werden die Details erläutert. In Tabelle 4.4 werden die Skalierungsfaktoren für beide Ansätze bei unterschiedlichen Datenmengen dargestellt. Der Faktor von 0,49 beim Übergang von 10GB auf 25GB an Daten stellt Beispielsweise für das traditionelle Data Warehouse dar, dass sich die Laufzeit um $\sim$50% erhöht, wenn die Datenmenge um 100% gesteigert wird. Ein Faktor von 1,0 repräsentiert eine lineare Skalierung. Die Laufzeit erhöht sich also um 100%, wenn

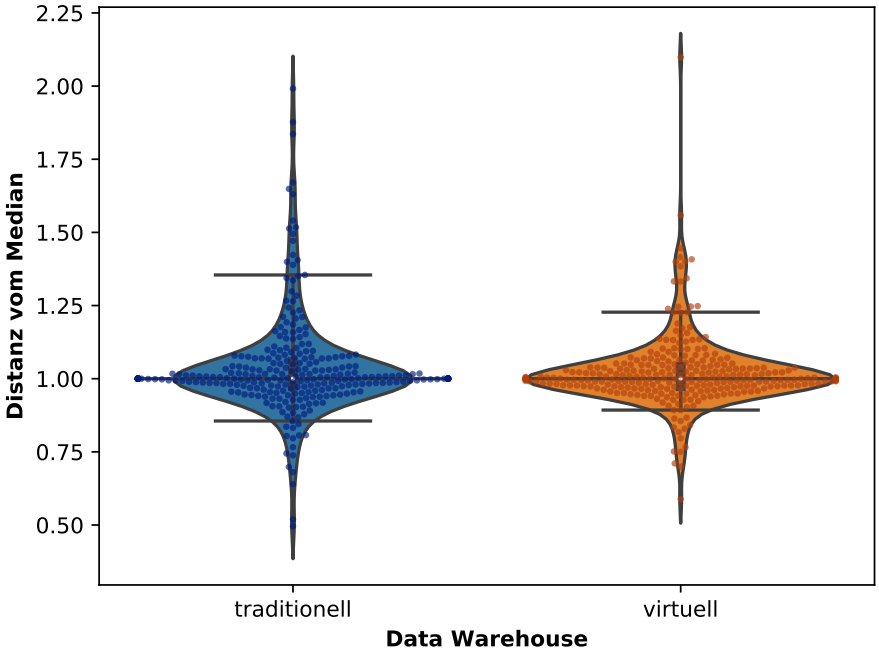



Abbildung 4.10: Abweichungen der Anfragen über mehrere Durchläufe

die Datenmenge um 100% erhöht wird. Diese Faktoren wurden für beide Ansätze bei den unterschiedlichen Übergängen der Datenmengen berechnet.

Es ist sehr deutlich zu sehen, dass beide Ansätze bei kleinen Datenmengen (1GB bis 10GB) außerordentlich gut skalieren, die Skalierungsfaktoren liegen hier bei 0,04 (traditionell) und 0,11 (virtuell). Die Laufzeit erhöht sich bei einer Datenerhöhung somit kaum, das traditionelle Data Warehouse skaliert an dieser Stelle etwas besser als das virtuelle. Beim Übergang von 10GB auf 25GB an Daten ist eine deutliche Verschlechterung der Skalierung beim traditionellen Ansatz zu sehen während das virtuelle Data Warehouse weiterhin nur einen Skalierungsfaktor von 1,173 aufweist. Beim letzten Übergang von 25GB auf 50GB an Daten ist bei beiden Ansätzen eine deutliche Verschlechterung der Skalierung zu verzeichnen. Beim traditionellen Ansatz liegt der Faktor bei 0,82, eine lineare Skalierung ist beinahe erreicht. Beim virtuellen Ansatz liegt der Faktor bei 1,21 und ist somit etwas schlechter als eine lineare Skalierung.

Hier wird deutlich, dass beide Systeme bei den gegebenen Datenmengen nicht ausgelastet werden konnten. Es wäre interessant zu wissen, wie sich die Skalierungsfaktoren bei einem weiteren Anstieg der Datenmengen verändern würden. Zum Beispiel bei einem Anstieg auf 100GB und 200GB an Quelldaten. Es wird angenommen, dass der Unterschied zwischen dem virtuellen und traditionellen Ansatz bei noch größer werdenden Datenmengen gravierender werden würde. Das virtuelle Data Warehouse müsste aufgrund des schneller ansteigenden Aufwandes schlechter skalieren.

### 4.4.2 Stabilität und Relevanz der Ergebnisse

Nachfolgend wird auf die Stabilität der Systeme und auf die statistische Relevanz der Ergebnisse eingegangen. Für die Evaluierung wurden insgesamt 864 Anfragen an

die Data Warehouses ausgewertet. Aufgrund der Menge der untersuchten Parameter und Arten von Anfragen, ergaben sich 288 unterschiedliche Anfragen. Jede hiervon wurde dreimal ausgeführt, wodurch die Gesamtmenge von 864 Queries entstand. Diese drei Wiederholungen wurden ausgeführt, um die Signifikanz der Ergebnisse zu erhöhen und den Einfluss von Ausreißern zu reduzieren. Diese Wiederholungszahl ist leider sehr gering und repräsentiert keine statistische Signifikanz. Aus zeitlichen Gründen war eine weitere Erhöhung hiervon nicht möglich.

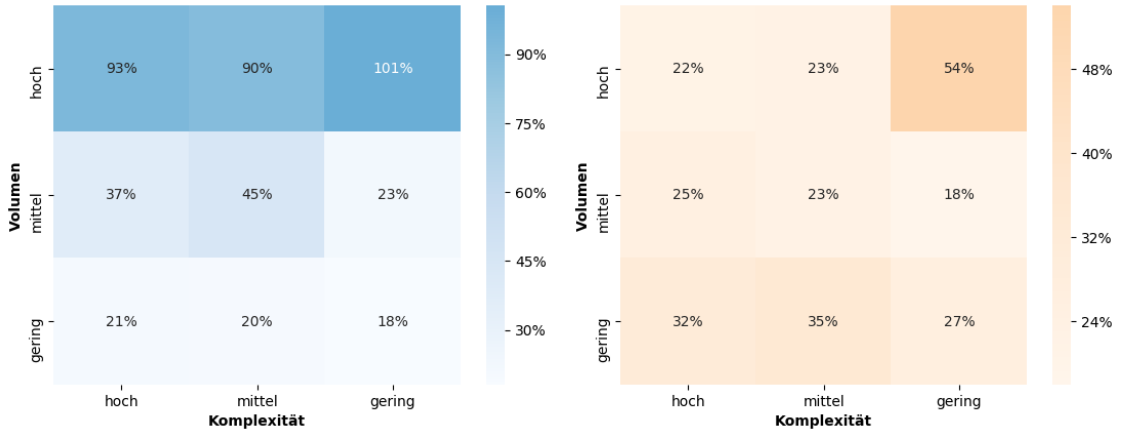
Aus diesem Grunde wird an dieser Stelle ein Überblick über die Varianz der untersuchten Anfragen gegeben. In Abbildung 4.10 wird für beide Ansätze diese Varianz dargestellt. Es wurde für jede unterschiedliche Anfrage (288 Stück) der Median über die drei Durchläufe berechnet. Jeder einzelne Datenpunkt in dieser Grafik (576 Stück) repräsentiert einen der zwei Läufe, die nicht den berechneten Median darstellen. Auf der Y-Achse wird der Abstand dieser Punkte von dem Median für diese Art von Anfrage dargestellt. Zum Beispiel bedeutet ein Wert von 2,0, dass dieser Lauf doppelt soviel Zeit benötigt hat wie der Median über diese Anfrage. Über die Datenpunkte wurden Violin-Plots gezeichnet. Diese beschreiben anhand der Breite die Wahrscheinlichkeitsdichte der Daten. Zusätzlich wurden zwei "Whisker" eingezeichnet, innerhalb derer 95% der Datenpunkte liegen.

Die Grafik stellt somit übersichtlich dar, wie groß die Schwankungen über die drei Wiederholungen für jede einzelne Art von Anfrage sind. Anhand der Grafik kann man schnell erkennen, dass ein Großteil der Anfragen relativ stabil verlief. Es gab jedoch einige wenige extreme Ausreißer, die bis zu doppelt oder halb so lang benötigten wie der Median der drei Anfragen. 95% der Daten befinden sich beim traditionellen Ansatz im Bereich zwischen 0,8 und 1,55. 90% der Daten liegen im Bereich +/-20% vom Median und 50% liegen im Bereich +/-5% um den Median herum. Ähnlich sieht es beim virtuellen Ansatz aus, hier sind die Ergebnisse interessanterweise sogar noch besser. 95% der Anfragen liegen im Bereich zwischen 0,85 und 1,35. Somit war das virtuelle Data Warehouse bei den Wiederholungen über die einzelnen Anfragen hinweg stabiler als das traditionelle Data Warehouse. Über unterschiedliche Anfragen hinweg verhielt sich jedoch zumindest die unpartitionierte Implementierung um einiges instabiler. Diese Erkenntnisse ersetzten keine statistische Relevanz durch umfangreichere Tests. Es soll nur ein Überblick darüber gewährt werden, inwiefern die Ergebnisse durch Ausreißer beeinflusst werden könnten.

### 4.4.3 Einfluss der Partitionierung

Die Partitionierung beeinflusst beide Systeme sehr positiv. Ausnahmslos alle Anfragen haben bei den Tests durch eine Partitionierung nach Jahr und Monat profitiert, sogar die Anfragen, die keine Selektionen durchführen. Als Gründe hierfür wurden zum einen die offensichtliche Verbesserung durch die Partitionierung selbst festgestellt (bei selektiven Anfragen), zum anderen wurde die Performanz durch eine erhöhte Parallelisierung verbessert. In Abbildung 4.11 werden die Auswirkungen der Partitionierung auf beide Ansätze nochmal zusammengefasst.

Diese Auswirkungen hängen bei beiden Ansätzen von der Selektivität der Anfragen ab. Eine Selektion auf der Dimension der Partitionierung hat enorme Auswirkungen auf die Performanz (2- bis 5-mal schneller). Beim traditionellen Ansatz profitieren




(a) Traditionell (b) Virtuell

Abbildung 4.11: Relative Performanz nach Einführung der Partitionierung

auch Selektionen entlang anderer Dimensionen geringfügig (bis zu 50% schneller). Die Komplexität einer Anfrage hat wenig bis kaum Auswirkungen auf den Vorteil durch die Partitionierung. Hier ist wichtig zu beachten, dass nur der reine Einfluss der Partitionierung betrachtet wird. Eine Selektion entlang einer nicht partitionierten Dimension beschleunigt eine Anfrage wesentlich deutlicher als um 50%. Hier liegt der Fokus auf der erreichten Beschleunigung beim partitionierten Data Warehouse verglichen mit dem unpartitionierten. Die Schlussfolgerung ist also, dass das partitionierte traditionelle Data Warehouse auch durch Selektionen entlang von unpartitionierten Dimensionen mehr profitiert als das unpartitionierte traditionelle Data Warehouse.

Die Implementierung des virtuellen Ansatzes ist für ein zuverlässiges Verhalten von der Partitionierung abhängig. Ohne diese Optimierung scheint der Impala-Query-Optimizer Probleme zu haben effiziente Ablaufpläne zu erstellen. Es ist häufiger vorgekommen, dass Anfragen aufgrund eines schlecht optimierten Ablaufplans bis zu zehn Minuten gedauert haben. Eine sehr ähnliche Anfrage mit geringfügigen Änderungen dauert wiederum nur einige wenige Minuten. Dieses inkonsistente Verhalten wurde durch die Einführung der Partitionierung komplett beseitigt. Der umgesetzte traditionelle Ansatz hatte hiermit keine Probleme. Die unpartitionierte Implementierung war viel langsamer, aber trotzdem stabil.

### 4.4.4 Einflüsse durch die Arten von Anfragen

Die Art und Weise, auf die die beiden implementierten Ansätze Anfragen verarbeiten, ist völlig unterschiedlich. Das traditionelle Data Warehouse verbindet die Faktentabelle mit den dimensionellen Tabellen, selektiert die Fakten anhand dieser Dimensionen und aggregiert zum Schluss die Daten. Beim virtuellen Data Warehouse gliedert sich die Arbeit völlig unterschiedlich. Eine Anfrage wird in sehr viele kleine Teilaufgaben aufgebrochen, bis auf die Ebene der Quellen. Diese Teilaufgaben beinhalten viele zusätzliche Tätigkeiten wie die Bereinigung und Integration von Daten.

Trotz dieser Unterschiede haben die Parameter Anfragevolumen und -komplexität fast identische Auswirkungen auf die Laufzeiten der Anfragen. Diese wurden in den vorherigen Unterkapiteln bereits ausführlicher erläutert, nachfolgend eine kurze Zusammenfassung:

- Eine höhere Selektivität der Anfragen verbessert die Skalierbarkeit
- Die Selektivität von Anfragen hat mit und ohne Partitionierung einen enormen Einfluss auf die Verarbeitungsgeschwindigkeiten. Mit Partitionierung ist der positive Effekt nochmal um einiges größer
- Das Ausführen von Verbunden kann den Aufwand und damit die Laufzeit vervielfachen. Mit steigendem Datenvolumen steigt auch der zusätzliche Aufwand
- Das Gruppieren und anschließende sortieren von Ergebnissen ist mit einem kaum erkennbaren Aufwand verbunden.

## 4.5 Diskussion & Schlussfolgerungen

In diesem letzten Abschnitt werden die bisher gefundenen Ergebnisse interpretiert, diskutiert und Schlussfolgerungen daraus gezogen. Zunächst werden die Einflüsse der untersuchten Parameter auf die beiden Ansätze diskutiert. Hierzu gehören die Partitionierung, die Anzahl der Quellen, das Anfragevolumen und die Anfragekomplexität. Anschließend werden die allgemeinsten Schlussfolgerungen erläutert und die Vor- und Nachteile der beiden Ansätze bei den gegebenen Anforderungen abgewogen.

### 4.5.1 Auswirkungen der Parameter

Nachfolgend werden für jeden untersuchten Parameter die wichtigsten Ergebnisse diskutiert.

**Partitionierung**
Bei der Evaluierung des Einflusses der Partitionierung sind mehrere Ergebnisse herausgekommen, zusammengefasst wurde folgendes beobachtet:

- Sehr ähnliche Auswirkungen auf traditionellen und virtuellen Ansatz
- Ausnahmslos positiver Einfluss auf Verarbeitungsgeschwindigkeit
- Verbesserung der Stabilität
- Anfragekomplexität und Partitionierung beeinflussen sich nicht

*Sehr ähnliche Auswirkungen auf traditionellen und virtuellen Ansatz*
Die Auswirkungen, die die Partitionierung auf die beiden untersuchten Ansätze hatte, waren erstaunlich übereinstimmend. Die oben erwähnten Beobachtungen wurden bei beiden Ansätzen in einem ähnlichen Ausmaß gemessen. Die Ablaufpläne für eine Abfrage sind beim traditionellen und beim virtuellen Ansatz völlig unterschiedlich. Eines haben sie jedoch gemeinsam: Bevor irgendwelche Berechnungen durchgeführt werden können, müssen die benötigten Fakten- und Dimensionsdaten von der Festplatte gelesen werden. Und hier liegt die Erklärung für die Beobachteten ähnlichen Auswirkungen der Partitionierung. Beide Data Warehouses müssen eine ähnliche Menge an Faktendaten von der Festplatte lesen, und hier greift die Partitionierung bei beiden. Auch wenn die Ablaufpläne auf den ersten Blick völlig unterschiedlich aussehen, sind sie sich auf der tiefsten Ebene erstaunlich ähnlich. Hier profitieren beide Ansätze in einem ähnlichen Maße durch eine effizientere Selektion der Daten und eine erhöhte Parallelisierung beim Lesen. Dies erklärt die sich ähnelnden Ergebnisse, jedoch nicht die Ausreißer. Diese werden in einem der nachfolgenden Paragraphen erläutert. Zusammenfassend lässt sich schließen, dass die Partitionierung einen ähnlichen Einfluss auf Berechnungen hat, die auf dieselben Daten angewiesen sind.

*Ausnahmslos positiver Einfluss auf Verarbeitungsgeschwindigkeit*
Es haben alle untersuchten Anfragen unter allen untersuchten Bedingungen von der Partitionierung mehr oder weniger profitiert. Auch Full-Table-Scans und Selektionen entlang unpartitionierter Dimensionen haben erstaunlicherweise profitiert. Ursache für diese Verbesserungen waren hauptsächlich die folgenden zwei Punkte:

- *Ausschließen von großen Datenmengen*

  Das ist der offensichtliche Grund für die schnelle Verarbeitung von Anfragen, die entlang der partitionierten Dimension selektieren. Ein großer Teil der Faktendaten kann durch die Partitionierung direkt ausgeschlossen werden.

- *Erhöhte Parallelisierung*

  Das ist die weniger offensichtliche Ursache für die Verbesserung der Laufzeiten bei Anfragen, die entweder gar nicht selektieren (Full-Table-Scan), oder entlang einer unpartitionierter Dimension selektieren. Durch die Partitionierung wurden die Dateien der Faktentabelle in kleinere Dateien aufgespalten. Hierdurch konnte Impala die Ablaufpläne so anpassen, dass eine höhere Parallelisierung möglich war.

Hieraus resultiert, dass die erhöhte Parallelisierung nicht unbedingt ein Vorteil der Partitionierung sein muss. Wenn die Datenmengen bereits klein und fein aufgespalten sind, dann würde eine weitere Aufspaltung durch eine Partitionierung in einem zusätzlichen Overhead resultieren. Bei den untersuchten Implementierungen wurden die Faktendaten in einem Post-Processing-Schritt soweit es geht auf ∼128MB große Dateien aufgeteilt. Dies stellte sich aufgrund der verwendeten Komprimierung als nicht optimal heraus, eine noch feinere Aufteilung wäre besser gewesen. Eine optimierte Wahl dieser Dateigröße hätte vermutlich die Leistung der Implementierungen verbessert und die Einflüsse der Parameter wie der Partitionierung weniger verzerrt.

*Verbesserung der Stabilität*
Es wurde eine Verbesserung der Stabilität bei beiden Ansätzen beobachtet. Vor allem das virtuelle Data Warehouse hat aus dieser Hinsicht sehr profitiert. Durch die Partitionierung konnten alle extremen Ausreißer durch eine Veränderung der Ablaufpläne beseitigt werden. Die Ursache für die Generierung suboptimaler Ablaufpläne bei den unpartitionierten Lösungen konnte nicht entdeckt werden.

*Anfragekomplexität und Partitionierung beeinflussen sich nicht*
Anfragen mit unterschiedlichen Komplexitäten werden nicht durch die Partitionierung beeinflusst, zumindest nicht bei den getesteten Anfragen. Wenn eine Anfrage zum Beispiel durch einen zusätzlichen Verbund komplexer gestaltet wird, dann erhöht sich der Aufwand beim unpartitionierten und partitionierten Data Warehouse gleichermaßen. Zumindest, wenn der Verbund nicht anhand eines partitionierten Schlüssels geschieht. Wie bereits bei den vorherigen Beobachtungen geklärt, hat die Partitionierung hauptsächlich einen Einfluss auf den Ladeaufwand der Faktentabelle. Dies passt mit der gerade erwähnten Beobachtung zusammen. Die Partitionierung hatte bei den Tests keinen messbaren Einfluss auf die ausgeführten Berechnungen.

**Anfragevolumen und -komplexität**
Der Einfluss dieses Parameters war bei beiden Ansätzen nahezu identisch. Sowohl das Anfragevolumen, als auch die Anfragekomplexität haben die Data Warehouses auf eine ähnliche Art und Weise beeinflusst. Bei dem Anfragevolumen kann dies durch die sich auf der untersten Ebene ähnelnden Ablaufpläne erklärt werden. Hier befindet sich die Logik, die festlegt, welche Daten gelesen werden. Durch eine Selektion wird diese Logik bei beiden Ansätzen, wie bereits erwähnt, ähnlich beeinflusst. Auch dass die Anfragekomplexität ähnliche Auswirkungen auf die beiden

Ansätze hat, kann leicht erklärt werden. Bei höheren Anfragekomplexitäten wird der Aufwand durch zusätzliche Verbunde, Gruppierungen und das Sortieren von Endergebnissen beeinflusst. Weil bei beiden Ansätze die dazukommende Menge an Verbunden, die Menge der zu gruppierenden und zu sortierenden Daten ähnlich ist, sind die Auswirkungen dieses Parameters auch ähnlich. Dieses Ergebnis ist an diese Implementierung gebunden und nicht Allgemein gültig. Bei einer anderen Gestaltung der virtuellen Tabellen könnten die Ergebnisse anders aussehen.

*Selektion hat sehr hohe Auswirkungen auf die Verarbeitungszeit und Skalierbarkeit*
Diese Beobachtung wurde erwartet und kann leicht erklärt werden. An dieser Stelle ist es wichtig zwischen zwei Arten von Selektionen zu unterscheiden. Selektionen, die entlang einer partitionierten Dimension filtern und Selektionen, die dies nicht tun. Wenn die Daten entlang einer partitionierten Dimension gefiltert werden, ist der Geschwindigkeitsvorteil nochmal um einiges höher. Die folgenden zwei Ursachen sind hierfür verantwortlich:

- *Reduzierung des Leseaufwandes*

  Dies ist der Grund, wieso eine Selektion entlang einer partitionierten Dimension nochmal um einiges profitabler ist. Nur bei einer Partitionierung können große Teile der Daten direkt ausgeschlossen werden. Ansonsten müssen alle Daten gelesen werden um die benötigten Daten zu finden.

- *Reduzierung des Rechenaufwandes*

  Durch diesen Punkt wird der Aufwand reduziert, egal ob die Selektion entlang einer partitionierten oder unpartitionierten Dimension stattgefunden hat. Die durchzuführenden Berechnungen sind immer gleich, nur die Menge der zu verarbeitenden Daten ist unterschiedlich.

Interessant ist auch, dass eine selektive Anfrage wesentlich besser skaliert als eine, die nicht selektiert. Wenn die zu verarbeitende Datenmenge beispielsweise von 1GB auf 50GB erweitert wird, dann wird sich die relative Verarbeitungszeit bei einer selektiven Anfrage weniger erhöhen, als bei einer nicht selektiven. Der Grund hierfür liegt darin, dass der Anstieg der absolut zu verarbeitenden Datenmenge bei der selektiven Anfrage wesentlich geringer ist. Zusätzlich besteht bei sehr einfachen Anfragen ein großer Teil der Verarbeitungszeit aus Overhead in verschiedenen Bereichen. Ein Anstieg der zu verarbeitenden Datenmengen hat somit einen wesentlich geringeren Einfluss auf die relative Verarbeitungszeit, als bei sehr komplexen Anfragen. Bei diesen besteht nämlich ein Großteil der Verarbeitungszeit aus den durchzuführenden Berechnungen.

*Auswirkungen der Anfragekomplexität*
Auch die Anfragekomplexität wurde in zwei Stufen erhöht. Bei der ersten Erhöhung wurden die Anfragen durch zusätzliche Verbunde erweitert. Diese Änderung hatte sehr hohe Auswirkungen auf die Laufzeiten, welche deutlich von dem Anfragevolumen und Datenvolumen abhängen. Bei einem Anstieg der zu verarbeitenden Datenmengen steigt der Aufwand somit deutlich an (bis zu Verzehnfachung der Laufzeit). Bei sehr selektiven Anfragen hält sich der Aufwand in Grenzen, auch bei 50GB an Daten steigt die Laufzeit nur um $\sim$60% an.

Bei der zweiten Erhöhung wurde der Aufwand durch ein Gruppieren der Daten anhand der Produktnamen und ein anschließendes Sortieren anhand des Profits gesteigert. Erstaunlicherweise hatten diese Operationen kaum Auswirkungen auf die Laufzeiten, vor allem beim virtuellen Data Warehouse nicht. Bei hohem Anfragevolumen und Datenmengen betrug der Anstieg der Laufzeit bis zu ∼33% beim traditionellen Ansatz und ∼10% beim virtuellen Ansatz. Bei mittlerem oder geringem Volumen ist kaum ein messbarer Unterschied zu sehen.

**Anzahl der Quellen**
Die Anzahl der Quellsysteme wirkt sich wie erwartet unterschiedlich auf die beiden untersuchten Ansätze aus. Die zwei wichtigsten Ergebnisse werden nachfolgend Vorgestellt. Der erste bezieht sich auf das traditionelle Data Warehouse und der zweite auf das virtuelle.

*Beeinflusst auch das traditionelle Data Warehouse*
Die Anzahl der Quellen hatte einen Einfluss auf das implementierte traditionelle Data Warehouse, weil die Aufteilung der Faktendaten im HDFS zu einem Teil von der Anzahl der Quellen abhing (mehr dazu in Abschnitt 4.2.3). Diese Aufteilung hatte wiederum einen Einfluss auf die Performanz. Die angestrebte Dateigröße von 128MB war nicht optimal und konnte bei vielen Szenarien aufgrund des Aufbaus des Integrationsprozesses nicht erreicht werden. Der Einfluss dieses Parameters hätte deutlich reduziert werden können, wenn eine optimale Dateigröße gewählt worden wäre und diese durch einen anderen Integrationsprozess oder einen zusätzlichen Post-Processing-Schritt erreicht werden würde.

*Einfluss sehr abhängig von Datenvolumen, eher negativer Einfluss*
Wie erwartet hat die Anzahl der Quellen einen deutlichen Einfluss auf das semi-virtuelle Data Warehouse. Schließlich hängt die Komplexität der Ablaufpläne sehr von diesem Parameter ab, wie im Abschnitt A.4 dargestellt wird. Je mehr Quellen vorhanden sind, desto umfangreicher sind die Integrationsprozesse. Erstaunlicherweise gibt es auch Fälle, in denen der Ansatz von einer höheren Anzahl von Quellsystemen profitiert hat.

Alles in allem wurde das semi-virtuelle Data Warehouse durch diesen Parameter deutlich negativ beeinflusst, die Auswirkungen hängen jedoch vom Daten- und Anfragevolumen ab. Bei geringem Volumen benötigen Anfragen bis zu dreimal länger zur Verarbeitung. Bei sehr hohem Volumen konnte die Leistung sogar um einen geringen Faktor (30%) verbessert werden. Da an dieser Stelle wieder der einzige sich ändernde Faktor die Größe der Dateien ist, könnte eine verbesserte Parallelisierung wieder eine Rolle spielen. Durch diese könnte der zusätzliche Overhead des komplexeren Ablaufplans bei hohem Volumen ausgeglichen werden. Dies ist an dieser Stelle nur eine Vermutung, aus Zeitgründen wurde hier nicht tiefer nachgehakt.

### 4.5.2 Generelle Schlussfolgerungen und Empfehlung

Das semi-virtuelle Data Warehouse stellte sich in der getesteten Umgebung als valide Alternative heraus. Es konnte genauso gut skalieren wie der traditionelle Ansatz, auch war es in der partitionierten Version genauso stabil. Bis auf die Anzahl der Quellen wurde es durch die untersuchten Parameter ähnlich beeinflusst. Der entscheidende Nachteil ist die verschlechterte Verarbeitungsgeschwindigkeit. Für alle

Anfragen wurde drei- bis fünfmal soviel Zeit zur Verarbeitung benötigt, verglichen mit dem traditionellen Ansatz. Dies ist das Resultat der ETL-Prozesse und der optimierten Persistierung dieser beim traditionellen Ansatz. Die verbesserte Performanz und die Unabhängigkeit von der Anzahl der Quellen bilden die Hauptvorteile dieses Ansatzes. Wenn jedoch Berücksichtigt wird, dass es sich hierbei um komplexe analytische Abfragen handelt, und diese innerhalb von wenigen Sekunden bis zu wenigen Minuten verarbeitet werden, dann fällt der Vorteil durch diese Optimierungen als gering aus.

Vor allem wenn die Vorteile des virtuellen Data Warehouses gegenübergestellt werden, erscheinen die Geschwindigkeitsverbesserungen durch den traditionellen Ansatz als vergleichsweise gering. Das virtuelle Data Warehouse wurde bei dieser Arbeit innerhalb eines Bruchteils der Zeit entwickelt. Der Hauptgrund hierfür lag in einer wesentlich schnelleren Testbarkeit der entwickelten Komponenten. Wenn Beispielsweise eine Transformation für den ETL-Prozess des traditionellen Ansatzes entwickelt wurde, dann musste diese Transformation auf einem Test-Datenset ausgeführt werden, um zu schauen, ob die Ergebnisse korrekt sind. Der Overhead der MapReduce Jobs ist durch die vielen Zugriffe auf die Festplatten hoch und sorgt damit für lange Wartezeiten. Teilweise traten Fehler nur auf großen Datensets auf. Bei diesen dauerten die Berechnungen bis zu eine Stunde. Dieselben Komponenten konnten beim virtuellen Ansatz wesentlich schneller getestet werden. Da alle Transformationen im Arbeitsspeicher durchgeführt werden und keine Sicherung von Zwischenergebnissen auf der Festplatte notwendig sind, stehen die Ergebnisse wesentlich früher zur Verfügung. Ein weiterer wesentlicher Vorteil des virtuellen Data Warehouses ist die höhere Flexibilität. Eine Anpassung der Architektur ist durch die fehlende Persistierung mit einem geringerem Aufwand verbunden. Es müssen nur eine oder mehrere virtuelle Tabellen angepasst werden. Eine erneute Ausführung von ETL-Prozessen ist nicht erforderlich. Es können auch keine Synchronisationsprobleme auftauchen, weil es keine Schichten gibt, auf denen Duplikate vorliegen.

Diese und weitere Vor- und Nachteile wurden bereits in Kapitel 2 allgemein erläutert. In den getesteten Umgebungen ist der Geschwindigkeitsvorteil des traditionellen Data Warehouses als relativ gering ausgefallen. Das semi-virtuelle Data Warehouse hatte an dieser Stelle keine Probleme damit, die Anforderungen zu erfüllen. Gleichzeitig konnte es einige Vorteile mitbringen. Diese Ergebnisse basieren natürlich auf der getesteten Umgebung. Wie bereits erwähnt, wäre es sehr interessant zu wissen, wie die Ergebnisse bei noch größeren Datenmengen aussehen würden. Oder bei zusätzlichen und divergenteren Quellsystemen. Oder bei umfangreicheren ETL-Prozessen. Dies sind alles Faktoren, die bis zu einer Grenze untersucht wurden. Bis zu dieser Grenze konnte das virtuelle Data Warehouse ohne Probleme mitskalieren. Bei einer weiteren Erhöhung der Anforderungen könnte es anders aussehen. Je höher der Aufwand der Berechnungen, desto lohnenswerter ist eine Optimierung wie die beim traditionellen Data Warehouse. Es muss somit für jeden Fall separat entschieden werden, welcher Ansatz besser wäre. Sowohl die aktuellen Anforderungen, als auch die erwarteten Anforderungen in der Zukunft müssen bei dieser Entscheidung berücksichtigt werden. Durch diese Arbeit konnten die Möglichkeiten einer semi-virtuellen Data Warehouse Lösung etwas deutlicher gemacht werden. Die Grenzen konnten leider nicht aufgezeigt werden.

### 4.5.3 Beantwortung der Forschungsfragen

Anhand der Ergebnisse aus der Evaluierung werden nachfolgend die Anfangs gestellten Forschungsfragen beantwortet:

**Inwiefern ist ein semi-virtuelles Data Warehouse verglichen mit einem traditionellen Ansatz in der Lage, bei steigenden Datenmengen zu skalieren?**
Unter den untersuchten Bedingungen hatte das semi-virtuelle Data Warehouse keine Probleme bei steigenden Datenmengen mitzuskalieren. Es war durchschnittlich immer um das drei- bis fünffache langsamer als der traditionelle Ansatz, dieser Faktor hat sich bei steigenden Datenmengen kaum verändert. Somit konnte die virtuelle Implementierung genauso gut skalieren, wie der traditionelle Ansatz. Nachträglich hat sich jedoch herausgestellt, dass die verwendeten Datenmengen das verwendete Rechencluster nicht genug ausreizen konnten. Ein relevanter Faktor bei der Untersuchung der Skalierung ist die verwendete Menge an Daten in Relation zu den verwendeten Rechenressourcen. Das Resultat ist somit, dass das semi-virtuelle Data Warehouse unter den getesteten Bedingungen genauso gut skalieren konnte wie das traditionelle. Bei höheren Datenmengen könnte sich der Spalt zwischen den beiden weiten.

**Welchen Einfluss hat die Virtualisierung eines semi-virtuellen Data Warehouses auf dessen Verarbeitungszeiten bei analytischen Anfragen?**
Die eingetretenen Unterschiede stellten sich als geringer heraus, als erwartet. Durchschnittlich hat sich die Verarbeitungszeit um das 3,11-fache bis 4,64-fache erhöht. Bei den aufwendigsten Anfragen hat sich die Laufzeit nur verdoppelt. Wenn man sich zusätzlich die absoluten Laufzeiten anschaut, dann sieht man, dass der Mehraufwand durch die Virtualisierung an dieser Stelle vernachlässigbar ist. Bei dem größten Datenset (50GB) beträgt die mittlere Laufzeit beim traditionellen Data Warehouse 5,53 Sekunden und beim virtuellen 22,71 Sekunden. Die aufwendigste Anfrage wird in 1,5 Minuten (traditionell) und 3,3 Minuten (virtuell) verarbeitet.

**Ändert sich durch die Virtualisierung der Einfluss von Parametern wie Partitionierung, Anzahl der Quellsysteme und verschiedene Arten von analytischen Anfragen?**
Obwohl die Ablaufpläne der SQL-Query-Engine bei dem traditionellen und virtuellen Ansatz völlig unterschiedlich sind, haben die untersuchten Parameter sehr ähnliche Einflüsse auf die beiden Architekturen. Sowohl die Partitionierung, das Anfragevolumen, als auch die Anfragekomplexität haben sehr ähnliche Auswirkungen auf die beiden Ansätze.

Alles in allem resultierte nur die Anzahl an Quellen für sehr unterschiedliche Reaktionen bei beiden Ansätze. Während das traditionelle Data Warehouse die hierdurch entstehenden Änderungen durch den ETL-Prozess größtenteils ausgleicht, muss das virtuelle Data Warehouse diese zusätzliche Arbeit spontan erledigen, wodurch die Laufzeiten deutlicher beeinflusst werden.

# 5. Zusammenfassung und zukünftige Arbeiten

Nachfolgend wird die Arbeit kurz mit den wichtigsten Kernaussagen zusammengefasst und die anfangs gestellten Forschungsfragen beantwortet. Anschließend werden einige kritische Aspekte der Arbeit beleuchtet und ein Ausblick über mögliche aufbauende Arbeiten gegeben.

## 5.1 Reflexion der Vorgehensweise

Zunächst wird in Kapitel 2 ein Überblick über die behandelten Fachbereiche gegeben und Informationen bereitgestellt, die für das weitere Verständnis der Arbeit benötigt werden. Anhand eines Use-Cases wird dargestellt, weshalb ein transaktionelles Datenbanksystem für analytische Anfragen weniger geeignet ist. Die Unterschiede zwischen transaktionellen und analytischen Systemen werden aufgezeigt und die hieraus resultierenden unterschiedlichen Anforderungen erläutert. Um mit den im Use-Case erwähnten analytischen Anforderungen umgehen zu können, kann unter anderem ein Data Warehouse verwendet werden. Es wird zunächst geklärt, was ein Data Warehouse ist, wozu man es braucht, und welche allgemeinen Konzepte es gibt. Anschließend werden einige bekannte Architekturen vorgestellt, darunter einige traditionelle und das semi-virtuelle Data Warehouse. Die für die Evaluierung gewählten Ansätze und die Gründe für deren Wahl werden erläutert und vorgestellt. Konkret wird die Performanz eines dimensionalen Data Marts mit einem semi-virtuellen Data Warehouse verglichen, bei dem bis auf die Ergebnisse der Extraktionsphase alles virtualisiert ist. Am Ende des theoretischen Teils werden die Überschneidungen zwischen dem Konzept des Data Warehouses und dem Bereich Big Data aufgezeigt. Durch diese vorhandenen Überschneidungen müssen sich auch Data Warehouse Systeme bestimmten Anforderungen stellen. Auch sie müssen mit Datenmengen umgehen können, die ein hohes Volumen besitzen, mit hoher Geschwindigkeit erzeugt werden und eine hohe Varianz an Formaten aufweisen können. Die für die Evaluierung durchgeführten Tests wurden in einer Hadoop Umgebung durchgeführt, die

Entscheidung hierfür und die verwendeten Komponenten werden in diesem Teil erläutert. Bei Hadoop handelt es sich um ein Framework, welches die Entwicklung von Big Data Infrastrukturen erleichtert. Im Kern verwenden die umgesetzten Data Warehouses eine verteilte SQL-Query-Engine innerhalb von Hadoop.

Nachdem die theoretischen Konzepte und Entscheidungen erläutert wurden, wird auf die umgesetzten Implementierungen in Kapitel 3 eingegangen. Dieses besteht aus drei Teilen, in denen die Quellschicht, das traditionelle Data Warehouse und das semi-virtuelle Data Warehouse erläutert werden. Die Quellschicht bildet das Fundament, auf das sich die beiden Data Warehouses stützen. Durch diese Schicht können zwei für die Evaluierung relevanten Parameter gesteuert werden: die zu verarbeitende Datenmenge und die Anzahl an Quellsystemen. Anschließend werden die für die Evaluierung verwendeten Architekturen vorgestellt. Die Architekturen und alle Designentscheidungen, die einen Einfluss auf die Evaluierung und die Performanz haben können, werden erläutert und begründet.

Die Evaluierung und der Vergleich der beiden untersuchten Ansätze findet anschließend in Kapitel 4 statt. Zunächst werden die Rahmenbedingungen erläutert, unter denen die Tests durchgeführt wurden. Hierdurch soll eine bessere Vorstellung über die Testdurchführung und die Validität der Ergebnisse vermittelt werden. Es wird auf die untersuchten Parameter (Datenmenge, Anzahl der Quellen, Partitionierung), auf die durchgeführten analytischen Anfragen, die zur Messung der Leistung verwendeten Performanz Metriken und das verwendete Testsystem eingegangen. Durch die Variation der Parameter konnte besser festgestellt werden, an welchen Stellen sich durch die Virtualisierung Unterschiede ergeben. Auch die Einflüsse von weiteren weniger beachteten Faktoren werden kurz erläutert. Hierzu gehören der Technologie-Stack, das Verhältnis von vorhandenen Rechenressourcen zu dem Rechenaufwand und weitere Parameter wie der Umfang der ETL-Prozesse. Zu den untersuchten Performanz Metriken gehören die Verarbeitungszeit, die Skalierbarkeit und die Stabilität. Die Präsentation der Ergebnisse wurde zunächst für jeden Ansatz für sich durchgeführt, anschließend wurden diese miteinander verglichen. Die Ergebnisse wurden diskutiert, mit den Erwartungen abgeglichen und generelle Schlussfolgerungen gezogen. Anhand dieser Erkenntnisse konnten dann die Forschungsfragen beantwortet werden. Nachfolgend werden die wichtigsten Ergebnisse zusammengefasst:

Durch die Virtualisierung haben sich die Einflüsse der untersuchten Parameter kaum verändert. Die Partitionierung, das Anfragevolumen und die Anfragekomplexität haben sehr ähnliche Auswirkungen auf die beiden Architekturen. Nur die Anzahl der Quellen beeinflusste wie erwartet das virtuelle Data Warehouse wesentlich stärker als das traditionelle, vor allem weniger komplexe Anfragen waren hiervon betroffen. Es stellte sich auch heraus, dass das virtuelle Data Warehouse erstaunlich gut mit den durch den Use-Case gestellten Anforderungen umgehen konnte. Bei den untersuchten Datenmengen bis zu 50GB konnte das virtuelle Data Warehouse nahezu genauso gut skalieren wie das traditionelle. Es war jedoch bei allen Tests um einen Faktor langsamer, der bei dem drei- bis fünffachen liegt. Dieser Faktor ist bei dem implementierten Use-Case jedoch weniger relevant, weil die meisten Anfragen innerhalb von Sekunden verarbeitet werden konnten. Bei dem größten Datenset lag der Median der Verarbeitungszeiten von allen Anfragen bei $\sim$22 Sekunden (virtueller Ansatz). Beim traditionellen Ansatz lag der Median bei $\sim$5 Sekunden. Für die

aufwendigsten Anfragen benötigte das traditionelle Data Warehouse ∼1,5 Minuten, das virtuelle ∼3,3 Minuten. Zusammengefasst lässt sich schließen, dass das virtuelle Data Warehouse sich für diesen Use-Case als bessere Alternative herausstellt. Die Vorteile einer effizienteren Entwicklung und langfristig höheren Flexibilität überwiegen an dieser Stelle die Performanceeinbußen. Jedoch stellte sich heraus, dass die getesteten Datenmengen das Testsystem nicht vollständig auslasten konnten. Interessant wäre es zu sehen, wie die Systeme mit weiter steigenden Anforderungen umgehen würden.

## 5.2 Beantwortung der Forschungsfragen

Nachfolgend werden die gestellten Forschungsfragen zusammenfassend beantwortet.

- *Inwiefern ist ein semi-virtuelles Data Warehouse verglichen mit einem traditionellen Ansatz in der Lage, bei steigenden Datenmengen zu skalieren?*

  Unter den getesteten Bedingungen hatte das semi-virtuelle Data Warehouse keine Probleme bei steigenden Datenmengen mitzuskalieren, die Skalierungsfaktoren waren nahezu identisch mit denen des traditionellen Ansatzes. Bei höheren Datenmengen könnte sich der Spalt zwischen den beiden weiten.

- *Welchen Einfluss hat die Virtualisierung eines semi-virtuellen Data Warehouses auf dessen Verarbeitungszeiten bei analytischen Anfragen?*

  Es ergaben sich wesentlich geringere Unterschiede als erwartet, durchschnittlich hat sich die Verarbeitungszeit um das 3,11-fache bis 4,64-fache erhöht. Die absoluten Laufzeiten bewegten sich im Sekunden- bis Minutenbereich, die Nachteile durch die Virtualisierung waren in der Testumgebung vernachlässigbar.

- *Ändert sich durch die Virtualisierung der Einfluss von Parametern wie Partitionierung, Anzahl der Quellsysteme und verschiedene Arten von analytischen Anfragen?*

  Trotz der sich deutlich unterscheidenden Ablaufpläne bei beiden Ansätzen haben die untersuchten Parameter sehr ähnliche Einflüsse auf die beiden Architekturen. Ein deutlicher Unterschied konnte wie erwartet nur bei der Anzahl der Quellsysteme gemessen werden.

## 5.3 Kritische Betrachtung

Es wurde zu wenige Zeit für die Durchführung der Tests reserviert, wodurch mehrere Probleme entstanden. Zum einen hätten bei einem umfangreicheren Zeitbudget die Wiederholungszahlen bei allen Tests erhöht werden können, was die statistische Relevanz der Ergebnisse verbessert hätte. Zum anderen wären Tests auf noch größeren Datenmengen möglich gewesen. Es stellte sich heraus, dass die verwendeten Datenmengen für eine vollständige Auslastung des Rechenclusters nicht ausreichend waren. Dadurch konnten die Grenzen der Virtualisierung bei dem gegebenen Use-Case und Rechencluster nicht erreicht werden.

Ein Faktor, der die Ergebnisse der Tests negativ beeinflusst hat, ist die Dateigröße der Dateien im HDFS. Diese Dateigröße ist relevant, weil die verwendete Query-Engine das maximale Potential der Parallelisierung nur nutzen kann, wenn die zu verarbeitenden Daten in ausreichend kleine Dateien aufgespalten vorliegen. Auch die resultierenden Ablaufpläne hängen hiervon ab. Bei den Tests wurden die Daten soweit es ging in ∼128MB große Dateien aufgespalten, hierbei wurde sich an der offiziellen Empfehlungen aus der Impala Dokumentation[1] ausgerichtet. Diese Größe stellte sich jedoch aufgrund der verwendeten Kompression als nicht optimal heraus, weil eine 128MB große Datei 1,28GB an unkomprimierten Faktendaten enthält. Eine noch feinere Aufteilung wäre besser gewesen. Dies ist problematisch, weil die Ergebnisse hierdurch teilweise negativ beeinflusst wurden. Zum Beispiel hat es dazu geführt, dass das traditionelle Data Warehouse durch eine Erhöhung der Anzahl der Quellen profitiert hat. Durch die Wahl einer optimalen Dateigröße[2] hätten die Auswirkungen dieses Problems auf die Ergebnisse reduziert werden können. Dieses Problem war jedoch nicht leicht zu erkennen und wurde erst bei der Auswertung der Testergebnisse entdeckt.

## 5.4 Ausblick

Der untersuchte Themenbereich ist sehr groß und es gibt viele Stellen, an denen man mehr machen könnte. Dies könnte man nutzen, um die Ergebnisse der Arbeit zu festigen oder um neue Ergebnisse in benachbarten Bereichen zu sammeln. Aufbauend auf dieser Arbeit gibt es unter anderem die folgenden Themen:

**Tests unter erhöhtem Aufwand**
Das virtuelle Data Warehouse konnte auch bei den aufwendigsten analytischen Anfragen genauso gut skalieren wie das traditionelle. Dies ist ein Ergebnis, das nicht erwartet wurde. Die Grenzen der Virtualisierung konnten durch die Tests nicht erreicht werden. Umfangreichere Tests auf größeren Datenmengen und mit komplexeren ETL-Prozessen wären an dieser Stelle interessant.

**Optimierungen virtuelles Data Warehouse**
Der Fokus dieser Arbeit lag darin den zusätzlichen Aufwand zu messen, der durch das spontane durchführen der (E)TL-Prozesse entsteht. In der Praxis werden jedoch

[1] https://www.cloudera.com/documentation/enterprise/5-15-x/topics/impala_perf_cookbook.html#perf_cookbook__perf_cookbook_partitioning

[2] nicht immer möglich, z.B. bei partitionierter Faktentabelle auf kleinen Datensets

Zwischenergebnisse von virtuellen Tabellen auf der Festplatte oder im Arbeitsspeicher gesichert, wodurch die Performanz des virtuellen Data Warehouses optimiert wird. Hierdurch würde sich das virtuelle Data Warehouse noch weiter an den traditionellen Ansatz annähern. Interessant wäre es zu untersuchen, wie sehr die Performanz durch diese Optimierung verbessert werden kann, und wie sich der Tradeoff zwischen Speicherverbrauch und Leistungsverbesserung verhält.

**Verschiedene Grade an Virtualisierung**
Der Grad der Virtualisierung kann individuell festgelegt werden. Durch eine Variation dieses Parameters könnte gemessen werden, welche Teile der ETL-Prozesse am aufwendigsten sind und wo es Sinn macht, Zwischenergebnisse zu persistieren.

**Vergleich mit traditioneller relationaler Datenbank**
Die Tests wurden auf einem Hadoop Cluster durchgeführt und es wurde eine verteilte SQL-Query-Engine verwendet. Es wäre interessant zu sehen, wie sich die Ergebnisse auf einer traditionellen relationalen Datenbank unterscheiden würden.

# A. Anhang

## A.1 Ausreißer beim virtuellen Data Warehouse

Die nachfolgenden beiden Listings repräsentieren die zusammengefassten Ablaufpläne von Impala für die folgenden beiden Anfragen:

- *Unpartitioniertes* virtuelles Data Warehouse, 25GB Daten, 10 Quellen, hohes Volumen und Komplexität
- *Partitioniertes* virtuelles Data Warehouse, 25GB Daten, 10 Quellen, hohes Volumen und Komplexität

Quelltext A.1: Unpartitioniertes virtuelles Data Warehouse, Laufzeit: 719 Sekunden

```
Operator                          #Hosts   Avg Time    Max Time    #Rows     Peak Mem  Detail
------------------------------------------------------------------------------------------------------
145:MERGING-EXCHANGE                   1    0.000ns     0.000ns       13    96.00 KB  UNPARTITIONED
92:SORT                                6  333.334us     1.000ms       13    12.02 MB
144:AGGREGATE                          6    2.833ms     4.000ms       13     1.97 MB  FINALIZE
143:EXCHANGE                           6    0.000ns     0.000ns       78    56.00 KB  HASH(pname)
91:AGGREGATE                           6   11s417ms    13s697ms       78     2.11 MB  STREAMING
00:UNION                               6   18s964ms    22s688ms    1.23B    40.00 KB
|--18:HASH JOIN                        6    8s713ms    10s019ms  123.31M     2.14 GB  INNER JOIN, PARTITIONED
|  |--102:EXCHANGE                     6    1s051ms     1s287ms  123.31M    14.88 MB  HASH(purchaseid)
|  |  14:NESTED LOOP JOIN              2    5s449ms     7s430ms   63.20B   102.00 KB  CROSS JOIN, BROADCAST
|  |  |--100:EXCHANGE                  2    0.000ns     0.000ns        1    16.00 KB  BROADCAST
|  |  |  11:SCAN HDFS                  1  112.000ms   112.000ms        1    24.00 KB  raw2.store
|  |  13:HASH JOIN                     2    2s205ms     2s924ms  123.31M     1.99 MB  INNER JOIN, BROADCAST
|  |  |--99:EXCHANGE                   2  500.000us     1.000ms       13    16.00 KB  BROADCAST
|  |  |  12:SCAN HDFS                  1  191.000ms   191.000ms       13    53.00 KB  raw2.product
|  |  10:SCAN HDFS                     2  705.002ms   979.003ms  123.31M    96.52 MB  raw2.purchase_item
|  101:EXCHANGE                        6   29.666ms    56.000ms    4.71M    14.18 MB  HASH(purchaseid)
|  17:HASH JOIN                        5  223.400ms   297.000ms    4.71M    20.04 MB  INNER JOIN, BROADCAST
|  |--98:EXCHANGE                      5   11.800ms    13.000ms  494.88K   528.00 KB  BROADCAST
|  |  16:SCAN HDFS                     1   68.000ms    68.000ms  494.88K     1.65 MB  raw2.customer
|  15:SCAN HDFS                        5   27.600ms    38.000ms    4.71M    12.23 MB  raw2.purchase
|--27:HASH JOIN                        6    8s510ms    10s286ms  123.34M     2.14 GB  INNER JOIN, PARTITIONED
|  |--107:EXCHANGE                     6    1s010ms     1s307ms  123.34M    14.90 MB  HASH(purchaseid)
|  |  23:NESTED LOOP JOIN              2    4s725ms     5s572ms   63.21B   102.00 KB  CROSS JOIN, BROADCAST
|  |  |--105:EXCHANGE                  2    0.000ns     0.000ns        1    16.00 KB  BROADCAST
|  |  |  20:SCAN HDFS                  1   73.000ms    73.000ms        1    24.00 KB  raw3.store
|  |  22:HASH JOIN                     2    1s999ms     2s366ms  123.34M     1.99 MB  INNER JOIN, BROADCAST
|  |  |--104:EXCHANGE                  2    0.000ns     0.000ns       13    16.00 KB  BROADCAST
|  |  |  21:SCAN HDFS                  1  123.000ms   123.000ms       13    53.00 KB  raw3.product
|  |  19:SCAN HDFS                     2  613.501ms   781.001ms  123.34M    96.52 MB  raw3.purchase_item
|  106:EXCHANGE                        6   32.666ms    49.000ms    4.71M    14.19 MB  HASH(purchaseid)
|  26:HASH JOIN                        6  202.167ms   259.000ms    4.71M    20.04 MB  INNER JOIN, BROADCAST
|  |--103:EXCHANGE                     6   11.666ms    17.000ms  494.88K   960.00 KB  BROADCAST
|  |  25:SCAN HDFS                     1  181.000ms   181.000ms  494.88K     1.65 MB  raw3.customer
|  24:SCAN HDFS                        6   20.500ms    31.000ms    4.71M    12.37 MB  raw3.purchase
|--36:HASH JOIN                        6    8s219ms     9s120ms  123.30M     2.14 GB  INNER JOIN, PARTITIONED
|  |--112:EXCHANGE                     6    1s014ms     1s194ms  123.30M    14.89 MB  HASH(purchaseid)
```

```
|   |   32:NESTED LOOP JOIN      2     5s519ms     7s607ms    63.19B   102.00 KB  CROSS JOIN, BROADCAST
|   |   |--110:EXCHANGE          2     0.000ns     0.000ns         1    16.00 KB  BROADCAST
|   |   |   29:SCAN HDFS         1    65.000ms    65.000ms         1    16.00 KB  raw4.store
|   |   31:HASH JOIN             2     2s282ms     2s971ms   123.30M     1.99 MB  INNER JOIN, BROADCAST
|   |   |--109:EXCHANGE          2     0.000ns     0.000ns        13    16.00 KB  BROADCAST
|   |   |   30:SCAN HDFS         1    79.000ms    79.000ms        13    53.00 KB  raw4.product
|   |   28:SCAN HDFS             2   694.502ms   950.003ms   123.30M    96.52 MB  raw4.purchase_item
|   111:EXCHANGE                 6    28.666ms    43.000ms     4.71M    14.18 MB  HASH(purchaseid)
|   35:HASH JOIN                 6   208.500ms   280.000ms     4.71M    20.04 MB  INNER JOIN, BROADCAST
|   |--108:EXCHANGE              6    11.666ms    16.000ms   494.88K   384.00 KB  BROADCAST
|   |   34:SCAN HDFS             1    56.000ms    56.000ms   494.88K     1.65 MB  raw4.customer
|   33:SCAN HDFS                 6    24.333ms    36.000ms     4.71M    12.43 MB  raw4.purchase
|--45:HASH JOIN                  6     8s286ms     9s298ms   123.19M     2.14 GB  INNER JOIN, PARTITIONED
|   |--117:EXCHANGE              6     1s073ms     1s228ms   123.19M    14.89 MB  HASH(purchaseid)
|   |   41:NESTED LOOP JOIN      2     4s697ms     6s076ms    63.13B   102.00 KB  CROSS JOIN, BROADCAST
|   |   |--115:EXCHANGE          2     0.000ns     0.000ns         1    16.00 KB  BROADCAST
|   |   |   38:SCAN HDFS         1    65.000ms    65.000ms         1    24.00 KB  raw5.store
|   |   40:HASH JOIN             2     1s961ms     2s474ms   123.19M     1.99 MB  INNER JOIN, BROADCAST
|   |   |--114:EXCHANGE          2     0.000ns     0.000ns        13    16.00 KB  BROADCAST
|   |   |   39:SCAN HDFS         1   128.000ms   128.000ms        13    53.00 KB  raw5.product
|   |   37:SCAN HDFS             2   723.501ms   853.001ms   123.19M    96.52 MB  raw5.purchase_item
|   116:EXCHANGE                 6    29.833ms    39.000ms     4.71M    14.16 MB  HASH(purchaseid)
|   44:HASH JOIN                 5   234.600ms   306.000ms     4.71M    20.04 MB  INNER JOIN, BROADCAST
|   |--113:EXCHANGE              5    12.200ms    17.000ms   494.88K     1.20 MB  BROADCAST
|   |   43:SCAN HDFS             1   147.000ms   147.000ms   494.88K     1.65 MB  raw5.customer
|   42:SCAN HDFS                 5    24.600ms    38.000ms     4.71M    12.38 MB  raw5.purchase
|--54:HASH JOIN                  6     8s468ms    10s034ms   123.25M     2.14 GB  INNER JOIN, PARTITIONED
|   |--122:EXCHANGE              6   995.669ms     1s277ms   123.25M    14.89 MB  HASH(purchaseid)
|   |   50:NESTED LOOP JOIN      2     4s495ms     5s441ms    63.16B   102.00 KB  CROSS JOIN, BROADCAST
|   |   |--120:EXCHANGE          2     0.000ns     0.000ns         1    16.00 KB  BROADCAST
|   |   |   47:SCAN HDFS         1    72.000ms    72.000ms         1    24.00 KB  raw6.store
|   |   49:HASH JOIN             2     1s956ms     2s410ms   123.25M     1.99 MB  INNER JOIN, BROADCAST
|   |   |--119:EXCHANGE          2     0.000ns     0.000ns        13    16.00 KB  BROADCAST
|   |   |   48:SCAN HDFS         1    66.000ms    66.000ms        13    53.00 KB  raw6.product
|   |   46:SCAN HDFS             2   738.001ms     1s042ms   123.25M    96.52 MB  raw6.purchase_item
|   121:EXCHANGE                 6    31.500ms    48.000ms     4.71M    14.16 MB  HASH(purchaseid)
|   53:HASH JOIN                 5   187.000ms   256.000ms     4.71M    20.04 MB  INNER JOIN, BROADCAST
|   |--118:EXCHANGE              5    13.800ms    18.000ms   494.88K   544.00 KB  BROADCAST
|   |   52:SCAN HDFS             1   115.000ms   115.000ms   494.88K     1.65 MB  raw6.customer
|   51:SCAN HDFS                 5    25.600ms    32.000ms     4.71M    12.43 MB  raw6.purchase
|--63:HASH JOIN                  6     8s333ms     9s642ms   123.29M     2.14 GB  INNER JOIN, PARTITIONED
|   |--127:EXCHANGE              6   990.836ms     1s288ms   123.29M    14.88 MB  HASH(purchaseid)
|   |   59:NESTED LOOP JOIN      2     4s501ms     5s642ms    63.19B   102.00 KB  CROSS JOIN, BROADCAST
|   |   |--125:EXCHANGE          2     0.000ns     0.000ns         1    16.00 KB  BROADCAST
|   |   |   56:SCAN HDFS         1    62.000ms    62.000ms         1    24.00 KB  raw7.store
|   |   58:HASH JOIN             2     1s953ms     2s394ms   123.29M     1.99 MB  INNER JOIN, BROADCAST
|   |   |--124:EXCHANGE          2     0.000ns     0.000ns        13    16.00 KB  BROADCAST
|   |   |   57:SCAN HDFS         1    67.000ms    67.000ms        13    53.00 KB  raw7.product
|   |   55:SCAN HDFS             2   622.501ms   689.001ms   123.29M    96.48 MB  raw7.purchase_item
|   126:EXCHANGE                 6    34.666ms    71.000ms     4.71M    14.18 MB  HASH(purchaseid)
|   62:HASH JOIN                 6   165.667ms   201.000ms     4.71M    20.04 MB  INNER JOIN, BROADCAST
|   |--123:EXCHANGE              6    11.000ms    18.000ms   494.88K   528.00 KB  BROADCAST
|   |   61:SCAN HDFS             1    75.000ms    75.000ms   494.88K     1.65 MB  raw7.customer
|   60:SCAN HDFS                 6    22.166ms    33.000ms     4.71M    12.39 MB  raw7.purchase
|--72:HASH JOIN                  6     8s510ms    10s476ms   123.25M     2.14 GB  INNER JOIN, PARTITIONED
|   |--132:EXCHANGE              6   979.835ms     1s335ms   123.25M    14.90 MB  HASH(purchaseid)
|   |   68:NESTED LOOP JOIN      2     5s018ms     6s432ms    63.17B   102.00 KB  CROSS JOIN, BROADCAST
|   |   |--130:EXCHANGE          2     0.000ns     0.000ns         1    16.00 KB  BROADCAST
|   |   |   65:SCAN HDFS         1    70.000ms    70.000ms         1    24.00 KB  raw8.store
|   |   67:HASH JOIN             2     2s028ms     2s531ms   123.25M     1.99 MB  INNER JOIN, BROADCAST
|   |   |--129:EXCHANGE          2     0.000ns     0.000ns        13    16.00 KB  BROADCAST
|   |   |   66:SCAN HDFS         1   119.000ms   119.000ms        13    45.00 KB  raw8.product
|   |   64:SCAN HDFS             2   704.502ms   784.003ms   123.25M    96.52 MB  raw8.purchase_item
|   131:EXCHANGE                 6    35.000ms    55.000ms     4.71M    14.15 MB  HASH(purchaseid)
|   71:HASH JOIN                 5   164.800ms   236.000ms     4.71M    20.04 MB  INNER JOIN, BROADCAST
|   |--128:EXCHANGE              5    13.400ms    19.000ms   494.88K   656.00 KB  BROADCAST
|   |   70:SCAN HDFS             1    94.000ms    94.000ms   494.88K     1.66 MB  raw8.customer
|   69:SCAN HDFS                 5    20.200ms    28.000ms     4.71M    12.36 MB  raw8.purchase
|--81:HASH JOIN                  6     8s440ms    10s339ms   123.29M     2.14 GB  INNER JOIN, PARTITIONED
|   |--137:EXCHANGE              6   961.335ms     1s215ms   123.29M    14.89 MB  HASH(purchaseid)
|   |   77:NESTED LOOP JOIN      2     4s899ms     6s292ms    63.19B   102.00 KB  CROSS JOIN, BROADCAST
|   |   |--135:EXCHANGE          2     0.000ns     0.000ns         1    16.00 KB  BROADCAST
|   |   |   74:SCAN HDFS         1    31.000ms    31.000ms         1    24.00 KB  raw9.store
|   |   76:HASH JOIN             2     2s125ms     2s683ms   123.29M     1.99 MB  INNER JOIN, BROADCAST
|   |   |--134:EXCHANGE          2     0.000ns     0.000ns        13    16.00 KB  BROADCAST
|   |   |   75:SCAN HDFS         1    48.000ms    48.000ms        13    53.00 KB  raw9.product
|   |   73:SCAN HDFS             2   633.502ms   766.003ms   123.29M    96.54 MB  raw9.purchase_item
|   136:EXCHANGE                 6    41.666ms    66.000ms     4.71M    14.16 MB  HASH(purchaseid)
|   80:HASH JOIN                 5   179.400ms   220.000ms     4.71M    20.04 MB  INNER JOIN, BROADCAST
|   |--133:EXCHANGE              5    10.800ms    12.000ms   494.88K   672.00 KB  BROADCAST
|   |   79:SCAN HDFS             1   133.000ms   133.000ms   494.88K     1.66 MB  raw9.customer
|   78:SCAN HDFS                 5    24.400ms    43.000ms     4.71M    12.24 MB  raw9.purchase
|--90:HASH JOIN                  6     8s375ms     9s635ms   123.29M     2.14 GB  INNER JOIN, PARTITIONED
|   |--142:EXCHANGE              6   934.169ms     1s061ms   123.29M    14.90 MB  HASH(purchaseid)
|   |   86:NESTED LOOP JOIN      2     6s314ms     6s466ms    63.19B   102.00 KB  CROSS JOIN, BROADCAST
|   |   |--140:EXCHANGE          2     0.000ns     0.000ns         1    16.00 KB  BROADCAST
|   |   |   83:SCAN HDFS         1    74.000ms    74.000ms         1    24.00 KB  raw10.store
|   |   85:HASH JOIN             2     2s261ms     2s614ms   123.29M     1.99 MB  INNER JOIN, BROADCAST
|   |   |--139:EXCHANGE          2     0.000ns     0.000ns        13    16.00 KB  BROADCAST
|   |   |   84:SCAN HDFS         1    67.000ms    67.000ms        13    45.00 KB  raw10.product
|   |   82:SCAN HDFS             2   602.501ms   730.001ms   123.29M    96.51 MB  raw10.purchase_item
|   141:EXCHANGE                 6    39.333ms    74.000ms     4.71M    14.15 MB  HASH(purchaseid)
|   89:HASH JOIN                 5   186.800ms   249.000ms     4.71M    20.04 MB  INNER JOIN, BROADCAST
|   |--138:EXCHANGE              5    10.200ms    16.000ms   494.88K   432.00 KB  BROADCAST
|   |   88:SCAN HDFS             1   102.000ms   102.000ms   494.88K     1.66 MB  raw10.customer
|   87:SCAN HDFS                 5    24.600ms    47.000ms     4.71M    12.38 MB  raw10.purchase

09:HASH JOIN                     6     8s650ms    10s129ms   123.23M     2.14 GB  INNER JOIN, PARTITIONED
```

| | | | | | | |
|---|---|---|---|---|---|---|
| \|--97:EXCHANGE | 6 | 1s024ms | 1s310ms | 123.23M | 14.90 MB | HASH(purchaseid) |
| \| 05:NESTED LOOP JOIN | 2 | 5s487ms | 7s345ms | 63.16B | 102.00 KB | CROSS JOIN, BROADCAST |
| \| \|--95:EXCHANGE | 2 | 0.000ns | 0.000ns | 1 | 16.00 KB | BROADCAST |
| \| \| 02:SCAN HDFS | 1 | 159.000ms | 159.000ms | 1 | 24.00 KB | raw1.store |
| \| 04:HASH JOIN | 2 | 2s229ms | 3s035ms | 123.23M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--94:EXCHANGE | 2 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| \| 03:SCAN HDFS | 1 | 128.000ms | 128.000ms | 13 | 53.00 KB | raw1.product |
| \| 01:SCAN HDFS | 2 | 857.002ms | 1s087ms | 123.23M | 96.52 MB | raw1.purchase_item |
| 96:EXCHANGE | 6 | 27.500ms | 42.000ms | 4.71M | 14.14 MB | HASH(purchaseid) |
| 08:HASH JOIN | 4 | 248.250ms | 317.001ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \|--93:EXCHANGE | 4 | 11.500ms | 13.000ms | 494.88K | 304.00 KB | BROADCAST |
| \| 07:SCAN HDFS | 1 | 147.000ms | 147.000ms | 494.88K | 1.65 MB | raw1.customer |
| 06:SCAN HDFS | 4 | 28.000ms | 50.000ms | 4.71M | 14.42 MB | raw1.purchase |

Quelltext A.2: Partitioniertes virtuelles Data Warehouse, Laufzeit: 85 Sekunden

| Operator | #Hosts | Avg Time | Max Time | #Rows | Peak Mem | Detail |
|---|---|---|---|---|---|---|
| 135:MERGING-EXCHANGE | 1 | 0.000ns | 0.000ns | 13 | 96.00 KB | UNPARTITIONED |
| 92:SORT | 6 | 500.001us | 1.000ms | 13 | 12.02 MB | |
| 134:AGGREGATE | 6 | 2.833ms | 4.000ms | 13 | 1.97 MB | FINALIZE |
| 133:EXCHANGE | 6 | 0.000ns | 0.000ns | 78 | 56.00 KB | HASH(pname) |
| 91:AGGREGATE | 6 | 9s848ms | 10s658ms | 78 | 2.11 MB | STREAMING |
| 00:UNION | 6 | 10s060ms | 10s917ms | 1.23B | 160.00 KB | |
| \|--18:HASH JOIN | 6 | 2s179ms | 2s405ms | 123.31M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--100:EXCHANGE | 6 | 171.333ms | 282.001ms | 4.71M | 11.01 MB | BROADCAST |
| \| \| 17:HASH JOIN | 5 | 186.800ms | 212.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--99:EXCHANGE | 5 | 16.600ms | 22.000ms | 494.88K | 304.00 KB | BROADCAST |
| \| \| \| 16:SCAN HDFS | 1 | 114.000ms | 114.000ms | 494.88K | 1.66 MB | raw2.customer |
| \| \| 15:SCAN HDFS | 5 | 31.200ms | 61.000ms | 4.71M | 16.24 MB | raw2.purchase |
| \| 14:NESTED LOOP JOIN | 6 | 1s111ms | 1s199ms | 123.31M | 102.00 KB | CROSS JOIN, BROADCAST |
| \| \|--98:EXCHANGE | 6 | 0.000ns | 0.000ns | 1 | 16.00 KB | BROADCAST |
| \| \| 11:SCAN HDFS | 1 | 80.000ms | 80.000ms | 1 | 24.00 KB | raw2.store |
| \| 13:HASH JOIN | 6 | 516.001ms | 550.001ms | 123.31M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--97:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| \| 12:SCAN HDFS | 1 | 59.000ms | 59.000ms | 13 | 53.00 KB | raw2.product |
| \| 10:SCAN HDFS | 6 | 170.167ms | 190.000ms | 123.31M | 5.44 MB | raw2.purchase_item |
| \|--27:HASH JOIN | 6 | 2s011ms | 2s245ms | 123.34M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--104:EXCHANGE | 6 | 214.000ms | 499.000ms | 4.71M | 11.01 MB | BROADCAST |
| \| \| 26:HASH JOIN | 6 | 188.167ms | 228.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--103:EXCHANGE | 6 | 13.333ms | 16.000ms | 494.88K | 544.00 KB | BROADCAST |
| \| \| \| 25:SCAN HDFS | 1 | 38.000ms | 38.000ms | 494.88K | 1.66 MB | raw3.customer |
| \| \| 24:SCAN HDFS | 6 | 24.166ms | 55.000ms | 4.71M | 12.42 MB | raw3.purchase |
| \| 23:NESTED LOOP JOIN | 6 | 1s096ms | 1s169ms | 123.34M | 102.00 KB | CROSS JOIN, BROADCAST |
| \| \|--102:EXCHANGE | 6 | 166.667us | 1.000ms | 1 | 16.00 KB | BROADCAST |
| \| \| 20:SCAN HDFS | 1 | 47.000ms | 47.000ms | 1 | 24.00 KB | raw3.store |
| \| 22:HASH JOIN | 6 | 525.668ms | 566.001ms | 123.34M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--101:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| \| 21:SCAN HDFS | 1 | 71.000ms | 71.000ms | 13 | 53.00 KB | raw3.product |
| \| 19:SCAN HDFS | 6 | 168.667ms | 192.000ms | 123.34M | 9.02 MB | raw3.purchase_item |
| \|--36:HASH JOIN | 6 | 1s979ms | 2s143ms | 123.30M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--108:EXCHANGE | 6 | 172.667ms | 223.000ms | 4.71M | 10.98 MB | BROADCAST |
| \| \| 35:HASH JOIN | 5 | 194.600ms | 255.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--107:EXCHANGE | 5 | 13.200ms | 17.000ms | 494.88K | 576.00 KB | BROADCAST |
| \| \| \| 34:SCAN HDFS | 1 | 73.000ms | 73.000ms | 494.88K | 1.65 MB | raw4.customer |
| \| \| 33:SCAN HDFS | 5 | 22.200ms | 32.000ms | 4.71M | 16.23 MB | raw4.purchase |
| \| 32:NESTED LOOP JOIN | 6 | 1s147ms | 1s304ms | 123.30M | 102.00 KB | CROSS JOIN, BROADCAST |
| \| \|--106:EXCHANGE | 6 | 0.000ns | 0.000ns | 1 | 16.00 KB | BROADCAST |
| \| \| 29:SCAN HDFS | 1 | 35.000ms | 35.000ms | 1 | 24.00 KB | raw4.store |
| \| 31:HASH JOIN | 6 | 536.001ms | 589.001ms | 123.30M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--105:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| \| 30:SCAN HDFS | 1 | 75.000ms | 75.000ms | 13 | 53.00 KB | raw4.product |
| \| 28:SCAN HDFS | 6 | 177.333ms | 196.000ms | 123.30M | 9.01 MB | raw4.purchase_item |
| \|--45:HASH JOIN | 6 | 2s410ms | 4s445ms | 123.19M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--112:EXCHANGE | 6 | 235.833ms | 467.000ms | 4.71M | 11.05 MB | BROADCAST |
| \| \| 44:HASH JOIN | 5 | 206.600ms | 270.001ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--111:EXCHANGE | 5 | 17.200ms | 25.000ms | 494.88K | 6.19 MB | BROADCAST |
| \| \| \| 43:SCAN HDFS | 1 | 45.000ms | 45.000ms | 494.88K | 1.66 MB | raw5.customer |
| \| \| 42:SCAN HDFS | 5 | 24.600ms | 39.000ms | 4.71M | 16.24 MB | raw5.purchase |
| \| 41:NESTED LOOP JOIN | 6 | 1s896ms | 5s982ms | 123.19M | 102.00 KB | CROSS JOIN, BROADCAST |
| \| \|--110:EXCHANGE | 6 | 0.000ns | 0.000ns | 1 | 16.00 KB | BROADCAST |
| \| \| 38:SCAN HDFS | 1 | 108.000ms | 108.000ms | 1 | 24.00 KB | raw5.store |
| \| 40:HASH JOIN | 6 | 643.001ms | 1s358ms | 123.19M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--109:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| \| 39:SCAN HDFS | 1 | 58.000ms | 58.000ms | 13 | 53.00 KB | raw5.product |
| \| 37:SCAN HDFS | 6 | 189.000ms | 223.000ms | 123.19M | 9.02 MB | raw5.purchase_item |
| \|--54:HASH JOIN | 6 | 2s217ms | 2s512ms | 123.25M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--116:EXCHANGE | 6 | 217.833ms | 356.000ms | 4.71M | 11.05 MB | BROADCAST |
| \| \| 53:HASH JOIN | 5 | 203.800ms | 250.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--115:EXCHANGE | 5 | 13.400ms | 17.000ms | 494.88K | 2.06 MB | BROADCAST |
| \| \| \| 52:SCAN HDFS | 1 | 101.000ms | 101.000ms | 494.88K | 1.66 MB | raw6.customer |
| \| \| 51:SCAN HDFS | 5 | 27.800ms | 49.000ms | 4.71M | 12.35 MB | raw6.purchase |
| \| 50:NESTED LOOP JOIN | 6 | 1s172ms | 1s671ms | 123.25M | 102.00 KB | CROSS JOIN, BROADCAST |
| \| \|--114:EXCHANGE | 6 | 166.667us | 1.000ms | 1 | 16.00 KB | BROADCAST |
| \| \| 47:SCAN HDFS | 1 | 45.000ms | 45.000ms | 1 | 24.00 KB | raw6.store |
| \| 49:HASH JOIN | 6 | 511.334ms | 577.001ms | 123.25M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--113:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| \| \| 48:SCAN HDFS | 1 | 40.000ms | 40.000ms | 13 | 53.00 KB | raw6.product |
| \| 46:SCAN HDFS | 6 | 206.333ms | 245.000ms | 123.25M | 12.62 MB | raw6.purchase_item |
| \|--63:HASH JOIN | 6 | 2s135ms | 2s354ms | 123.29M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--120:EXCHANGE | 6 | 210.000ms | 288.001ms | 4.71M | 11.05 MB | BROADCAST |
| \| \| 62:HASH JOIN | 5 | 193.000ms | 264.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--119:EXCHANGE | 5 | 12.600ms | 14.000ms | 494.88K | 608.00 KB | BROADCAST |
| \| \| \| 61:SCAN HDFS | 1 | 119.000ms | 119.000ms | 494.88K | 1.65 MB | raw7.customer |
| \| \| 60:SCAN HDFS | 5 | 23.800ms | 41.000ms | 4.71M | 12.24 MB | raw7.purchase |
| \| 59:NESTED LOOP JOIN | 6 | 1s073ms | 1s207ms | 123.29M | 106.00 KB | CROSS JOIN, BROADCAST |
| \| \|--118:EXCHANGE | 6 | 0.000ns | 0.000ns | 1 | 16.00 KB | BROADCAST |
| \| \| 56:SCAN HDFS | 1 | 81.000ms | 81.000ms | 1 | 24.00 KB | raw7.store |
| \| 58:HASH JOIN | 6 | 511.001ms | 616.001ms | 123.29M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--117:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| \| 57:SCAN HDFS | 1 | 85.000ms | 85.000ms | 13 | 53.00 KB | raw7.product |
| \| 55:SCAN HDFS | 6 | 209.500ms | 248.000ms | 123.29M | 16.16 MB | raw7.purchase_item |
| \|--72:HASH JOIN | 6 | 2s253ms | 2s545ms | 123.25M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--124:EXCHANGE | 6 | 222.333ms | 403.000ms | 4.71M | 10.99 MB | BROADCAST |
| \| \| 71:HASH JOIN | 5 | 193.200ms | 234.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--123:EXCHANGE | 5 | 13.400ms | 15.000ms | 494.88K | 1.00 MB | BROADCAST |
| \| \| \| 70:SCAN HDFS | 1 | 69.000ms | 69.000ms | 494.88K | 1.66 MB | raw8.customer |
| \| \| 69:SCAN HDFS | 5 | 24.600ms | 38.000ms | 4.71M | 12.38 MB | raw8.purchase |
| \| 68:NESTED LOOP JOIN | 6 | 1s105ms | 1s434ms | 123.25M | 102.00 KB | CROSS JOIN, BROADCAST |
| \| \|--122:EXCHANGE | 6 | 0.000ns | 0.000ns | 1 | 16.00 KB | BROADCAST |
| \| \| 65:SCAN HDFS | 1 | 134.000ms | 134.000ms | 1 | 24.00 KB | raw8.store |
| \| 67:HASH JOIN | 6 | 496.168ms | 554.001ms | 123.25M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--121:EXCHANGE | 6 | 166.667us | 1.000ms | 13 | 16.00 KB | BROADCAST |
| \| \| 66:SCAN HDFS | 1 | 58.000ms | 58.000ms | 13 | 53.00 KB | raw8.product |
| \| 64:SCAN HDFS | 6 | 200.167ms | 238.000ms | 123.25M | 16.20 MB | raw8.purchase_item |
| \|--81:HASH JOIN | 6 | 2s121ms | 2s280ms | 123.29M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--128:EXCHANGE | 6 | 191.667ms | 343.000ms | 4.71M | 11.02 MB | BROADCAST |
| \| \| 80:HASH JOIN | 5 | 197.600ms | 238.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--127:EXCHANGE | 5 | 14.600ms | 20.000ms | 494.88K | 272.00 KB | BROADCAST |
| \| \| \| 79:SCAN HDFS | 1 | 71.000ms | 71.000ms | 494.88K | 1.65 MB | raw9.customer |
| \| \| 78:SCAN HDFS | 5 | 22.800ms | 36.000ms | 4.71M | 12.42 MB | raw9.purchase |
| \| 77:NESTED LOOP JOIN | 6 | 1s133ms | 1s407ms | 123.29M | 106.00 KB | CROSS JOIN, BROADCAST |
| \| \|--126:EXCHANGE | 6 | 166.667us | 1.000ms | 1 | 16.00 KB | BROADCAST |
| \| \| 74:SCAN HDFS | 1 | 40.000ms | 40.000ms | 1 | 24.00 KB | raw9.store |
| \| 76:HASH JOIN | 6 | 494.668ms | 549.001ms | 123.29M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--125:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| \| 75:SCAN HDFS | 1 | 52.000ms | 52.000ms | 13 | 53.00 KB | raw9.product |
| \| 73:SCAN HDFS | 6 | 216.167ms | 244.000ms | 123.29M | 23.33 MB | raw9.purchase_item |
| \|--90:HASH JOIN | 6 | 2s300ms | 2s408ms | 123.29M | 192.13 MB | INNER JOIN, BROADCAST |
| \| \|--132:EXCHANGE | 6 | 233.000ms | 353.000ms | 4.71M | 10.98 MB | BROADCAST |
| \| \| 89:HASH JOIN | 5 | 189.200ms | 235.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \| \|--131:EXCHANGE | 5 | 15.000ms | 19.000ms | 494.88K | 1.03 MB | BROADCAST |
| \| \| \| 88:SCAN HDFS | 1 | 117.000ms | 117.000ms | 494.88K | 1.66 MB | raw10.customer |
| \| \| 87:SCAN HDFS | 5 | 30.200ms | 54.000ms | 4.71M | 12.33 MB | raw10.purchase |
| \| 86:NESTED LOOP JOIN | 6 | 1s044ms | 1s153ms | 123.29M | 102.00 KB | CROSS JOIN, BROADCAST |
| \| \|--130:EXCHANGE | 6 | 0.000ns | 0.000ns | 1 | 16.00 KB | BROADCAST |
| \| \| 83:SCAN HDFS | 1 | 46.000ms | 46.000ms | 1 | 24.00 KB | raw10.store |
| \| 85:HASH JOIN | 6 | 499.668ms | 532.001ms | 123.29M | 1.99 MB | INNER JOIN, BROADCAST |
| \| \|--129:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| \| 84:SCAN HDFS | 1 | 79.000ms | 79.000ms | 13 | 53.00 KB | raw10.product |
| \| 82:SCAN HDFS | 6 | 217.667ms | 262.000ms | 123.29M | 30.46 MB | raw10.purchase_item |
| 09:HASH JOIN | 6 | 1s962ms | 2s332ms | 123.23M | 192.13 MB | INNER JOIN, BROADCAST |
| \|--96:EXCHANGE | 6 | 170.500ms | 231.000ms | 4.71M | 10.99 MB | BROADCAST |
| \| 08:HASH JOIN | 5 | 213.400ms | 264.000ms | 4.71M | 20.04 MB | INNER JOIN, BROADCAST |
| \| \|--95:EXCHANGE | 5 | 11.000ms | 13.000ms | 494.88K | 1.01 MB | BROADCAST |
| \| \| 07:SCAN HDFS | 1 | 79.000ms | 79.000ms | 494.88K | 1.65 MB | raw1.customer |
| \| 06:SCAN HDFS | 5 | 22.800ms | 37.000ms | 4.71M | 12.41 MB | raw1.purchase |
| 05:NESTED LOOP JOIN | 6 | 1s190ms | 1s662ms | 123.23M | 102.00 KB | CROSS JOIN, BROADCAST |
| \|--94:EXCHANGE | 6 | 0.000ns | 0.000ns | 1 | 16.00 KB | BROADCAST |
| \| 02:SCAN HDFS | 1 | 64.000ms | 64.000ms | 1 | 24.00 KB | raw1.store |
| 04:HASH JOIN | 6 | 526.501ms | 580.001ms | 123.23M | 1.99 MB | INNER JOIN, BROADCAST |
| \|--93:EXCHANGE | 6 | 0.000ns | 0.000ns | 13 | 16.00 KB | BROADCAST |
| \| 03:SCAN HDFS | 1 | 126.000ms | 126.000ms | 13 | 53.00 KB | raw1.product |
| 01:SCAN HDFS | 6 | 650.501ms | 2s062ms | 123.23M | 5.44 MB | raw1.purchase_item |

## A.2 Einflüsse der Anzahl der Quellen

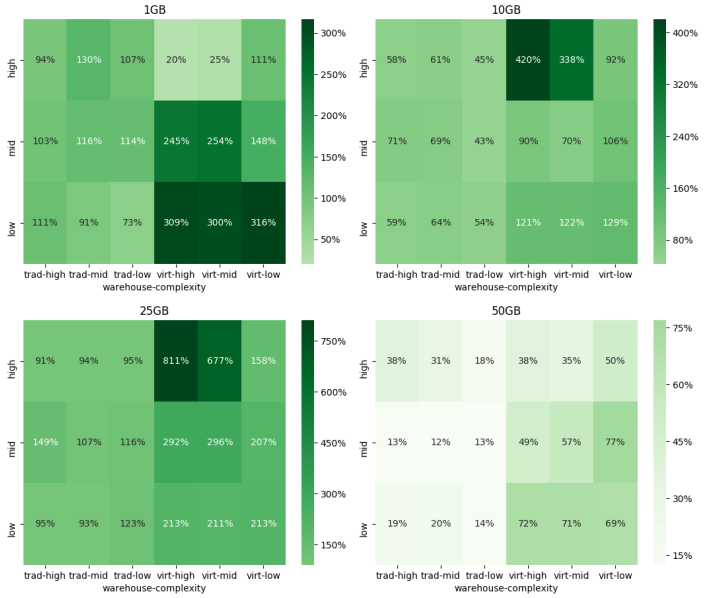



Abbildung A.1: Einfluss Quellen; ohne Partitionierung

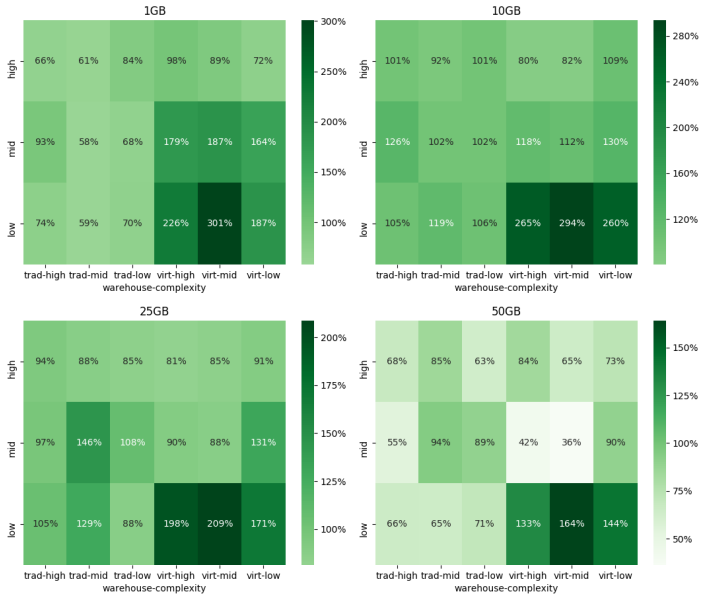



Abbildung A.2: Einfluss Quellen; mit Partitionierung nach Jahr / Monat

## A.3 Einflüsse der Partitionierung

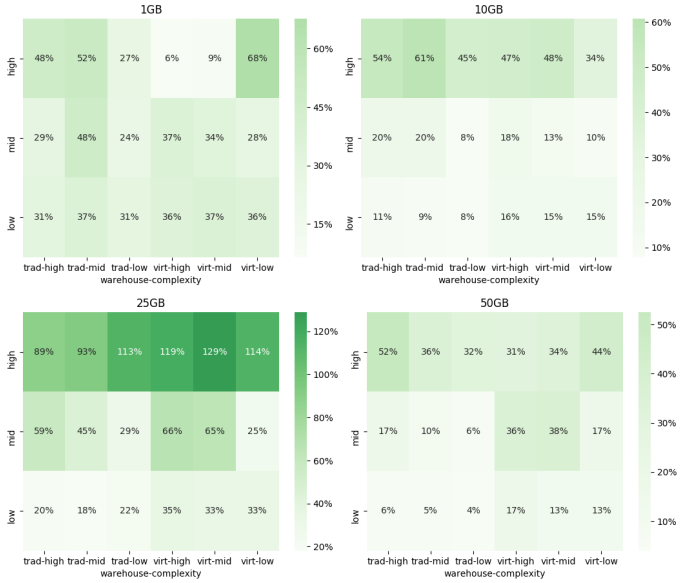



Abbildung A.3: Einfluss Partitionierung; mit drei Quellen

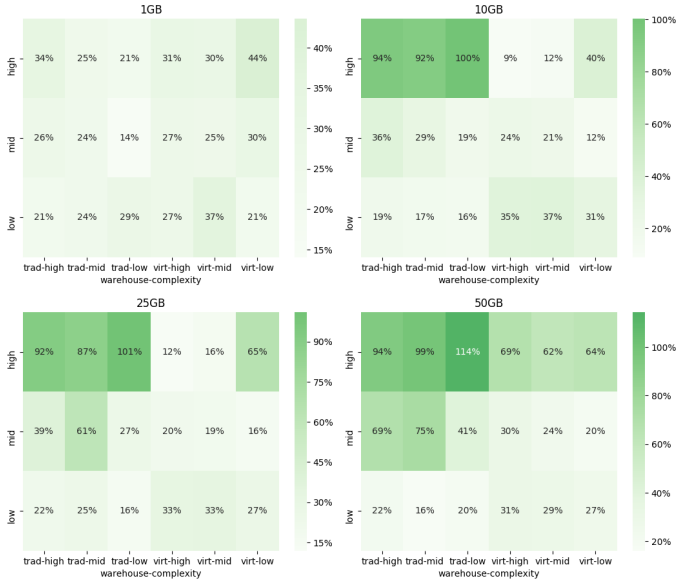



Abbildung A.4: Einfluss Partitionierung; mit zehn Quellen

## A.4 Ablaufpläne virtuelles Data Warehouse

Nachfolgend soll der Komplexitätsunterschied der Ablaufpläne zwischen dem virtuellen und dem traditionellen Data Warehouse anhand einer Abfrage verdeutlicht werden.

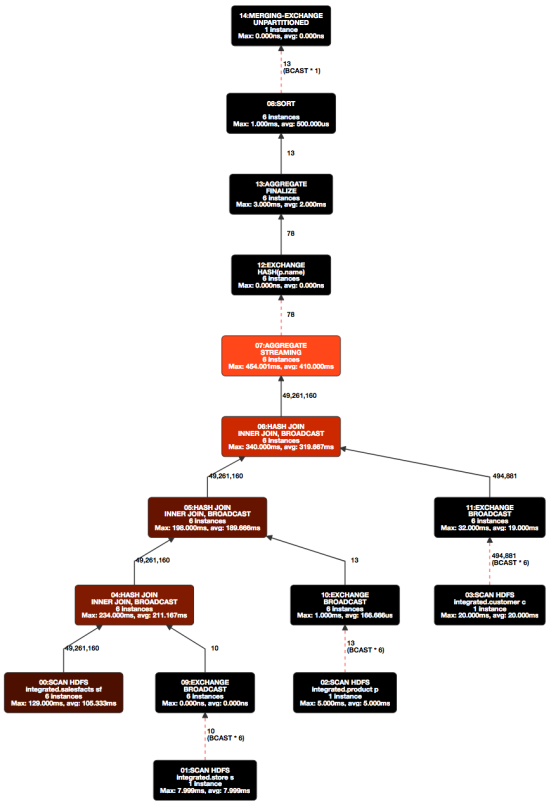



Abbildung A.5: Ablaufplan traditionelles Data Warehouse bei hohem Volumen, hoher Komplexität und zehn Quellsystemen

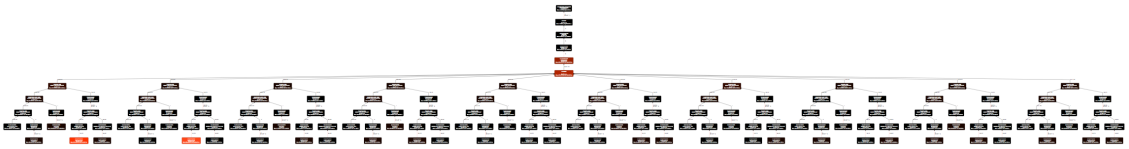


Abbildung A.6: Ablaufplan virtuelles Data Warehouse bei hohem Volumen, hoher Komplexität und zehn Quellsystemen

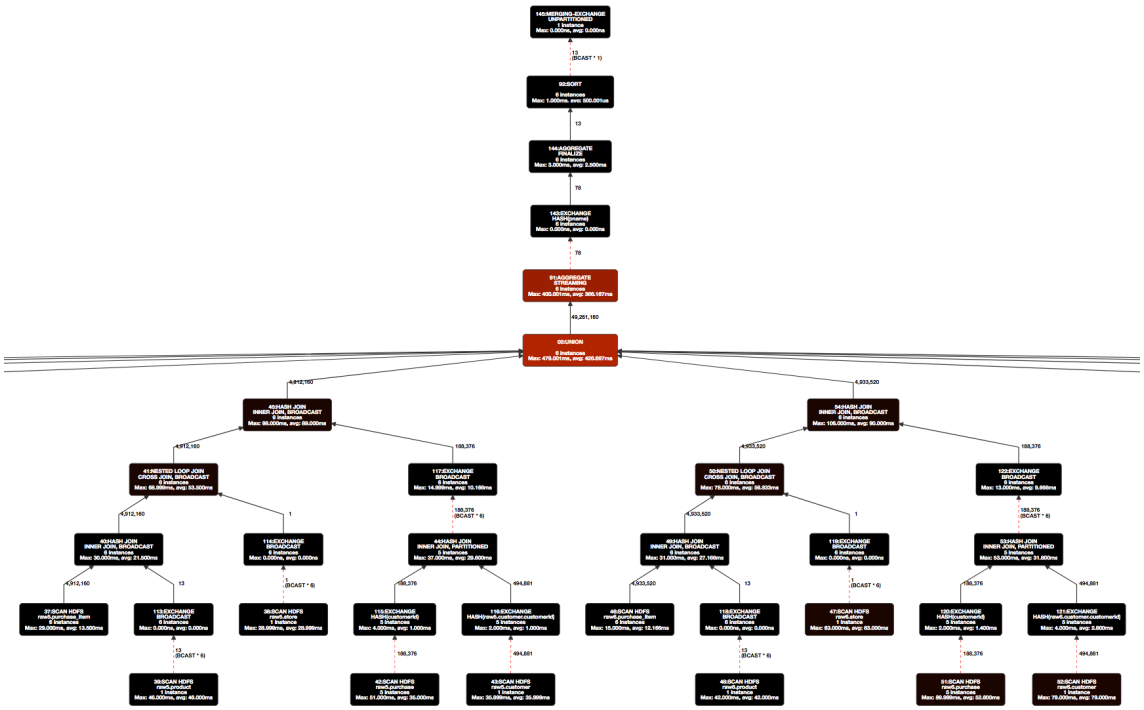


Abbildung A.7: Ablaufplan virtuelles Data Warehouse bei hohem Volumen, hoher Komplexität und zehn Quellsystemen; Vergrößert

# Literaturverzeichnis


[Breslin 2004] Breslin, Mary W.: Data Warehousing Battle of the Giants: Comparing the Basics of the Kimball and Inmon Models. In: Business Intelligence Journal 9 (2004) (zitiert auf Seite 12, 16 und 17)

[Dean und Ghemawat 2004] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004, S. 137–150 (zitiert auf Seite 33 und 34)

[Floratou u. a. 2014] Floratou, Avrilia ; Minhas, Umar F. ; Özcan, Fatma: SQL-on-Hadoop: Full Circle Back to Shared-Nothing Database Architectures. In: PVLDB 7 (2014), S. 1295–1306 (zitiert auf Seite 38)

[Gluchowski und Chamoni 2016] Gluchowski, Peter ; Chamoni, Peter: Analytische Informationssysteme: Business Intelligence-Technologien und -Anwendungen. 5th. Springer Gabler, 2016 (zitiert auf Seite 1, 17, 20, 21, 25, 28 und 29)

[Grover u. a. 2015] Grover, Mark ; Malaska, Ted ; Seidman, Jonathan ; Shapira, Gwen: Hadoop Application Architectures. 1st. O'Reilly Media, Inc., 2015 (zitiert auf Seite 10, 11 und 34)

[Inmon 1996] Inmon, W.H.: Building the data warehouse. Wiley, 1996 (zitiert auf Seite 7 und 12)

[Kimball und Ross 2013] Kimball, Ralph ; Ross, Margy: The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. 3rd. Wiley Publishing, 2013 (zitiert auf Seite 13, 14, 15, 18, 19 und 20)

[Köppen u. a. 2012] Köppen, Veit ; Saake, Gunter ; Sattler, Kai-Uwe: Data Warehouse Technologien. mitp, 2012 (zitiert auf Seite 6, 9, 10, 15, 18, 19, 20, 26 und 31)

[Kornacker u. a. 2015] Kornacker, Marcel ; Behm, Alexander ; Bittorf, Victor ; Bobrovytsky, Taras ; Ching, Casey ; Choi, Alan ; Erickson, Justin ; Grund, Martin ; Hecht, Daniel ; Jacobs, Matthew ; Joshi, Ishaan ; Kuff, Lenni ; Kumar, Dileep ; Leblang, Alex ; Li, Nong ; Pandis, Ippokratis ; Robinson, Henry ; Rorke, David ; Rus, Silvius ; Russell, John ; Tsirogiannis, Dimitris ; Wanderman-Milne, Skye ; Yoder, Michael: Impala: A Modern, Open-Source SQL Engine for Hadoop. In: CIDR, 2015 (zitiert auf Seite 34 und 38)

[Laney 2001] Laney, Douglas: 3D Data Management: Controlling Data Volume, Velocity, and Variety. February 2001. – Forschungsbericht (zitiert auf Seite 32)

[van der Lans 2012] Lans, Rick van der: Data Virtualization for Business Intelligence Systems: Revolutionizing Data Integration for Data Warehouses. 1st. Morgan Kaufmann Publishers Inc., 2012 (zitiert auf Seite 1, 8, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31 und 46)

[Linstedt und Olschimke 2015] Linstedt, Dan ; Olschimke, Michael: Building a Scalable Data Warehouse with Data Vault 2.0. 1st. Morgan Kaufmann Publishers Inc., 2015 (zitiert auf Seite 21)

[Melnik u. a. 2010] Melnik, Sergey ; Gubarev, Andrey ; Long, Jing J. ; Romer, Geoffrey ; Shivakumar, Shiva ; Tolton, Matt ; Vassilakis, Theo: Dremel: Interactive Analysis of Web-Scale Datasets. In: Proc. of the 36th Int'l Conf on Very Large Data Bases, 2010, S. 330–339 (zitiert auf Seite 34)

[NIST ] NIST: NIST Big Data Working Group (NBD-WG). https://bigdatawg.nist.gov/home.php. – Zugegriffen: 19.09.2018 (zitiert auf Seite 33)

[The Aberdeen Group 2011] The Aberdeen Group: Agile BI benchmark report;. 2011 (zitiert auf Seite 22 und 23)

[Wang und Ng 2010] Wang, G. ; Ng, T. S. E.: The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In: 2010 Proceedings IEEE INFOCOM, March 2010, S. 1–9 (zitiert auf Seite 24)

[Ward und Barker 2013] Ward, Jonathan S. ; Barker, Adam: Undefined By Data: A Survey of Big Data Definitions. In: CoRR abs/1309.5821 (2013) (zitiert auf Seite 10, 32 und 33)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 15. Oktober 2018