JOHANNES KEPLER
UNIVERSITÄT LINZ

Konzeption und Umsetzung von RDF-Summarization-Cubes in SPARK-SQL für das Profiling von schema.org-Daten

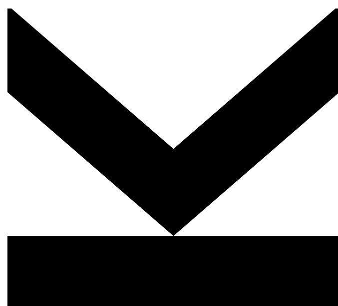
Eingereicht von
Roland Buschberger, BSc

Angefertigt am
Institut für Wirtschaftsinformatik - Data & Knowledge Engineering

Betreuer
o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Mitbetreuung
Dr. Bernd Neumayr

Oktober 2018



Masterarbeit
zur Erlangung des akademischen Grades
Master of Science
im Masterstudium
Wirtschaftsinformatik

JOHANNES KEPLER UNIVERSITÄT LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, am 30.10.2018 Roland Buschberger, BSc

Kurzfassung

Das von den führenden Suchmaschinenanbietern ins Leben gerufene schema.org-Vokabular erlaubt die semantische Beschreibung einer Web-Seite in maschinenlesbarer Form. Um im Rahmen von Data-Profiling die tatsächliche Verwendung des schema.org-Vokabulars untersuchen zu können, wird in dieser Arbeit ein entsprechender Entwurf erarbeitet und umgesetzt. Für die Analyse der Verwendung des schema.org-Vokabulars werden die von Web-Data-Commons veröffentlichten, aus den Korpussen von Common-Crawl extrahierten strukturierten Web-Daten herangezogen. Aus den Rohdaten werden Primärfakten erstellt, welche die schema.org-Klassenhierarchie als semantische Dimensionen enthalten. Ausgehend von den Primärfakten werden die Umsetzungsvarianten "Cube" und "Star" (zusammengefasst als RDF-Summarization-Cubes) erzeugt, welche die entsprechenden schema.org-Analysen ermöglichen. Die Umsetzung wird in PySpark-SQL implementiert und in einem Hadoop-Cluster einer Proof-of-Concept-Umgebung entwickelt und getestet. Die ersten Analysen in Bezug auf die strukturierten Web-Daten-Formate Microdata, JSON-LD und RDFa auf Basis eines relativ kleinen Ausschnittes von Web-Data-Commons haben ergeben, dass in JSON-LD am öftesten schema.org-Klassen verwendet werden. Um die Skalierbarkeit der Umsetzung zu überprüfen, wurde der Proof-of-Concept-Prototyp mithilfe der Plattform Databricks in der Microsoft-Azure-Cloud auf eine größere Datenmenge ausgeführt.

Abstract

The schema.org vocabulary was developed by the leading search engine operators to facilitate the semantic annotation of websites in a machine readable and consistent way. In this thesis a data profiling approach for the analysis of the usage of the schema.org vocabulary is developed. The analysis is based on the data from Web Data Commons which extract structured data from the Common Crawl corpuses. The schema.org hierarchy will embed into the created primary fact tables by the usage of semantic dimensions. On the basis of the primary fact tables the variants "Cube" and "Star" (collectively referred to as RDF-Summarization-Cubes) are developed to enable proper schema.org analysis. The proof-of-concept prototype is implemented in PySpark SQL and is executed and tested in a proof-of-concept Hadoop cluster. The first analysis of the usage of the structured data formats Microdata, JSON-LD and RDFa based on a rather small fragment of web data commons shows that the schema.org vocabulary is used most frequently with the JSON-LD format. To show its scalability the proof-of-concept prototype was also deployed in the Microsoft Azure Cloud using Databricks and executed on a larger fragment of the web data commons corpus.

Inhaltsverzeichnis

1 Einleitung **12**
- 1.1 Motivation . . . 12
- 1.2 Aufgabenstellung . . . 14
- 1.3 Aufbau der Arbeit . . . 15

2 Grundlagen **17**
- 2.1 Data-Profiling . . . 17
- 2.2 Strukturierte Daten im Web und schema.org . . . 19
- 2.3 Web-Data-Commons und Common-Crawl . . . 21
- 2.4 Data-Warehousing und semantische Dimensionen . . . 23
- 2.5 Big-Data-Technologien . . . 30

3 Entwurf **41**
- 3.1 Primärfakten . . . 43
- 3.2 Erweiterte schema.org-Hierarchie . . . 45
- 3.3 Variante 1 “Cube” - einfach und voraggregiert . . . 50
- 3.4 Variante 2 “Star” - für flexible Abfragen . . . 52
- 3.5 Zusammenfassung . . . 55

4 Umsetzung **57**
- 4.1 Proof-of-Concept-Umgebung . . . 57
- 4.2 Vorbereitungen . . . 58
- 4.3 Erstellung von Faktentabellen . . . 60
- 4.4 Erstellung von Hierarchietabellen . . . 65
- 4.5 Umsetzung von Variante 1 . . . 70
- 4.6 Umsetzung von Variante 2 . . . 75
- 4.7 Zusammenfassung . . . 81

5 Auswertung und Evaluierung **82**
- 5.1 Vergleich der Ergebnisse von Variante 1 und Variante 2 . . . 83

5.2 Verwendung von schema.org-Klassen in Microdata, JSON-LD und RDFa . . . 86
5.3 Beispiel-Abfragen auf RDF-Summarization-Cubes . . . 99
5.4 Skalierbarkeit - Umsetzung in der Cloud mit Microsoft Azure und Databricks . . . 103
5.5 Zusammenfassung . . . 110

6 Fazit **111**
6.1 Zusammenfassung und Schlussfolgerung . . . 111
6.2 Ausblick . . . 112

Abbildungsverzeichnis

2.1 Eine Domain-Ontologie in RDF 28
2.2 Eine Aggregations-Ontologie in RDF 29
2.3 Zusammenhang von Spark und Spark-SQL 40

3.1 Übersicht der Umsetzung und Abgrenzung der Arbeit 42
3.2 Primärfakten konzeptionelles Schema in UML 45
3.3 Beispiel zu Primärfakten als UML-Objektdiagramm 46
3.4 Erweiterte schema.org-Hierarchie 47
3.5 Einschränkungen der erweiterten schema.org-Hierarchie 48
3.6 Transitiv-reflexive Hülle der erweiterten schema.org-Hierarchie 49
3.7 Erweiterte schema.org-Hierarchie unterteilt in Ebenen 50
3.8 Konzeptuelles Modell Ressource-Cube (Variante 1) 51
3.9 Konzeptuelles Modell Statement-Cube (Variante 1) 52
3.10 Konzeptuelles Modell Ressource-Star (Variante 2) 54
3.11 Konzeptuelles Modell Statement-Star (Variante 2) 55
3.12 Vergleich von Variante 1 und Variante 2 56

4.1 Proof-of-Concept-Umgebung 58

5.1 JSON-LD eingebettet in HTML und entsprechende RDF-Tripel 92
5.2 JSON-LD als RDF-Graph (Teil 1) 93
5.3 JSON-LD als RDF-Graph (Teil 2) 93
5.4 Visualisierung der Duplicate-Blank-Node-Identifier 94
5.5 Lösungsvorschlag für die Duplicate-Blank-Node-Identifier . . . 95
5.6 Screenshot - Persistierung der Rohdaten und der schema.org-Hierarchie im Azure-Blob-Store 104
5.7 Screenshot - Erstellung der Databricks-Umgebung 105
5.8 Screenshot - Databricks-Konfiguration 105
5.9 Screenshot - Graphische Databricks-Benutzerschnittstelle . . . 106
5.10 Screenshot - Databricks-Spark-Cluster 106
5.11 Screenshot - Databricks-Notebook 107

5.12 Screenshot - Anzahl von Einträgen in einer Rohdatendatei . . 108
5.13 Screenshot - Anzahl von Einträgen in zehn Rohdatendateien . 108
5.14 Screenshot - Anzahl von Einträgen in *cube_resources* mit einer Rohdatendatei 109
5.15 Screenshot - Anzahl von Einträgen in *cube_resources* mit zehn Rohdatendateien 109

Tabellenverzeichnis

2.1 Statistik des Common-Crawl-Korpus November 2017 [9] 22
2.2 Enthaltene strukturierte Web-Daten-Formate im Common-Crawl-Korpus November 2017 [9] 23

4.1 Rohdaten von Web-Data-Commons 64
4.2 Faktentabelle *resourcetypes* 64
4.3 Faktentabelle *statementtypes* 64
4.4 Auszug aus dem Ressource-Cube *cube_resources* 73
4.5 Auszug aus dem Statement-Cube *cube_statements* 75
4.6 Auszug aus dem Ressource-Star *star_resourcetypes* (Teil 1) . . 78
4.7 Auszug aus dem Ressource-Star *star_resourcetypes* (Teil 2) . . 78
4.8 Auszug aus dem Statement-Star *star_statementtypes* (Teil 1) . 80
4.9 Auszug aus dem Statement-Star *star_statementtypes* (Teil 2) . 80
4.10 Auszug aus dem Statement-Star *star_statementtypes* (Teil 3) . 80

5.1 Vergleichsergebnis zwischen Ressource-Cube und Ressource-Star 83
5.2 Vergleichsergebnis zwischen Statement-Cube und Statement-Star 85
5.3 Ergebnis der Abfrage der 20 häufigsten Klassen in einer Microdata-Datei 87
5.4 Ergebnis der Abfrage der 20 häufigsten Klassen in einer JSON-LD-Datei 88
5.5 Ergebnis der Abfrage der 20 häufigsten Klassen in einer RDFa-Datei 89
5.6 Vergleich der Verwendung von schema.org-Klassen in den Formaten Microdata, JSON-LD und RDFa 90
5.7 Problem durch Duplicate-Blank-Node-Identifier in JSON-LD-Rohdaten 91
5.8 Stichprobe der Blank-Node-Identifier in einer Microdata-Datei 96
5.9 Kontrolle der Blank-Node-Identifier des JSON-LD-Korpus 2017-12 98

5.10 Ergebnis Abfrage 1 des Statement-Cube 99
5.11 Ergebnis Abfrage 2 des Statement-Cube 100
5.12 Ergebnis Abfrage 3 des Statement-Cube 101
5.13 Ergebnis einer Abfrage des Ressource-Star 101
5.14 Ergebnis einer Abfrage des Ressource-Star mit Datenherkunft 102
5.15 Ergebnis einer Abfrage des Statement-Star 103

Listings

3.1 Beispiel von in HTML eingebetteten JSON-LD-Daten 43
4.1 Web-Data-Commons-Daten in das HDFS kopieren 59
4.2 Verwendete Python-Libraries und Spark-Umgebung 59
4.3 Schema des Data-Frame für die Rohdaten 60
4.4 Data-Frame der RDF-Klassifizierung 61
4.5 Data-Frame *df_resources* 61
4.6 Data-Frame *df_resourcetypes* 62
4.7 Data-Frame *df_statementtypes* 62
4.8 Persistierung der Faktentabellen in Parquet-Dateien 63
4.9 Data-Frame *df_schemaorg* der schema.org-Hierarchie 65
4.10 Data-Frame *df_hierarchy* der erweiterten schema.org-Hierarchie 65
4.11 Data-Frame *df_hierarchy_tr* der transitiv-reflexiven Hülle . . 66
4.12 Data-Frame *df_hierarchy_leveled* der in Ebenen unterteilten Hierarchie 68
4.13 Persistierung der Hierarchien in Parquet-Dateien 70
4.14 Einlesen der Parquet-Dateien für die Cube-Generierung 70
4.15 Bereinigung nicht zugeordneter Ressourcen-Klassen für den Ressource-Cube 71
4.16 Data-Frame des Ressource-Cube *df_cube_resources* 72
4.17 Bereinigung nicht zugeordneter Ressourcen-Klassen für den Statement-Cube 73
4.18 Data-Frame des Statement-Cube *df_cube_statements* 74
4.19 Einlesen der Parquet-Dateien für die Star-Generierung 75
4.20 Data-Frame *df_star_resourcetypes* 76
4.21 Data-Frame *df_star_statementtypes* 79
5.1 Abfrage 1 zum Ergebnisvergleich 83
5.2 Abfrage 2 zum Ergebnisvergleich 84
5.3 Abfrage des Ressource-Cube zur Ermittlung der 20 häufigsten Klassen 86
5.4 Abfrage 1 des Statement-Cube 99
5.5 Abfrage 2 des Statement-Cube 100

5.6 Abfrage 3 des Statement-Cube 100
5.7 Abfrage des Ressource-Star 101
5.8 Abfrage des Ressource-Star mit Datenherkunft 102
5.9 Abfrage des Statement-Star 102
5.10 Konfiguration und Verwendung des Azure-Blob-Stores 107

Kapitel 1

Einleitung

In diesem Kapitel wird auf die Motivation dieser Arbeit eingegangen und an das Thema Data-Profiling im Kontext von strukturierten Web-Daten herangeführt. Nach der Aufgabenstellung und Abgrenzung der Arbeit wird ein Überblick über die Struktur und Inhalte der folgenden Kapitel gegeben.

1.1 Motivation

Die Anwendungsfälle von Data-Profiling sind vielfältig und reichen von Daten-Exploration über Daten-Integration bis hin zur Überprüfung der Datenqualität neu erschlossener Datenquellen [1]. In allen Anwendungsfällen ist die Kenntnis über Metadaten der zu verarbeitenden Daten erforderlich. Data-Profiling ermöglicht die Eruierung dieser Metadaten und wird von Johnson wie folgt definiert:

"Data-profiling refers to the activity of creating small but informative summaries of a database." [2]

Eine derartige Untersuchung von Daten gibt Auskunft über deren Beschaffenheit in Form von Statistiken über die Anzahl von "NULL" sowie eindeutigen Werten in Spalten einer Tabelle [1]. Der verwendete Datentyp und die am häufigsten vorkommenden Muster von Werten in einer Spalte zählen genauso zum Ergebnis von Data-Profiling wie Erkenntnisse über funktionale Abhängigkeiten zwischen Spalten [1], und dienen als Grundlage für die Umsetzung der verschiedensten Anwendungsfälle. Mithilfe von Data-Profiling sollen ein Verständnis über die zu verarbeitenden Daten geschaffen und Unstimmigkeiten in den Datensätzen erkannt werden können [3]. Die Erhebung der angeführten Statistiken wird sehr häufig in relationalen Systemen durchgeführt,

in denen die Semantik der Daten klar ist und die Daten in strukturierter Form vorliegen [3].

Neue Herausforderungen, aber auch weitere Möglichkeiten ergeben sich im Umfeld von Data-Profiling durch Anwendungsfälle, die neue Trends im Datenmanagement einbeziehen [1]. Darunter fallen vor allem Anwendungsfälle im Bereich Big-Data-Analytics, in denen das hohe Datenvolumen, die hohe Geschwindigkeit, in der neue Daten generiert werden, und die große Vielfalt an Daten zu bewältigen sind [4][5].

Daten mit den beschriebenen Eigenschaften sind auch in der Domäne des Semantic-Web zu finden. Öffentlich zugängliche und verlinkte Daten (Linked-Open-Data (LOD)) sind oft unterschiedlich strukturiert und unterscheiden sich auch in ihrer Semantik [3]. Der Begriff *Linked-Data* fasst die Methoden zusammen, um strukturierte Web-Daten in maschinenlesbarer Form im Internet zu verlinken [6]. Google-Search beschreibt strukturierte Web-Daten als ein standardisiertes Format, um Informationen über eine Webseite zur Verfügung zu stellen und den Inhalt der Seite zu klassifizieren [7]. Als Formate haben sich Microdata, RDFa, Microformats [8] und JSON-LD [9] entwickelt.

Die Differenzen der Struktur und Bedeutung sind auf die divergente Herkunft der Daten zurückzuführen [3]. Um eine einheitliche Interpretation von strukturierten Web-Daten zu ermöglichen, wurde 2011 das Projekt schema.org von den führenden Suchmaschinenbetreibern im Web ins Leben gerufen [10]. Das Projekt schema.org stellt ein einheitliches Vokabular zur Verfügung, das eine Vielzahl von im Web vorkommenden Themen und Begriffen in einer Hierarchie abbildet [10]. Das schema.org-Vokabular kann zum Beispiel unter Verwendung des Resource-Discription-Frameworks (RDF) in Webseiten eingebunden werden.

Im Sinne von Data-Profiling gilt es, vor der Durchführung von Analysen der schema.org-Hierarchie die tatsächliche Verwendung und Verbreitung des schema.org-Vokabulars in Webseiten zu erheben. Das Wissen über die Verwendung des schema.org-Vokabulars ermöglicht Analysten, aussagekräftige Abfragen durchzuführen, welche sich auf jene Bereiche der schema.org-Hierarchie konzentrieren, die auch ein entsprechendes Vorkommen auf den zu analysierenden Webseiten aufweisen. Zusätzlich können durch Data-Profiling Probleme der Datenqualität erkannt werden, was sich wiederum positiv auf die Qualität der eigentlichen Analysen auswirkt.

Die Kenntnis über die tatsächliche Verwendung des schema.org-Vokabulars bietet zudem ein Hilfsmittel für die Weiterentwicklung des Vokabulars. Die schema.org-Community ist seit dem Jahr 2011 damit beschäftigt, das bereits vorhandene Vokabular zu pflegen und weiterzuentwickeln [10]. Diese von der schema.org-Community durchgeführten Tätigkeiten können durch das auf die Verwendung des schema.org-Vokabulars gerichtete Data-Profiling

unterstützt werden.

Da die Ergebnisse des Data-Profilings hauptsächlich von Personen ausgewertet und interpretiert werden, ist die Visualisierung der erhobenen Metadaten in entsprechender Form, die für Personen anschaulich und leicht verständlich ist, auch von Bedeutung [1].

1.2 Aufgabenstellung

Ausgangspunkt dieser Arbeit sind die Daten von Web-Data-Commons [8], einem Projekt der Universität Mannheim, welches strukturierte Web-Daten aus den Korpussen von Common-Crawl [12] extrahiert. Die extrahierten Daten werden von Web-Data-Commons als RDF-Quads, getrennt nach ihrem Ursprungsformat (JSON-LD, RDFa und Microdata) der Öffentlichkeit zugänglich gemacht. In einem großen Teil dieser Daten werden die Klassen und Properties des schema.org-Vokabulars verwendet.

Die zentrale Aufgabenstellung dieser Arbeit ist die Konzeption und Umsetzung eines Ansatzes zur Analyse der Verwendung des schema.org-Vokabulars in diesen strukturierten Web-Daten. Insbesondere soll für das Data-Profiling die Anzahl der Vorkommen von Klassen und Properties aus dem schema.org-Vokabular abfragbar gemacht werden. Weiters soll abfragbar gemacht werden, wie oft Instanzen welcher Klassen mittels welcher Properties in Beziehung gesetzt werden. Diese Abfragemöglichkeiten sollen für unterschiedliche Granularitätsebenen zur Verfügung gestellt werden. Dabei ist darauf zu achten, dass Klassen im schema.org-Vokabular in einer Klassenhierarchie angeordnet sind und beim Zählen der Instanzen einer Klasse zwischen direkten und indirekten Instanzen unterschieden werden soll. Abfragen sollen auch auf die Datenherkunft eingeschränkt werden können, um beispielsweise die Anzahl der Vorkommen von Klassen und Properties für bestimmte Webseiten abfragen zu können.

Bei Konzeption und Umsetzung sind folgende weitere Vorgaben zu berücksichtigen:

- Um flexible Abfragen auf die aggregierten Daten zu ermöglichen, sollen bekannte Konzepte aus dem Bereich Data-Warehousing und OLAP verwendet und auf die Aufgabenstellung angepasst werden. Im besonderen soll die schema.org-Hierarchie als semantische Dimension [13] eingebunden werden.
- Um mit den großen im Web verfügbaren Datenmengen umgehen zu können, sollen für die Umsetzung Big-Data-Technologien verwendet

werden. Insbesondere soll der Prototyp mit Apache Spark [14] und Spark-SQL [15], einem skalierbaren Framework für die Verarbeitung von Big Data, umgesetzt werden.

- Für die Entwicklung und Ausführung des Prototyps soll im Rahmen dieser Arbeit ein lokaler Hadoop-Cluster aufgebaut werden.

Nicht Teil der Aufgabenstellung sind die Implementierung von Import- und Exportschnittstellen, die Visualisierung von aggregierten Daten und Entity-Resolution (siehe dazu Kapitel 3).

Eine erste vorläufige Evaluierung von Funktionalität und Skalierbarkeit des Ansatzes im Rahmen dieser Arbeit umfasst:

- Die Ausführung des Prototyps auf jeweils einen kleinen Ausschnitt aus Web-Data-Commons für drei verschiedene Web-Datenformate (JSON-LD, RDFa, Microdata). Mittels Abfragen auf die aggregierten Daten und einem Vergleich zwischen den Web-Datenformaten wird die Plausibilität der Ergebnisse überprüft.

- Das Deployment des Prototyps in der Cloud (zusätzlich zum lokalen Hadoop-Cluster) und Ausführung auf einen größeren Ausschnitt aus Web-Data-Commons.

1.3 Aufbau der Arbeit

Als Einstieg werden in Kapitel 2 die Grundlagen für die in dieser Arbeit verwendeten Konzepte und Technologien angeführt. Dabei wird ausgehend von Data-Profiling und dessen Anforderungen auf die verwendeten Rohdaten und deren Vorkommen eingegangen. Außerdem werden Konzepte, die für die Aufbereitung der Daten eingesetzt werden, um Analysen zu ermöglichen, erläutert und aktuell gängige Technologien im Bereich Big Data beschrieben.

Der Entwurf des Lösungsansatzes wird in Kapitel 3 gezeigt. Nach einer Übersicht des Lösungsentwurfes und der Abgrenzung der Arbeit werden die Verarbeitungsschritte der Rohdaten über die Aufbereitung der Rohdaten mit der schema.org-Hierarchie erklärt.

Die Umsetzung des Entwurfs wird in Kapitel 4 in Form der implementierten Skripten und den daraus resultierenden aufbereiteten Daten gezeigt.

In Kapitel 5 werden unter Verwendung der in Kapitel 4 gezeigten Skripten erste Auswertungen der Daten durchgeführt, um einen Einblick in die Verwendung der schema.org-Hierarchie zu bekommen und um Beispielabfragen, mit denen auch die erstellten Skripten evaluiert werden, auf die dafür

am besten geeigneten Daten auszuführen, die auch in Kapitel 5 zu finden sind. Die Skalierbarkeit der erstellten Skripten wird mithilfe der in Microsoft Azure verfügbaren Plattform Databricks überprüft.

Kapitel 6 fasst abschließend die Arbeit zusammen und gibt einen Ausblick über mögliche Weiterentwicklungen der in dieser Arbeit erstellten Konzepte.

Kapitel 2

Grundlagen

Kapitel 2 beschäftigt sich mit den in der Arbeit eingesetzten Konzepten und Technologien.

2.1 Data-Profiling

Alle Aktivitäten und Prozesse, die verwendet werden, um Metadaten einer Datenmenge zu erheben, können unter dem Begriff Data-Profiling zusammengefasst werden [1]. Die Anwendungsfälle im Bereich Data-Profiling sind sehr umfangreich und haben meist das Ziel, Informationen über Daten zu liefern, um weitere Verarbeitungen und Analysen effizient und effektiv durchführen zu können [16].

Die Ergebnisse von Data-Profiling reichen von einfachen Aussagen über verwendete Datentypen, die Anzahl von *Null*-Werten oder in den Daten häufig auftretende Muster bis hin zu Auskünften über funktionale Abhängigkeiten zwischen unterschiedlichen Spalten einer relationalen Tabelle [1].

Auch wenn die Erkennung von Mustern in Daten einen Teil von Data-Profiling ausmacht, ist dennoch eine Unterscheidung zu Data-Mining, einem weiteren großen Bereich der Datenanalyse, zu treffen [16]. Im Vergleich zu Data-Mining verfolgt Data-Profiling ein anderes Ziel [16]. Mit der Erhebung von Metadaten, wie zum Beispiel Informationen über Spalten und Tabellen in einer relationalen Datenbank, findet Data-Profiling seine Anwendung in der Unterstützung des Datenmanagements, wohingegen die Ergebnisse der Analysen des Inhaltes dieser Tabellen, die im Data-Mining untersucht werden, die Geschäftsführung eines Unternehmens als Zielgruppe hat [16]. Sehr oft ist Data-Profiling auch als eine vorgelagerte Tätigkeit eines Data-Mining-Prozesses zu finden [16].

Neben der Unterstützung von Data-Mining durch zum Beispiel der Da-

tenbereinigung, um Daten aus verschiedenen Quellen zusammenzuführen, finden sich noch viele andere Anwendungsfälle für Data-Profiling. Die Einsatzgebiete von Data-Profiling reichen von Abfrageoptimierung in Datenbank-Management-Systemen über Datenintegrationsszenarien, bei denen Daten mit weitgehend unbekannten Metadaten in bestehende Datenstrukturen integriert werden sollen, bis hin zu Anwendungen in der Wissenschaft, um von Rohdaten oder oftmals auch von extrahierten Daten aus dem Internet ein für die geplanten wissenschaftlichen Experimente entsprechendes Schema abzuleiten [16].

Gerade die Extraktion von Daten aus dem Internet, beispielsweise LOD, bringt neue Möglichkeiten der Datenanalyse, aber auch neue Herausforderungen mit sich und eröffnet Anwendungsfälle in der Analyse von Big Data [16]. Die Einbeziehung von Daten in Analysen, die nicht in eigenem Besitz sind oder die bisher nicht für Untersuchungen in Betracht gezogen wurden, können den Umfang von aussagekräftigen Resultaten erweitern [16]. Allerdings treten bei der Arbeit mit Big Data weitere Herausforderungen neben jenen, welche auch beim traditionellen Data-Profiling zu bewältigen sind, auf [16]. Zu Beginn eines Data-Profiling-Vorhabens müssen die Input-Daten definiert werden. Um die Input-Daten entsprechend wählen zu können, muss bereits zu Beginn der erwartete Output feststehen, damit die richtigen Data-Profiling-Methoden auf die richtigen Daten angewendet werden können [1]. Eine wesentliche Herausforderung besteht auch in der Komplexität der Data-Profiling-Algorithmen [1]. Anders als beim traditionellen Data-Profiling, wo die zu untersuchenden Daten in strukturierter Form und semantisch klar definiert erwartet werden, stellt sich die Situation bei Big Data und bei der Analyse von LOD dar [3]. Die Daten kommen von unterschiedlichsten Quellen und weisen daher unterschiedliche Strukturen und Semantiken auf [3], was die Auswertung der Daten erheblich erschwert. Zudem bereiten die auszuwertenden Datenmengen Probleme [1]. Daher sind die Skalierbarkeit von Data-Profiling-Methoden und die verteilte Datenverarbeitung wichtige Eigenschaften, um das Problem des Datenvolumens zu bewältigen [1].

Zudem sind immer öfter nicht relationale Daten, unstrukturierte Daten, wie Texte, und heterogene Daten im Fokus von Untersuchungen [1]. Heterogenität unterscheidet zwischen syntaktischer Heterogenität, struktureller Heterogenität und semantischer Heterogenität [16]. Im traditionellen Data-Profiling werden Daten auf syntaktische Heterogenität untersucht, indem zum Beispiel Inkonsistenzen in deren Formatierung gesucht werden [16]. Die strukturelle Heterogenität bezieht sich auf ungleiche Schemata oder unterschiedliche Strukturen der Daten, was nur teilweise von traditionellem Data-Profiling abgedeckt wird [16]. Die dritte Kategorie, semantische Heterogenität, nimmt Bezug auf die Bedeutung der untersuchten Daten [16].

Vor allem im World Wide Web finden sich heterogene oder semistrukturierte Datenformate wie XML und RDF [1]. XML ist ein Standard für den Datenaustausch im Internet, insbesondere zwischen Web-Services. Durch Data-Profiling werden Schemaelemente wie Tags und Attribute in den XML-Daten analysiert [1].

Im Bereich LOD wird sehr häufig RDF eingesetzt, um Informationen auf Webseiten maschinenlesbar zur Verfügung zu stellen [1]. Für die Erhebung von RDF-Metadaten finden sich bereits erste Tools wie LODStats [17] und ProLOD++ [18], die Auskunft über vorkommende RDF-Muster, synonym verwendete Properties und deren Beziehungen sowie die konsistente Verwendung von Ontologien geben [1]. Eine weitere wichtige Analyse von RDF-Daten bezieht sich auf die Datenherkunft (engl. provenance) [1]. Wegen der Heterogenität der miteinander verflochtenen und verlinkten Quellen ist die Analyse der Datenherkunft sehr wichtig [1].

Die am schwierigsten zu bewältigende Herausforderung im traditionellen sowie im nicht traditionellen Data-Profiling ist die Interpretation der Ergebnisse [1]. Die Ergebnisse des Data-Profilings werden in den meisten Fällen von Personen interpretiert und erfordern daher eine entsprechende Visualisierung, um dem Interpreten die richtigen Schlussfolgerungen zu erleichtern oder erst zu ermöglichen [1].

Verfügbare Data-Profiling Methoden können oft mit heterogenen Daten, großem Datenvolumen und dem damit einhergehenden Anspruch auf Skalierbarkeit nicht umgehen [16][1]. Um die angeführten Herausforderungen bewältigen zu können, haben sich neue Datamanagement-Architekturen und Frameworks wie beispielsweise verteilte Systeme, Key-Value Stores, spaltenorientierte Layouts und Stream-Verarbeitung entwickelt [1].

2.2 Strukturierte Daten im Web und schema.org

Der Inhalt und die Bedeutung einer Webseite ist für Maschinen nicht ohne aufwendige, selbst zu implementierende Applikationen, welche aus HTML strukturierte Web-Daten extrahieren, zu verstehen [10]. Da vor allem die Semantik von Webseiten bei der Verbesserung von Applikationen und Umsetzung von persönlichen Assistenten wie Google-Search, Google-App oder Microsoft-Cortana eine sehr wichtige Rolle spielt [10], können Herausgeber von Webseiten die Bedeutung ihrer Webseite mithilfe von im HTML-Code eingebetteten strukturierten Daten (im weiteren als strukturierte Web-Daten bezeichnet) explizit anführen [7]. Das schema.org-Vokabular wird hauptsächlich für strukturierte Web-Daten in den Formaten Microdata, RDFa und JSON-LD eingesetzt [19].

Neben den aktuell gängigen Formaten finden sich aber auch noch andere Ansätze. Als eines der ersten Formate wurde XML eingesetzt [10]. Nach der ursprünglichen Idee, XML für die Standardisierung von Syntax zu verwenden, wurde auch der Nutzen für strukturierte Web-Daten entdeckt [10].

Mit einem gerichteten Graphen als Datenmodell wurde mit dem Meta-Content-Framework versucht, eine Wissensrepräsentation im World Wide Web umzusetzen und damit Applikationen die Verarbeitung von vielen verschiedenen Webseiten zu ermöglichen [10].

Ende der Neunziger bis Anfang der Zweitausender Jahre wurden neue Standards für Syntax und Datenmodelle entwickelt [10]. In dieser Zeit entstand auch eine Vielzahl von unabhängigen Vokabularen für verschiedenste Anwendungsfälle wie zum Beispiel Rich-Site-Summary (RSS), mit dem Benutzer beliebige, RSS unterstützende Nachrichtenquellen auf Webseiten verwenden können [10]. Auch verschiedene Mikroformate wie vCard/hCard entstanden in dieser Zeit für den Austausch von zum Beispiel Kontaktinformationen [10]. Friend-of-a-Friend eignet sich für Social-Network-Data und wird oft von der Linked-Data-Community verwendet [10]. JSON-LD verwendet das bekannte und weitverbreitete JSON und wird als leichtgewichtiges Linked-Data-Format, welches besonders für auf REST basierende Web-Services und semistrukturierte Datenbanken wie MongoDB geeignet ist, eingesetzt [20].

Die Unabhängigkeit der einzelnen Vokabulare führte allerdings zu einer starken Duplizierung von Begriffen und verwendeten Klassen, was die Anwendung von strukturierten Web-Daten für Herausgeber von Webseiten und deren Webmaster sehr erschwert [10]. Daher waren die verwendeten Formate sehr oft falsch verwendet oder es wurde einfach auf den Einsatz von strukturierten Web-Daten verzichtet [10].

Um dieses Problem einzudämmen, entwickelten die vier Suchmaschinenanbieter Bing, Google, Yahoo! und später auch Yandex im Jahr 2011 das schema.org-Vokabular mit dem Ziel, Herausgebern von Webseiten ein einheitliches Schema und Vokabular zu bieten, mit dem weitreichend verschiedenste Themen und Inhalte wie Personen, Orte, Events, Produkte und Angebote abgedeckt werden können [10]. Unabhängig von den unterschiedlichen Suchmaschinen soll die Verwendung von strukturierten Web-Daten immer gleich ablaufen und damit ein vereinfachter und vermehrter Einsatz von strukturierten Web-Daten erzielt werden [10]. Das schema.org-Vokabular kann als eine Menge von Klassen, die hierarchisch angeordnet sind und jeweils eine Menge von Properties besitzen, betrachtet werden [19].

Das schema.org-Vokabular wurde mit 297 verschiedenen Klassen und 187 Beziehungen gestartet und wird von der schema.org-Community stetig weiterentwickelt [10]. Mitlerweile umfasst das schema.org-Vokabular 597 Klas-

sen, 875 Properties und 114 Enumerations und wird von über 10 Millionen Websites verwendet [19]. Eine weitere Erhebung bezüglich der Verwendung des schema.org-Vokabulars hat gezeigt, dass in der verwendeten Datenprobe, welche aus einer Kombination von Google-Index und Web-Data-Commons bestand, 31,3% der in der Datenprobe enthaltenen Websites das schema.org-Vokabular verwendeten [10]. Damit haben strukturierte Web-Daten die gleiche Größenordnung wie das World Wide Web selbst erreicht [10].

2.3 Web-Data-Commons und Common-Crawl

Damit Informationen über die Bedeutung von Webseiten in maschinenlesbarer Form zur Verfügung gestellt werden können, werden immer häufiger strukturierte Web-Daten-Formate eingesetzt. Metadaten, die Auskunft über die Bedeutung einer Webseite geben, werden mithilfe von Markup-Standards wie beispielsweise Microformats, RDFa, Microdata [8] oder JSON-LD [20] in HTML eingebunden. Diese Metadaten beschreiben unter anderem Produkte, Personen oder Orte, die auf einer Webseite gezeigt werden, in einem für Maschinen lesbaren Format. Spezielle Parser wie zum Beispiel Any23 [21] sind in der Lage, diese in HTML eingebetteten Informationen zu extrahieren und in andere Formate wie RDF zu transformieren [22]. Die Ergebnisse von Suchmaschinen wie Google, Yahoo!, Yandex und Bing werden mit den Metadaten angereichert, um entsprechende Beschreibungen der gesuchten Entitäten anzuzeigen [8]. Die genannten Suchmaschinenanbieter extrahieren die auf Webseiten enthaltenen Metadaten, stellen die Ergebnisse aber nicht öffentlich zur Verfügung [8].

Aus diesem Grund wurde das Projekt Web-Data-Commons gestartet, um den öffentlichen Zugang zu den in HTML eingesetzten strukturierten Web-Daten zu schaffen und weitere Forschung und Analysen in diesem Umfeld zu ermöglichen [8]. Die Grundlage für die Extraktion von strukturierten Web-Daten stellt die Common-Crawl-Foundation dar [8]. Die Common-Crawl-Foundation ist eine Non-Profit-Organisation, die zum Ziel hat, einen Abzug des Internets der Öffentlichkeit für Forschungsprojekte und Analysen zur Verfügung zu stellen [12]. Der gesamte Korpus von Common-Crawl enthält Petabytes von Daten und stellt Webseiten, extrahierte Metadaten und Text-Daten zur Verfügung [12]. Gespeichert werden die Daten im Web-ARChive-Format (WARC-Format) und sind über Amazon-Web-Services (AWS) frei zugänglich [12]. In AWS können die Daten direkt verarbeitet oder heruntergeladen werden [12].

Die von Common-Crawl veröffentlichten Korpusse werden von Web-Data-Commons genutzt, um die in HTML eingebetteten strukturierten Web-Daten

zu extrahieren. Die extrahierten Daten werden von Web-Data-Commons als RDF-Quads veröffentlicht. Zu Beginn von Web-Data-Commons wurden Microdata, RDFa und Microformats aus den Common-Crawl-Korpussen der Jahre 2010, 2012 und 2013 extrahiert [8]. Bei der 2016 durchgeführten Extraktion von strukturierten Web-Daten durch Web-Data-Commons wurde erstmals auch JSON-LD extrahiert [9].

1,2 Milliarden der 3,2 Milliarden HTML-Seiten aus dem Common-Crawl-Korpus von November 2017 enthalten strukturierte Web-Daten [9]. Die gefundenen strukturierten Web-Daten stammen von 7,4 Millionen unterschiedlichen Pay-level-domains und resultieren in 38,7 Milliarden Ergebnis-RDF-Quads [9].

Tabelle 2.1 [9] zeigt eine Statistik zum im November 2017 durchgeführten Crawl. Die rund 3,2 Milliarden geparsten HTML-Seiten, welche ca. 26 Millionen Pay-level-domains zuzuordnen sind, ergeben ein komprimiertes Datenvolumen von 66 Terabyte [9]. Die daraus von Web-Data-Commons extrahierten 38,7 Milliarden RDF-Quads resultieren in einem komprimierten Datenvolumen von 855 Gigabyte [9].

Tabelle 2.1: Statistik des Common-Crawl-Korpus November 2017 [9]

Crawl Date	November 2017
Total Data	66 Terabyte (compressed)
Parsed HTML URLs	3,155,601,774
URLs with Triples	1,228,129,002
Domains in Crawl	26,271,491
Domains with Triples	7,422,886
Typed Entities	9,430,164,323
Triples	38,721,044,133
Size of Extracted Data	855 GB (compressed)

In Tabelle 2.2 mit Daten von [9] ist das Vorkommen unterschiedlicher Formate in den extrahierten Daten ersichtlich. Bezogen auf die Anzahl der extrahierten Tripel, stellen Microdata, JSON-LD, das Microformat hcard und RDFa die am häufigsten verwendeten Formate für strukturierte Web-Daten dar. Microdata weist mit einer Anzahl von rund 24,3 Milliarden extrahierten Tripel mit Abstand die meisten Vorkommen auf.

Tabelle 2.2: Enthaltene strukturierte Web-Daten-Formate im Common-Crawl-Korpus November 2017 [9]

Format	**Domains**	**URLs**	**Typed Entities**	**Triples**
html-microdata	3,743,822	646,409,625	4,837,635,224	24,359,443,316
html-embedded-jsonld	2,685,738	190,890,906	818,557,558	3,623,025,088
html-mf-hcard	2,758,884	418,095,860	3,186,672,022	8,371,745,745
html-rdfa	1,209,430	220,889,867	430,349,620	1,629,581,643
html-mf-xfn	392,035	27,320,114	69,259,620	401,275,671
html-mf-adr	192,390	17,895,411	41,729,827	143,728,079
html-mf-geo	37,807	4,646,171	7,965,632	21,674,355
html-mf-hcalendar	40,257	2,343,185	14,251,603	62,666,600
html-mf-hreview	27,181	3,702,554	10,153,540	59,026,065
html-mf-hlisting	7,162	314,164	9,148,197	31,778,446
html-mf-hrecipe	5,179	543,865	3,681,013	15,228,823
html-mf2-h-adr	1,880	161,842	415,221	1,061,536
html-mf-hresume	223	16,902	37,970	53,951
html-mf-species	224	99,301	307,276	754,815
OVERALL	7,422,886	1,228,129,002	9,430,164,323	38,721,044,133

Auch wenn durch Common-Crawl und Web-Data-Commons eine sehr umfangreiche Datenquelle für Forschung und Analysen der Öffentlichkeit bereitgestellt wird, ist zu beachten, dass nicht alle im Internet vorkommenden Webseiten in den Korpussen und daraus extrahierten RDF-Quads enthalten sind.

2.4 Data-Warehousing und semantische Dimensionen

In Unternehmen finden sich oftmals mehrere voneinander unabhängige, heterogene operative Systeme mit unterschiedlichen Datenformaten, Zugriffsmechanismen und Speichertechniken [23]. Mit operativen Systemen werden die geschäftlichen Transaktionen eines Unternehmens und deren Verwaltung unterstützt [23]. Durch die unabhängige und heterogene Struktur der Systeme werden auch Informationen, die für Analysen und die Gewinnung von neuen Erkenntnissen Voraussetzung sind, im Unternehmen verteilt [23]. Neben den im Unternehmen verteilten Informationen sind auch Erkenntnisse, die sich erst nach der Verknüpfung von verschiedenen operativen Systemen gewinnen lassen, eine wesentliche Grundlage für die Unterstützung des Unternehmensmanagements in Bezug auf Planung und Entscheidung [23].

Um das Unternehmensmanagement beim Prozess der Entscheidungsfindung entsprechend unterstützen zu können, wurden entscheidungsunterstützende Systeme, die auch als Decision Support Systems, Executive Information Systems oder Management Information Systems bezeichnet werden, entwickelt [23]. Solche Systeme werden neben den konventionellen Datenbanksystemen der operativen Systeme betrieben und entkoppeln operative und analytische Systeme, um technische und administrative Schwierigkeiten der Datenbereitstellung bewältigen zu können [23].

Die für entscheidungsunterstützende Systeme essentiellen Daten und Informationen, die in den operativen Systemen eines Unternehmens gehalten werden, übersteigen die auf einem lokalen Desktop verarbeitbare Datenmenge [23]. Zudem besteht bei lokal verarbeiteten Daten die Gefahr, wegen versäumter Aktualisierung der lokal für Analysen gespeicherten Daten eine inkonsistente Grundlage für Analysen zu verwenden und dadurch Entscheidungen basierend auf falschen Fakten zu treffen [23]. Die sich üblicherweise laufend wiederholenden Analysen und der dafür notwendige hohe programmiertechnische Aufwand, verursacht durch die vielen Daten aus den verschiedenen operativen Systemen mit oftmals unterschiedlichen Strukturen und Schemata, bilden die Motivation für die Entwicklung von eigens für diese Anforderungen konzeptionierten Systemen [23].

Bei dem dabei verwendeten Ansatz werden die Daten aus den operativen Systemen zusätzlich in einem sogenannten Data-Warehouse abgelegt [23]. Data-Warehouse-Systeme unterscheiden sich bezüglich Struktur, Performanz, Funktionsweise und Zweck von in operativen Systemen eingesetzten Datenbanksystemen [23]. Der vorrangige Zweck von Data-Warehouse-Systemen besteht in der Bereitstellung von Funktionen, um den Entscheidungsfindungsprozess von Anwendern zu unterstützen [23]. Der Mehrwert der Verwendung eines Data-Warehouse besteht in der Integration und Bereinigung der Daten aus den verschiedenen operativen Systemen, was die Verwendung von eigenen Datenmodellen, die bei für den Entscheidungsprozess relevanten Analysen benötigt werden, erlaubt [23]. Bei der Zusammenführung und Aufbereitung der von unterschiedlichen Quellen stammenden Daten im Data-Warehouse wird die Funktionalität der Quellsysteme nicht beeinträchtigt [23]. Durch die fortwährende Speicherung von Daten und der laufenden Erweiterung der Datenbestände im Data-Warehouse entsteht eine Historie der Daten [23], die Analysen über den Zeitverlauf ermöglichen. Die im Data-Warehouse abgelegten Daten müssen nicht ausschließlich aus operativen Systemen des Unternehmens stammen, sondern können auch Daten aus dem World Wide Web für Auswertungen beinhalten [23]. Die Bündelung von Daten aus verschiedenen internen und externen Quellen führt zu einem großen zu bewältigenden Datenvolumen, was die Verdichtung der auszuwertenden Daten

auf geeignete Granularitätsstufen erfordert [23].

Der Begriff Data-Warehouse bezieht sich auf die Datenbanken, in denen die Daten aus den verschiedenen Quellen in aufbereiteter Form für Analysen gehalten werden [23]. Um Data-Warehousing, den dynamischen Prozess, der notwendig ist, um von den Datenquellen zu einem Analyseergebnis zu kommen, umzusetzen, ist eine umfassendere technische Infrastruktur erforderlich [23]. Data-Warehousing umfasst neben der Planung, dem Aufbau und dem Betrieb der als Data-Warehouse-System bezeichneten Infrastruktur auch die folgenden, auf die Beschaffung und Auswertung von Daten bezogenen Schritte [23]:

- Extraktion der relevanten Daten
- Transformation der Daten
- Laden der transformierten Daten in das Data-Warehouse
- Persistierung der Daten im Data-Warehouse
- Bereitstellung der Daten für Auswertungen
- Analyse der bereitgestellten Daten

Die ersten drei der angeführten Schritte werden auch als Extract-Transform-Load-Prozess (ETL-Prozess) bezeichnet [23]. Nach der Extraktion der Daten aus den benötigten Datenquellen werden die Daten in der Transformation bereinigt, um die Daten aus unterschiedlichen heterogenen Quellen zusammenzuführen und für Analysen aufzubereiten [23]. Abgeschlossen wird der ETL-Prozess durch das Laden der aufbereiteten Daten in das Data Warehouse [23].

Um das Datenmodell zu definieren, in das die aufbereiteten Daten geladen werden, wird im Data-Warehouse, anders als bei konventionellen Datenbanksystemen, bei denen die Datenmodelle häufig einem Entity-Relationship-Modell zugrunde liegen, die multidimensionale Datenmodellierung angewandt [23]. Dieses Konzept hat sich für die Modellierung von auswertungsorientierten Datenanalysen im Entscheidungsfindungsprozess ausgezeichnet und ermöglicht die Beantwortung von analytischen, mehrdimensionalen Abfragen [23]. Eine Dimension kann dabei als Datenstruktur verstanden werden, die für eine bestimmte Datenanalyse notwendige Aspekte abbildet [23]. Zu den für eine Analyse erforderlichen Aspekten zählen beispielsweise Ort, Zeit, Produkt und Kunde [23]. Hinsichtlich der Granularität der einzelnen Dimensionen werden diese in hierarchisch aufgebauten und klassifizierten Dimensionselementen angeordnet [23]. Die Dimension Zeit kann in die Granularitätsstufen

Tag, Woche, Monat und Jahr gegliedert werden [23]. Die Klassifizierungshierarchie kann als Baum betrachtet werden, in dem die Blätter von den Dimensionselementen gebildet werden [23]. Durch die hierarchische Anordnung der Dimensionen können die Daten entlang der Dimensionen aggregiert werden [23]. Die multidimensionalen Operatoren Drill-down und Rollup ermöglichen die Navigation entlang der Dimensionen [23]. Die Anwendung von Drill-down erlaubt die Navigation in Richtung einer feingranulareren Hierarchieebene, wohingegen Rollup in die gegengesetzte Richtung navigiert und die Daten grobgranularer aggregiert [23]. Durch den Einsatz der multidimensionalen Datenmodellierung wird Analysten die Betrachtung von betriebswirtschaftlichen Kennzahlen wie Umsätze, Gewinne und Verluste aus verschiedenen Perspektiven, die als Dimensionen abgebildet werden, ermöglicht [23].

Im multidimensionalen Datenmodell stellen Dimensionen qualifizierende Elemente dar [23]. Die Kombination von unterschiedlichen Dimensionen bildet einen sogenannten Datenwürfel [23]. Die Dimensionen werden dabei durch die Kanten des Würfels dargestellt [23]. Die Schnittpunkte der Dimensionen des Datenwürfels werden durch Datenzellen, in denen Fakten und Kennzahlen, die quantifizierenden Elemente des Würfels, enthalten sind, gebildet [23]. Kennzahlen repräsentieren numerische betriebswirtschaftliche Kennzahlen wie Umsätze, Gewinne und Verluste, die entlang der Achsen aggregiert und aus verschiedenen Perspektiven betrachtet werden können [23]. Die Aggregation eines Datenwürfels und der im Datenwürfel enthaltenen Kennzahlen resultiert in einem neuen Datenwürfel [23].

Die über einen Datenwürfel zur Verfügung gestellten Daten werden durch das sogenannte On-Line-Analytical-Processing (OLAP) analysiert. OLAP erlaubt explorative und interaktive Auswertungen des Datenwürfels, welche durch die Formulierung von Abfragen durch Analysten durchgeführt werden [23]. Wird das multidimensionale Datenmodell für Abfragen in einem Relationalen-Datenbank-Management-System (RDBMS) umgesetzt, wird von relationalem OLAP (ROLAP) gesprochen [23]. Die Verwendung von relationalen Strukturen für die Abbildung der multidimensionalen Datenmodelle erlaubt die effiziente Ausführung von multidimensionalen Abfragen [23].

Für die Abbildung der multidimensionalen Datenmodelle in RDBMS werden diese Modelle denormalisiert in einem sogenannten Star-Schema gespeichert [23]. Die Dimensionen des multidimensionalen Datenmodelles werden in Dimensionstabellen sternförmig um mindestens eine Faktentabelle angeordnet [23]. Eine Dimensionstabelle entspricht dabei einer Dimension des multidimensionalen Modelles [23]. Die Fakten der gleichen Granularitätsstufe des multidimensionalen Modelles sind in der Faktentabelle enthalten [23]. Für die Verknüpfung der Dimensionstabellen und der Faktentabellen sind die Primärschlüssel der Dimensionstabellen in der Faktentabelle als Fremd-

schlüssel enthalten [23].

Neben den traditionellen betriebswirtschaftlichen Kennzahlen, die mithilfe von Datenwürfeln in einem Data-Warehouse ausgewertet werden, steigt das Interesse an der Einbeziehung von Domain-Ontologien in Analysen [13]. Domain-Ontologien beschreiben ein in einem bestimmten Anwendungsgebiet gültiges Vokabular und dessen Bedeutung [13][11]. Abbildung 2.1 zeigt eine in RDF dargestellte Domain-Ontologie von [11]. Im abgebildeten RDF-Graph werden Klassen wie *Person* oder *Animal* mit deren Subklassen sowie den Object-Properties und Data-Properties definiert [11]. Object-Properties beschreiben, in welchem Verhältnis Klassen zueinander stehen. Aus der Domain-Ontologie kann entnommen werden, dass Personen andere Personen kennen [11]. Mit den angeführten Data-Properties werden Klassen genauer beschrieben. Jedes Leben hat ein Alter, das durch das Data-Property *age* abgebildet wird und den Datentyp *Integer* aufweist [11].

Um Domain-Ontologien in Analysen von Daten eines Data-Warehouse einbeziehen zu können, werden Domain-Ontologien als sogenannte semantische Dimensionen in die Faktentabelle des Data-Warehouse eingebunden [13]. Dazu wird im ersten Schritt die für die Analyse relevante Domain-Ontologie oder Teile davon identifiziert [13]. Die ausgewählte Hierarchie wird im nächsten Schritt als initiale Roll-up-Hierarchie verwendet [13]. Abschließend wird jedem Konzept, welches dem Blattknoten der Hierarchie der Domain-Ontologie entspricht, ein weiterer Blattknoten hinzugefügt, der alle Ausprägungen des jeweiligen Konzeptes abbildet [13]. Dieser hinzugefügte Blattknoten wird für die Verwendung in einem Data-Warehouse in dessen Faktentabelle eingebunden, um die Domain-Ontologie zu referenzieren [13].

Die üblichen Dimensionen eines Data-Warehouse unterscheiden sich von Domain-Ontologie referenzierenden semantischen Dimensionen bezüglich ihres Informationsgehaltes [13]. Die Knoten einer semantischen Dimension können weitere Informationen zum in der Domain-Ontologie abgebildeten Konzept enthalten [13]. Anders als herkömmliche Dimensionen in einem Data-Warehouse weisen semantische Dimensionen zudem, abgesehen von einem Top-Level und Bottom-Level der Hierarchie, keine definierten Ebenen auf [13]. Damit Daten entlang einer Hierarchie aggregiert werden können, sind definierte Ebenen Voraussetzung. Um Aggregate in semantischen Dimensionen zu bilden, können Aggregations-Ontologien definiert werden, die auf der jeweiligen Domain-Ontologie basieren [11]. Eine Aggregations-Ontologie definiert abstrakte Ebenen, welche aggregierte Sichten auf die Domain-Ontologie darstellen [11]. In Abbildung 2.2 [11] ist die Aggregations-Ontologie zur in Abbildung 2.1 [11] gezeigten Domain-Ontologie zu sehen, die wiederum in RDF abgebildet ist.

Die in den Blattknoten der Hierarchie der Domain-Ontologie vorkommen-

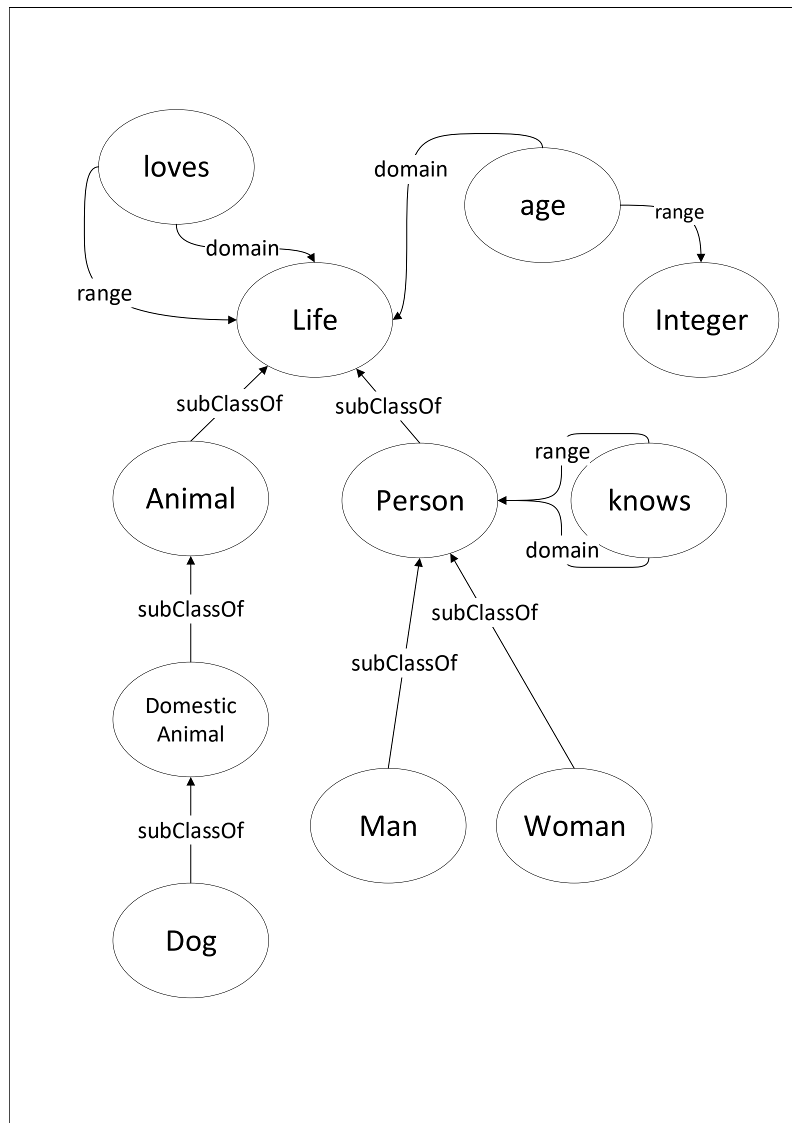


Abbildung 2.1: Eine Domain-Ontologie in RDF

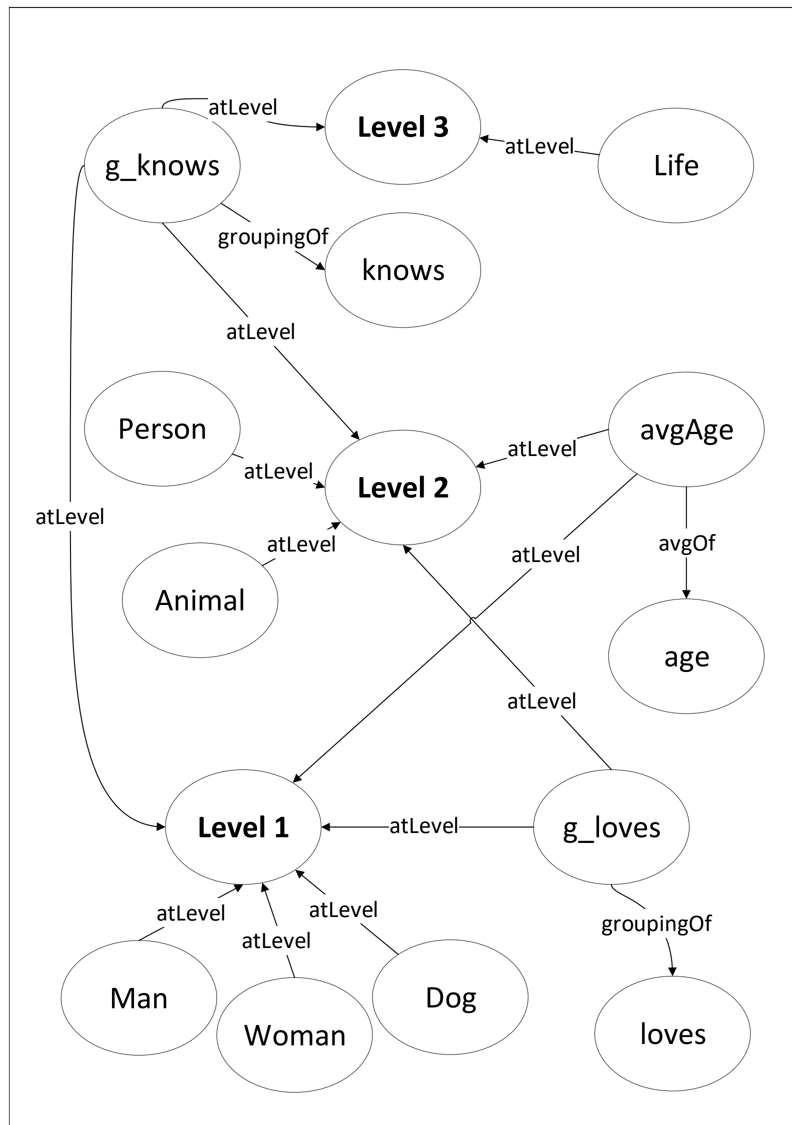


Abbildung 2.2: Eine Aggregations-Ontologie in RDF

den Klassen werden *Level1*, der feingranularsten Ebene in der Aggregations-Ontologie, zugeordnet. In der darüberliegenden Ebene *Level2* werden die Klassen aus *Level1* in deren Superklassen aggregiert. Das Ergebnis der Aggregation von *Level1* ist die Anzahl der vorkommenden *Persons* und *Animals* in den auf der Domain-Ontologie basierenden Daten. In *Level3* sind alle Klassen aggregiert, die von der Klasse *Live* abgeleitet werden können. Für die Aggregation von Object- und Data-Properties wird für jedes Property eine abstrakte Ausprägung in der Aggregations-Ontologie eingeführt und einem oder mehreren Ebenen zugeordnet. Bei der Aggregation von Object-Properties wird wie bei der Aggregation von Klassen das Vorkommen gezählt. Die Aggregation des Data-Property *age* erfolgt durch die Bildung des Durchschnittes der auf der jeweiligen Ebene vorkommenden Alter.

Durch die Verwendung von semantischen Dimensionen und der Definition einer Aggregations-Ontologie, die auf der durch semantischen Dimensionen in eine Faktentabelle eines Data-Warehouses eingebundenen Domain-Ontologie basiert, können Domain-Ontologien bei Analysen eines Data-Warehouse einbezogen werden.

2.5 Big-Data-Technologien

Bedingt durch die rasche Weiterentwicklung im Bereich der Informations- und Kommunikationstechnologien, der damit verbundenen Mobilität von Geräten und deren starke Vernetzung durch das Internet, werden immer mehr Daten gesammelt [24][4][25]. Die stetig schneller werdende Generierung von Daten und das damit verbundene Datenvolumen führt zu neuen Herausforderungen in der Datenverarbeitung und Datenanalyse [4][25]. Aus diesen Entwicklungen heraus entstand im Fachbereich der Datenverarbeitung der Begriff Big Data, der sowohl im akademischen Bereich als auch für Unternehmen an Relevanz gewinnt [24] und im wesentlichen die Technologien und Ansätze, mit denen die Herausforderung der großen Datenvolumina bewältigt werden sollen, zusammenfasst [4][24].

Historisch betrachtet ist nicht geklärt, wo der Begriff Big Data seinen Ursprung nimmt [4]. Im Jahr 1997 wurde der Begriff erstmals in einem öffentlichen Artikel [26] verwendet, um das Problem von nicht ausreichendem Arbeitsspeicher sowie Festplattenspeicher bei der Verarbeitung von großen Datensätzen zu bezeichnen [27]. Die frühere Verwendung des Begriffes wird nicht berücksichtigt, da die Bedeutung von der heutigen abweicht [4]. Bereits 2006 wurde von Doug Cutting und Yahoo! an der heute im Bereich von Big Data weit verbreiteten, verteilt arbeitenden und gut skalierbaren Technologie Hadoop gearbeitet [28]. Das im Bereich der IT tätige Marktfor-

schungsunternehmen Gartner nahm den Begriff 2011 in seinen *Hype-Cycle* auf, der einen Ausblick auf bevorstehende Technologietrends gibt [29][4]. Die Entwicklung weg von einem Trend hin zu einem etablierten Begriff zeigt eine Studie von Visible Technologies, aus der hervorgeht, dass sich in den Jahren 2009 bis 2012 die Anzahl der Begriffsnennungen in Social Media Channels um 1211% gesteigert hat [30][4]. Mittlerweile wird der Begriff Big Data häufig im Zusammenhang mit Business-Intelligence, Data-Warehouse und Data-Analytics auch von namhaften IT-Unternehmen und kommerziellen Datenbank-Herstellern wie SAP, IBM oder Oracle verwendet [4].

Auch wenn der Begriff Big Data bezüglich seiner Definition auf große Datenmengen schließen lässt, steckt mehr hinter der Bedeutung des Terminus Big Data. Vor allem die Definition einer großen Datenmenge ist schwierig und variiert zwischen Unternehmen und Anwendungsfällen. Zudem können auch traditionelle relationale Systeme große Datenmengen verarbeiten und lassen schnell an der Notwendigkeit von Big-Data-Technologien zweifeln. Relationale Systeme stoßen aber bei der Analyse von durch Benutzer generierten Inhalten (engl. User Generated Content) an ihre Grenzen [4]. Die unter anderem in Foren, Blogeinträgen oder Posts in Sozialen Netzwerken zu findenden, von Benutzern generierten Inhalte weisen häufig subjektive Meinungen der Benutzer auf, die wertvolle Informationen darstellen und auch Aufschluss über gesellschaftliche Stimmungen geben können [24][4]. Die von Benutzern generierten Inhalte treten in einem Umfang auf, der in Kombination mit der undefinierten und daher für relationale Systeme nicht geeigneten Datenstruktur zu Problemen der Datenverarbeitung in traditionellen relationalen Systemen führt [4]. Der Umfang von benutzergenerierten Inhalten, der täglich von bekannten Unternehmen gesammelt wird, bewegt sich zwischen 8 Terabyte bei Twitter, 500 Terabyte bei Facebook und 20 Petabyte bei Google [4].

Neben der Datenstruktur und der Datenmenge wird für die Definition von Big Data auch der Verarbeitungsaufwand einbezogen [31]. Bei dieser Betrachtung des Begriffes Big Data rückt die Komplexität der Datenverarbeitung in den Vordergrund [31]. Als Beispiel führt Pavlo Baron einen Use Case an, bei dem die zu verarbeitende Datenmenge auf einem USB-Stick Platz hätte, aber die Anforderung von simultanen Zugriffen auf die Daten im zweistelligen Millionenbereich je Sekunde eine komplexe Datenverarbeitung darstellt und daher als Big Data einzustufen ist [31]. Eine hohe Komplexität der Datenverarbeitung tritt auch bei Machine-Learning-Algorithmen auf, bei denen zum Beispiel das Trainieren von Modellen einen weitaus höheren Rechenaufwand erfordert, als das bei HTML-Analyse-Algorithmen, die einzelne HTML-Tags auslesen, der Fall ist [4]. Wenn die aktuell zu Verfügung stehenden Methoden zur Datenverarbeitung nicht mehr in der Lage sind, die Daten für den jewei-

ligen Anwendungsfall zu verarbeiten, kann von Big Data gesprochen werden [4].

Eine weit verbreitete und häufig verwendete Charakterisierung von Big Data stellen die 3V - Volume, Velocity und Variety - dar. Die bereits 2001 in einem technischen Bericht von Doug Leney [5] verwendeten Begriffe Volume, Velocity und Variety ließen damals schon auf die Herausforderungen im Bereich Big Data schließen [4] und haben sich daher für die Beschreibung der Big-Data-Charakteristik durchgesetzt. Im Laufe der Zeit wurden noch zwei weitere Eigenschaften - Veracity und Value - für die Charakterisierung von Big Data hinzugefügt.

Volume: Das "Big" im Begriff Big Data deutet schon auf diese Eigenschaft hin und bezieht sich auf große, zu verarbeitende Datenmengen. Wie bereits erwähnt, kann aber "Big" nicht alleine auf die Datenmenge bezogen werden, sondern ist nach Roger Magoulas je nach Anwendungsfall die zu verarbeitende Datenmenge und die zur Verfügung stehenden Verarbeitungsmethoden zu betrachten [32][4]. Die ständig steigende, für Analysen verfügbare Datenmenge führt neben den weiteren Analysemöglichkeiten von beispielsweise durch Benutzer generierte Inhalte zu neuen Herausforderungen der Datenverarbeitung [4]. Zum einen müssen die entsprechenden Daten und deren Volumen auf Speichermedien für Analysen vorgehalten werden [4]. Zum anderen müssen auch ausreichend Ressourcen wie Rechenleistung (CPU) und Arbeitsspeicher (RAM) für die Datenverarbeitung vorhanden sein, die auch bei geringeren Datenmengen und komplexen Algorithmen durchaus zu einem Engpass führen können [4]. Daher ist bei der Definition von "Big" auch die komplexe Verarbeitung, die an sich schon "Big" sein kann, zu berücksichtigen [31].

Velocity: Um einen möglichst hohen Informationsgewinn aus den gesammelten Daten zu erziehlen, ist eine rasche Verarbeitung der Daten erforderlich [33]. Je kürzer der Zeitraum zwischen dem Sammeln der Daten und deren Auswertung ist, desto höher ist oft der Wert der gewonnenen Information [4][33].

Variety: Semistrukturierte Daten wie Sensordaten, Texte, Audio- und Videodaten sowie Daten aus Logdateien [33] und Daten mit unterschiedlichen Formatierungen wie JSON, XML, HTML sind neben strukturierten Daten von Big-Data-Anwendungen zu verarbeiten [4]. Inkonsistenzen der Datenstrukturen und Formate gelten als große Herausforderung [33] für Analysten [4] und Dataprofiler. Neben der Schwierigkeit, aus den unterschiedlichen Formaten und Strukturen die richtigen Informationen zu filtern, besteht das Problem der sich im Laufe der Zeit ändernden Strukturen und Formate [4]

von bereits eingebundenen Datenquellen, die negative Auswirkungen auf die Ergebnisse der angestellten Analysen haben können. Zudem sind Datenfehler, die auch in traditionellen Systemen beachtet werden müssen, im Bereich von Big Data nicht zu vernachlässigen [4]. Als traditionelle Datenfehler können

- Fehlende Werte
- Tippfehler
- Inkonsistenzen in den Daten selbst
- Falsche Beschriftung
- Werte ohne Kontext
- Auf verschiedene Quellen verteilte Datensätze

identifiziert werden, was gemeinsam mit den inkonsistenten Datenstrukturen und Formaten zu einem deutlich höheren Aufwand der Datenaufbereitung führt als bei traditionellen Systemen wie relationalen Datenbanken [4].

Veracity: Um auch die Richtigkeit und Echtheit der Daten bei der Charakterisierung von Big Data zu berücksichtigen, führte IBM Veracity als weitere Big-Data-Eigenschaft ein [34]. Als Ursache für verfälschte Daten führt Jonas Freiknecht [4] folgende Punkte an:

- Werbung und Spam, die Personen, Produkte oder Vorkommnisse einseitig betrachten
- Texte, die durch automatisierte Übersetzung grammatikalische, sprachliche und inhaltliche Fehler aufweisen
- Falsch kategorisierte Suchergebnisse oder Foreneinträge
- Gezielte Falschaussagen oder Fehlinformationen

Value: Entscheidend ist nach Bernard Marr [35], dass aus Big Data ein entsprechender wirtschaftlicher Mehrwert für ein Unternehmen gewonnen werden kann, was durch die Big-Data-Eigenschaft Value ausgedrückt wird [35]. In beinahe allen Geschäftsfeldern lassen sich Vorteile aus der Verarbeitung von Big Data ziehen [35]. Beispielsweise können:

- Kunden besser verstanden und mithilfe von Recommender-Systemen gezieltere Angebote erstellt werden.
- Geschäftsprozesse optimiert werden.
- Gesundheitssysteme durch die Vorhersage von Krankheitsausbrüchen verbessert werden.
- Sicherheitsaspekte eines Staates verbessert werden.
- Sportler bei der Verbesserung ihrer Leistungen unterstützt werden.

Bernard Marr [35] sieht die Eigenschaft Value als die wichtigste der hier angeführten Big-Data-Eigenschaften [35]. Ohne den Fokus auf den für das Unternehmen durch Big Data möglichen Mehrwert, können keine Geschäftsfälle umgesetzt werden, die dem Unternehmen zu einem entsprechenden Wettbewerbsvorteil verhelfen [35].

Die Erkenntnis, dass traditionelle relationale Systeme nicht mit der großen, unterschiedlich strukturierten und oftmals semistrukturierten Datenmenge, die auch durch benutzergenerierte Inhalte für Analysen verfügbar wird, umgehen können, erfordert neue Ansätze und Technologien. Im Zusammenhang mit Big-Data-Technologie fällt häufig der von Doug Cutting und Yahoo! geprägte Begriff Hadoop. Das durch die Apache-Software-Foundation betreute Open-Source-Projekt Hadoop bietet die Möglichkeit der verteilten Speicherung und Verarbeitung von Daten in einem skalierbaren Cluster, der große Datenmengen in geringer Zeit parallel verarbeiten kann [36][4]. Die Skalierbarkeit in einem Hadoop-Cluster wird durch einen Scale-Out-Ansatz, bei dem neue zusätzliche Hardware anstatt wie bei Scale-Up neue mit mehr Ressourcen ausgestattete Hardware für die Erweiterung herangezogen wird, umgesetzt [4]. Da das in Java implementierte Framework Hadoop nur die Big-Data-Eigenschaften Volume und Velocity von den zuvor beschrieben Eigenschaften von Big Data erfüllt, ist Hadoop nicht als umfassende Big-Data-Technologie einzustufen [4]. Allerdings schafft Hadoop die Grundlage, um auch die weiteren Big-Data-Eigenschaften Variety und Veracity bedienen zu können [4]. Darunter fallen Algorithmen, die zum Beispiel die Struktur von Daten aus verschiedenen Quellen vereinheitlichen oder Ausreißer in den Daten erkennen, welche verteilt in einem Hadoop-Cluster ausgeführt werden können [4].

Der sogenannte Hadoop-Core, der auch als Hadoop-Common bezeichnet wird, besteht aus folgenden Komponenten, die alle für Hadoop Voraussetzung sind, aber auch unabhängig voneinander betrieben werden können [36]:

- **Hadoop Distributed File System (HDFS):** Das HDFS verwendet einen Master-Worker-Ansatz, bei dem der Master, der sogenannte Name-Node, die Metadaten der im Dateisystem abgelegten Daten hält [37]. Die Implementierung des HDFS basiert auf dem von Google entwickelten Google-File-System (GFS) [38]. Das Dateisystem ist für die verteilte Speicherung der Daten über die Worker-Knoten, auch als Data-Nodes bezeichnet, gespannt [37].

- **Hadoop Map-Reduce:** Um die im HDFS verteilten Daten auch verteilt verarbeiten zu können, stand in der ersten Version von Hadoop das ebenfalls von Google stammende Map-Reduce-Framework [39] zur Verfügung. Ein Map-Reduce-Job wird auf die im Cluster verfügbaren Data-Nodes verteilt. Die einzelnen Mapper des Jobs verarbeiten nur einen Teil der Daten und erzeugen jeweils ein Teilergebnis, welches durch einen oder mehrere Reducer mit anderen Teilergebnissen zu einem Endergebnis zusammengeführt wird.

- **Yet Another Ressource Negotiator (YARN):** In der Hadoop Version 2 wurde das Framework Map-Reduce in ein neues Modell YARN integriert [40]. Ein wesentlicher Vorteil von YARN ist die Möglichkeit, nicht nur Map-Reduce als Programmierframework zu verwenden, sondern auch andere Ansätze wie Spark verteilt in einem Hadoop-Cluster laufen zu lassen [4]. Dazu werden die im Cluster verfügbaren Ressourcen - dies sind Speicher, RAM und CPU - von YARN verwaltet und je nach Anforderung an die jeweiligen im Cluster laufenden Programme zugewiesen [40][4].

Für eine effiziente Verarbeitung von großen Datenmengen in einem Hadoop-Cluster ist das Prinzip der Datalocality ein sehr wesentliches [41][4]. Dabei wird der auszuführende Code zu den im Hadoop-Cluster verteilt gespeicherten Daten gebracht, anstatt, wie bei nicht verteilten Applikationen üblich, die Daten zum Code [41][4]. Dadurch wird verhindert, dass großen Datenmengen für die Verarbeitung über das Netzwerk geschickt werden müssen [41][4]. Stattdessen wird der Code, der im Vergleich zu den zu verarbeitenden Daten ein wesentlich geringeres Datenvolumen aufweist, zu den Data-Nodes gesendet, was die Verarbeitung viel effizienter gestaltet [41][4].

Der Hadoop-Core ermöglicht die verteilte Speicherung von Daten und bietet ein Programmiermodell, das die Ausführung von verteilten Applikationen erlaubt. Allerdings ist der Hadoop-Core als umfassende Big-Data-Technologie nicht ausreichend. Um auch umfassende Analysen und Auswertungen der verarbeiteten Daten für die Gewinnung neuer Informationen und Erkenntnisse zu ermöglichen, sind weitere Komponenten, die Hadoop nutzen oder in

Hadoop integriert werden können und auch als Hadoop-Ecosystem [41] bezeichnet werden, erforderlich. Eine genaue Definition des Hadoop-Ecosystems ist aufgrund der schnellen Entwicklung neuer Komponenten und der unterschiedlichen Betrachtungen vor allem durch die verschiedenen Hadoop-Distributionen und Cloud-Anbieter, die verschiedene Hadoop-Komponenten bündeln, nicht möglich. Zum Zeitpunkt des Schreibens dieser Arbeit sind folgende Anbieter von kommerziellen Hadoop-Distributionen und Service-Anbieter in der Cloud am stärksten verbreitet:

- Cloudera
- Hortonworks
- MapR
- Amazon Elastic MapReduce
- Microsoft Azure
- Google Cloud

Neben den Cloud-Only-Services der Cloud-Betreiber Amazon, Google und Microsoft können auch die kommerziellen Hadoop-Distributionen Cloudera, Hortonworks und MapR in der Cloud betrieben werden. Da auch unterschiedliche Übersichten des Hadoop-Ecosystems der einzelnen Distributionen im Umlauf sind, werden nachfolgend die wichtigsten Komponenten angeführt, die alle als Projekt der Apache Software Foundation geführt werden:

- **sqoop:** Ermöglicht den Datenaustauch zwischen relationalen Datenbanken und Hadoop [42]. Schnittstellen stehen für HDFS, Hive und HBase zur Verfügung [25].
- **Flume:** Bietet eine weitere Schnittstelle in einen Hadoop-Cluster, die für Nicht-Relationale Daten wie Log-Daten gedacht ist [43].
- **Kafka:** Ist eine verteilte Streaming-Plattform, die ähnlich wie eine Message-Queue mit einem Publish-Subscriber-Ansatz funktioniert und Streaming-Records speichert [44].
- **Kudu:** Ermöglicht Daten wie in relationalen Datenbanken in Tabellen, aber verteilt, zu speichern [45]. Zudem kann in Kudu-Tabellen das von relationalen Systemen bekannte Konzept eines Primärschlüssels verwendet werden [45].

- **HBase:** Die No-SQL-Datenbank ist für sehr große Tabellen mit Milliarden Zeilen und Millionen Spalten gedacht und baut auf das HDFS auf [46]. HBase folgt dem Beispiel von Google's Bigtable [47]. Als verteiltes Datenbanksystem folgt HBase dem CAP-Theorem [48], nach dem es nicht möglich ist, die Eigenschaften Consistency, Availability und Partition Tolerance in einem verteilten System zur Gänze zu erfüllen.

- **Hive:** Die Data-Warehouse-Software des Hadoop-Ecosystems erlaubt den Zugriff auf verteilt gespeicherte Daten via SQL-Abfragen über command line oder JDBC. Die abgesetzten SQL-Abfragen werden nach Map-Reduce übersetzt und als Job verteilt im Cluster ausgeführt. Hive ist durch die Fehlertoleranz der Map-Reduce-Jobs besonders für umfangreiche und lang laufende Auswertungen geeignet [49].

- **Impala:** Auch über Impala sind SQL-Abfragen verteilt im Cluster ausführbar. Anders als Hive verwendet Impala eigene Hintergrundprogramme auf den HDFS-Datenknoten, um auf die verteilten Daten zuzugreifen, und vermeidet den durch die Übersetzung nach Map-Reduce entstehenden zusätzlichen Aufwand. Impala ist bei der Ausführung von adhoc-Abfragen schneller als Hive, ist aber nicht fehlertolerant und vom Hive-Metastore abhängig [50].

- **Solr:** Unter der Verwendung des Lucene Cores [51] bietet Solr [52] die Möglichkeit, Daten zu indizieren, um die indizierten Daten hochperformant durchsuchen zu können.

- **oozie:** Der Workflow-Scheduler ermöglicht die Erstellung von Workflows und deren automatische Ausführung. In einen Workflow können unterschiedliche Jobs wie Map-Reduce, Pig, Hive oder sqoop eingebunden werden [53].

- **Pig:** Um Datentransformationen durchzuführen, stellt Pig eine weitere Option dar. Mit Pig Latin kann eine einfache Abfrage-Algebra genutzt werden, um Transformationen zu beschreiben. Die Transformationen werden über Map-Reduce-Jobs verteilt im Cluster ausgeführt [54].

- **Mahout:** Das verteilt arbeitende Lineare-Algebra-Framework ermöglicht Data-Scientists die Implementierung ihrer eigenen Algorithmen. Für die Ausführung der Algorithmen im Cluster soll Spark als Back-End-System verwendet werden [55].

- **Spark:** Das Programmierframework versucht, möglichst alle zu verarbeitenden Daten und berechnete Zwischenergebnisse im Arbeitsspei-

cher zu halten [14] und ist daher bei Batch-Verarbeitungen performanter als Map-Reduce-Jobs. Mit Spark-SQL, Spark-Streaming, der Machine-Learning-Library MLib und der Graphen-Datenbank GraphX bietet Spark mehr Funktionalität als Map-Reduce. Spark kann auch unabhängig von Hadoop verwendet werden, lässt sich aber sehr gut mit einem Hadoop-Cluster kombinieren [56].

- **Zookeeper:** Damit all die verteilten Systeme und Komponenten zusammenarbeiten können, wird Zookeeper als verteiltes Koordinationsservice eingesetzt [25]. Zookeeper orchestriert die einzelnen Komponenten und erlaubt zudem, Konfigurationen zu verwalten und verteilte Synchronisation von Komponenten umzusetzen [57].

Neben den vielen Komponenten haben sich auch neue Dateiformate für eine effiziente, verteilte Verarbeitung entwickelt. Je nach Anforderung können beispielsweise einfache Textdateien, Sequenz-Dateien, serialisierte Avro-Dateien, oder eines der spaltenorientierten Formate Parquet und Optimized Row Columnar verwendet werden.

Besonders Variety als Eigenschaft erschwert die Verarbeitung und Analyse von Big Data. Bis Analyseergebnisse von Big-Data-Applikationen vorliegen, durchlaufen die Daten verschiedene Arbeitsschritte, die von Batch-Verarbeitung über SQL-Abfragen bis hin zu Machine-Learning-Algorithmen reichen [14]. Die oben beschriebenen Komponenten des Hadoop-Ecosystem sind durchaus in der Lage, diese verschiedenen Anforderungen an die Datenverarbeitung von Big-Data-Applikationen zu bewältigen. Allerdings ist beinahe für jeden Verarbeitungsschritt eine andere Komponente zuständig, die bei der Umsetzung einer Big-Data-Applikation zu kombinieren sind [14]. Die mit der steigenden Anzahl von unterschiedlichen Komponenten steigende Komplexität wirkt sich negativ auf die Effizienz einer Big-Data-Applikation aus [14]. Um der Vielzahl an Komponenten und der damit verbundenen Komplexität entgegenzuwirken, begann an der University of California, Berkeley im Jahr 2009 die Entwicklung von Spark [14]. Die in Scala entwickelte Plattform Spark sollte eine einheitliche Plattform für die verteilte Datenverarbeitung ermöglichen und beinhaltet neben Stapelverarbeitung auch Funktionalitäten für die Umsetzung von SQL, Streaming, Machine-Learning und Graph-Verarbeitung [14]. Spark orientiert sich an Map-Reduce, führt aber eine abstrakte Schicht ein, um Daten zwischen Knoten im Spark-Cluster für die Verarbeitung auszutauschen, was den größten Unterschied im Vergleich zu Map-Reduce ausmacht [14]. Der Datenaustausch wird durch sogenannte Resilient Distributed Datasets (RDDs) umgesetzt [14]. Ein RDD kann als fehlertolerantes Datenobjekt beschrieben werden, das für die parallele Ver-

arbeitung in einem Cluster verteilt wird [14]. Anwendern stehen Programmierschnittstellen (API) in Scala, Java, Python und R zur Auswahl, mit denen RDDs erzeugt und manipuliert werden können [14]. Funktionen wie *map*, *filter* und *groupBy* stellen in Spark sogenannte *Transformationen* dar, die angewendet auf bestehende RDDs neue RDDs als Ergebnis liefern [14]. Der Aufruf einer Transformation hat allerdings noch keine Ausführung zur Folge [14]. Erst wenn eine, in Spark als *Aktion* bezeichnete Funktion, wie beispielsweise ein *count*, aufgerufen wird, startet die eigentliche Ausführung der Funktionen [14]. Dieser Ansatz wird als *lazy evaluation* bezeichnet und wird für die effiziente Berechnung des Ausführungsplans verwendet, der erst erstellt wird, wenn alle Transformationen bekannt sind, die für die Berechnung der ausgeführten Aktion benötigt werden [14]. Die erforderlichen Transformationen werden als Graph abgebildet, der auch die Grundlage für die Umsetzung der Fehlertoleranz in Spark bildet [14]. Anders als bei traditionellen Recovery-Ansätzen, die auf Datenreplikation oder Checkpointing basieren, wird in Spark *lineage* für das Recovery von fehlerhaften RDDs verwendet [14]. Dabei werden die Transformations-Graphen der RDDs herangezogen, um defekte RDDs ausgehend von den Ursprungsdaten wiederherzustellen [14]. Für die Persistierung der in Spark nur temporär für die Verarbeitung gespeicherten Daten werden externe Storage-Systeme benötigt [14]. Dafür bieten sich vor allem bei der parallelen Datenverarbeitung verteilte Filesysteme wie HDFS oder auch Key-Value-basierte Systeme wie Amazons S3 [58] oder Apache Cassandra [59] an [14].

Die mit der Analyse von Big Data einhergehende Anforderung, sowohl semistrukturierte Daten als auch relationale Daten auszuwerten, führte zu dem 2014 veröffentlichten Apache-Projekt Spark-SQL [15]. Das von Spark-SQL eingeführte DataFrame API und der verwendete erweiterbare Optimizer Catalyst ermöglichen Benutzern die effiziente und flexible Verarbeitung von sowohl semistrukturierten Daten als auch relationalen Daten mit nur einer API [15]. Catalyst kann durch Benutzer um neue Datenquellen, wie zum Beispiel semistrukturierte Daten in Form von JSON und Datenspeicher wie beispielsweise HBase, erweitert werden und unterstützt dadurch eine Vielzahl von Datenquellen für unterschiedlichste Auswertungen von Daten [15]. Zudem stehen sogenannte User-Defined-Functions (UDFs) für zusätzliche Erweiterungen zur Verfügung. Erweiterungen können durch den Benutzer inline im Code der verwendeten Programmiersprachen Scala, Python oder Java definiert und ohne aufwändiges Registrieren der UDFs, auch durch externe BI-Tools über JDBC oder ODBC, verwendet werden [15]. Verglichen mit Spark ist Spark-SQL bis zu zehnmal performanter und nutzt den Arbeitsspeicher effizienter [15]. In Spark-SQL können im Programm häufiger verwendete Daten in spaltenorientierter Form im Arbeitsspeicher gehalten werden. Diese

Spalten werden im Arbeitsspeicher komprimiert, um den Speicher effizienter nutzen zu können [15]. Anders als ein RDD in Spark kann ein Data-Frame in Spark-SQL als Tabelle wie in einem relationalen Datenbanksystem betrachtet werden und erlaubt daher übliche relationale Operatoren wie Projektion, Selektion, Join und Aggregation [15]. Spark-SQL ist eine Bibliothek, die auf Spark aufbaut und, neben der Erzeugung von Data-Frames aus Tabellen externer Quellen mit entsprechender Schnittstelle, die Erzeugung von Data-Frames direkt aus bestehenden RDDs erlaubt [15]. Abbildung 2.3 von [15] zeigt den Zusammenhang von Spark und Spark-SQL mit den Zugriffsmöglichkeiten auf die Spark und Spark-SQL Programmierschnittstellen.

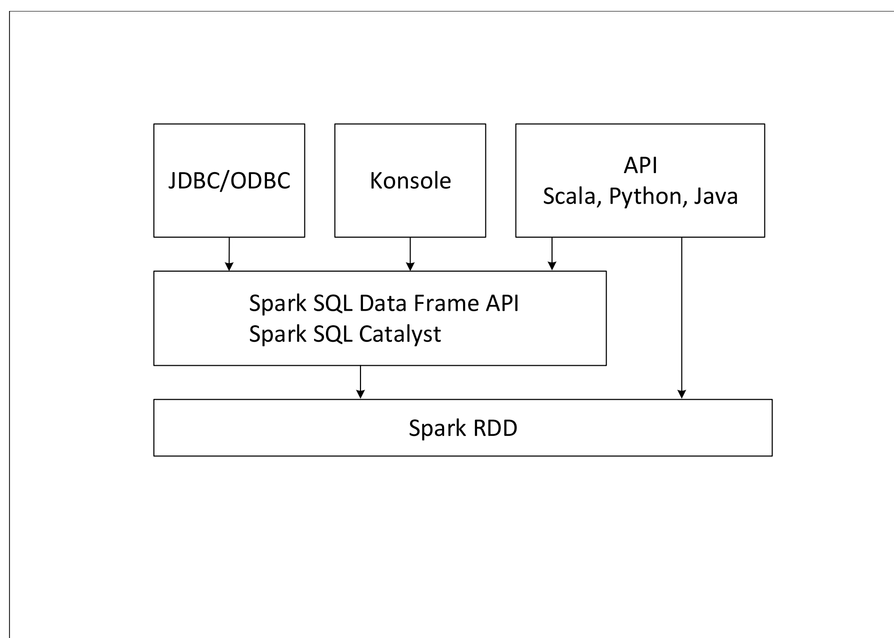


Abbildung 2.3: Zusammenhang von Spark und Spark-SQL

Um auch eine typsichere Schnittstelle verwenden zu können, wurde das Konzept des Datasets eingeführt [14]. Datasets sind in den Programmierschnittstellen für Scala und Java verfügbar, erlauben die Betrachtung von Data-Frames als Sammlung von statisch typisierten Java-Objekten und sollen zum Standard für den Datenaustausch zwischen verschiedenen Spark-Modulen werden [14].

Kapitel 3

Entwurf

Für die Lösung der Aufgabenstellung werden zwei unterschiedliche Varianten, die als RDF-Summarization-Cubes zusammengefasst werden, umgesetzt. In Variante 1 "Cube" werden voraggregierte Daten, sogenannte Sekundärfakten [60], für Abfragen zur Verfügung gestellt. Die Sekundärfakten sollen unkompliziert und rasch einen Überblick über die verwendeten schema.org-Klassen geben und sind daher einfach gehalten. Die Einfachheit der Sekundärfakten dient auch dazu, den Umfang der für Abfragen voraggregierten Daten möglichst gering zu halten, um die Aggregate in andere Systeme migrieren und für Abfragen bereitstellen zu können. Die Verwendung von Sekundärfakten hat zur Folge, dass Abfragen nur auf die aggregierten Daten möglich sind. Daher wird die Flexibilität bei Abfragen eingeschränkt. Um diese Einschränkung zu überwinden werden in Variante 2 "Star" erweiterte Primärfakten [60] erzeugt, mit denen Abfragen auf die aufbereiteten Rohdaten durchgeführt werden können.

Abbildung 3.1 zeigt eine Übersicht der verwendeten Daten und Konzepte sowie die Abgrenzung der Arbeit. Für die Extraktion von strukturierten Web-Daten aus Webseiten verwendet das in Abschnitt 2.3 gezeigte Projekt Web-Data-Commons die Archive, die von Common-Crawl, beschrieben in Abschnitt 2.3, zur Verfügung gestellt werden. Die in den Archiven von Common-Crawl enthaltenen Webseiten werden von Web-Data-Commons nach verwendeten strukturierten Web-Daten-Elementen durchsucht. Wird ein strukturiertes Web-Daten-Element im Quellcode einer Webseite gefunden, wird daraus von Web-Data-Commons ein RDF-Quad erstellt.

Des weiteren wird die Hierarchie des in Abschnitt 2.2 erklärten schema.org-Vokabulars für die Umsetzung der in Abschnitt 2.4 angeführten semantischen Dimensionen verwendet. Mit Hilfe von semantischen Dimensionen wird die schema.org-Hierarchie in die für Analysen verwendeten Primär- und Sekundärfakten eingebunden, wodurch die schema.org-Hierarchie in Abfragen ver-

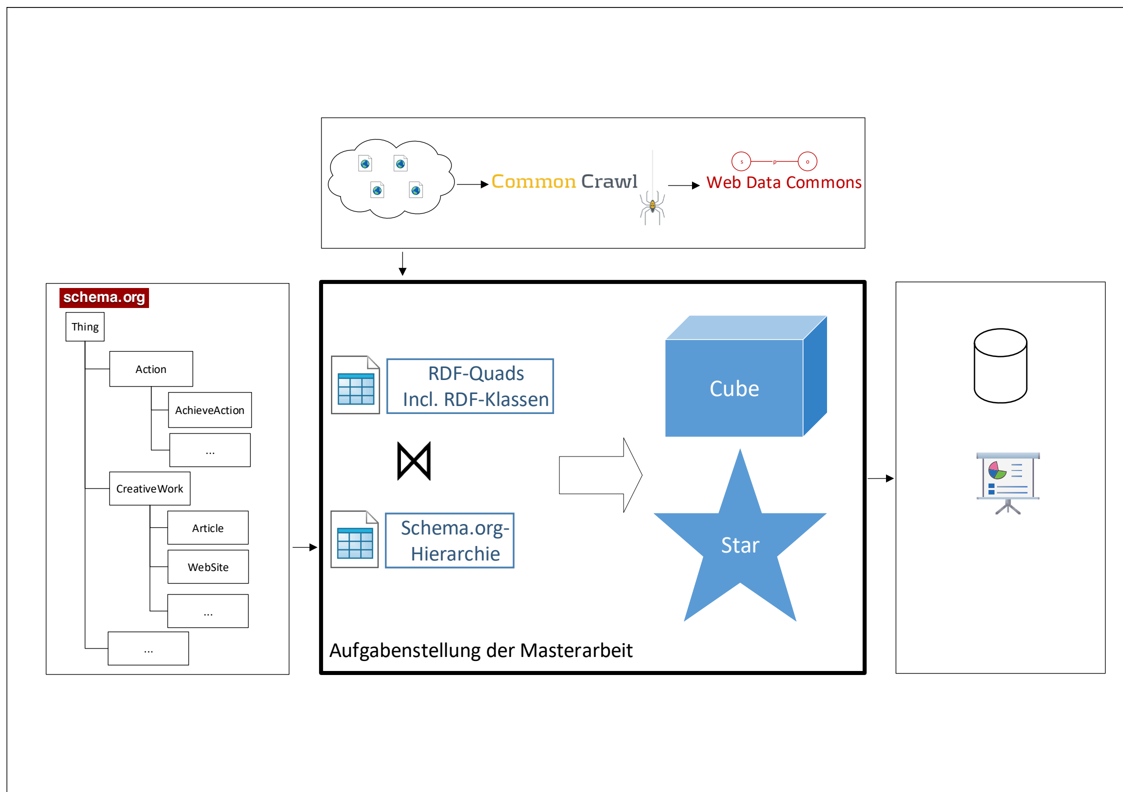


Abbildung 3.1: Übersicht der Umsetzung und Abgrenzung der Arbeit

wendet werden kann.

Der Kern der Arbeit besteht in der Aufbereitung der Daten, um Analysen zu ermöglichen und damit Erkenntnisse über die Verwendung von schema.org-Klassen zu erlangen. Für eine entsprechende Aufbereitung der Daten und die dafür notwendigen semantischen Dimensionen werden die in den Rohdaten vorkommenden RDF-Klassen mit der schema.org-Hierarchie verknüpft. Darauf aufbauend werden Variante 1 “Cube” und Variante 2 “Star” implementiert. In weiterer Folge kann das Ergebnis der Datenaufbereitung nicht nur für analytische Abfragen sondern auch für Visualisierungen von Analyseergebnissen dienen. Die Migration der aufbereiteten Daten in ein anderes System, die Visualisierung der Abfrageergebnisse, sowie die Schaffung von Schnittstellen zu anderen Systemen sind nicht Teil dieser Arbeit. Mögliche Visualisierungen der Abfrageergebnisse werden nur schemenhaft gezeigt.

3.1 Primärfakten

Die Grundlage für die von Web-Data-Commons erstellten RDF-Quads ist im Quellcode von Webseiten, wie beispielsweise ein JSON-LD-Eintrag in Abbildung 3.1 von der Webseite *https://www.allmodern.com/Dining-Tables-C366120.html?redir_sku=JCS1337*, zu finden. Jene Informationen werden extrahiert und nach RDF übersetzt.

Listing 3.1: Beispiel von in HTML eingebetteten JSON-LD-Daten

```
1 <script type="application/ld+json">{
2 "@context":"http://schema.org",
3 "@type":"WebSite"
4 ,"name":"AllModern",
5 "url":"https://www.allmodern.com"}</script>
```

Die von Web-Data-Commons zur Verfügung gestellten RDF-Quads werden als Rohdaten für diese Arbeit herangezogen. Ausgehend von der angeführten Datenbasis werden im ersten Schritt der Datenaufbereitung alle vorkommenden RDF-Klassen aus den Rohdaten gefiltert. Die gefilterten Klassen sind für die Generierung der Primärfakten in Form von Ressource-Fakten und Statement-Fakten erforderlich, welche zwei unterschiedliche Anwendungsfälle abdecken.

Um die Ermittlung der Anzahl je vorkommender Klasse zu ermöglichen, werden Ressource-Fakten erstellt. Die Kombination der Felder *resource* und *provenance* in den Ressource-Fakten entsprechen den Primärfakten, welche alle in den Rohdaten enthaltenen Ressourcen mit deren Klasse und der zugehörigen Datenherkunft enthalten. In dieser Arbeit entspricht eine Ressour-

ce (Assoziationsklasse Resource im UML-Diagramm in Abbildung 3.2) einer Kombination von einem RDF-Term (IRI, Blank Node oder Literal) und einer Datenherkunft (IRI). Eine Ressource entspricht also der Verwendung eines RDF-Terms als Subjekt oder Objekt auf einer bestimmten Webseite (Provenance). Für ein und denselben RDF-Term kann es also viele Ressourcen geben. Es ist zu beachten, dass dieser Ressource-Begriff nur lose mit dem Begriff "Resource" im üblichen RDF-Sprachgebrauch übereinstimmt. Zu beachten ist, dass der Begriff "Resource" in dieser Arbeit nicht einheitlich verwendet wird und im logischen Schema auch als Name des Feldes für den RDF-Term einer Ressource verwendet wird.

Zu jeder Ressource wird zumindest eine Ressource-Klasse (Resourcetype in Abbildung 3.2) ermittelt. Im Falle von IRIs und Blank-Nodes werden die zugehörigen RDF-Statements mit gleicher Provenance und mit rdf:type als Prädikat zur Ermittlung herangezogen und die jeweiligen Objekte der Statements als Ressource-Klasse verwendet; eine Ressource kann also auch mehrere Ressource-Klassen haben. Sollte es keine entsprechenden rdf:type-Statements geben, wird <NA_NORDFTYPE> als Ressource-Klasse verwendet. Literale bekommen die Ressource-Klasse <LITERAL>.

Für Statement-bezogene Analysen stellt die Kombination der Felder *subject-predicate-object-provenance* in den Statement-Fakten die Primärfakten dar. Ein Statement entspricht einem RDF-Quad in den Rohdaten, ergänzt um die jeweiligen Subjekt-Klassen (subjecttype in Abbildung 3.2) und Objekt-Klassen (objecttype) entsprechend der Ressource-Klassen von Subjekt beziehungsweise Objekt. Abbildung 3.2 zeigt das entworfene Datenmodell, welches Ressourcen und Statements veranschaulicht. Bei Bedarf werden angeführte IRIs in Beispielen und Tabellen mit Namespace-Prefixes abgekürzt.

Wie oben beschrieben wird die Verwendung von ein und demselben RDF-Term (egal ob Literal, IRI oder Blank-Node) auf zwei verschiedenen Seiten als zwei verschiedene Ressourcen betrachtet und für aggregierte Kennzahlen auch mehrfach gezählt. Auf Grund dieser Design-Entscheidung hat die Entity-Resolution (also das Zusammenführen verschiedener Identifier für die selbe Real-World-Entity im Rahmen der Datenintegration) kaum Bedeutung, weil ohnehin Mehrfachvorkommen auf unterschiedlichen Seiten mehrfach gezählt werden. Entity-Resolution wird daher in dieser Arbeit nicht weiter betrachtet.

Die Anwendung des Datenmodelles wird in Abbildung 3.3 anhand des in Listing 3.1 angeführten Beispieles gezeigt. Das Beispiel enthält eine Ressource der Klasse *<http://schema.org/WebSite>* und eine Ressource der Klasse *LITERAL*. Die Ressource der Klasse *<http://schema.org/WebSite>* stellt die Subjektklasse für das im Modell enthaltene Statement dar. Dieses Statement hat das Prädikat *<http://schema.org/name>*. Der Name der Webseite ist

von der Klasse LITERAL und entspricht der Objektklasse des Statements.

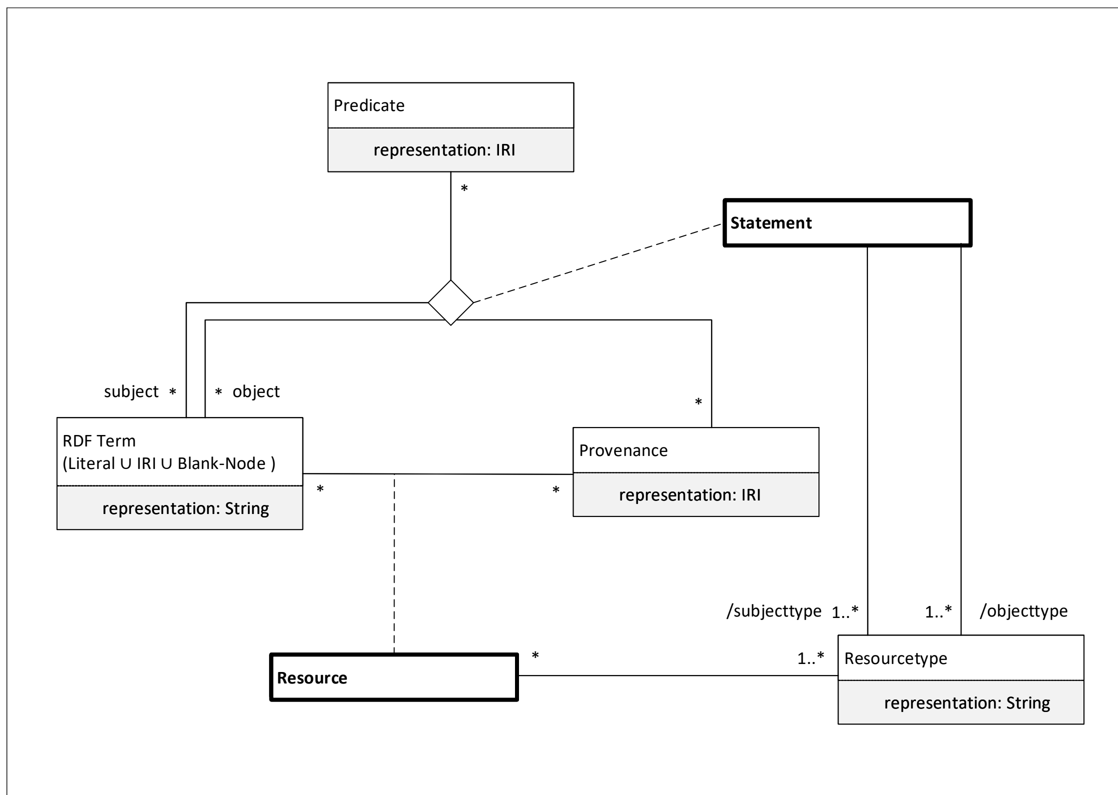


Abbildung 3.2: Primärfakten konzeptionelles Schema in UML

Die erstellten Ressource- und Statement-Fakten dienen als Ausgangspunkt für die Umsetzungen der Variante 1 “Cube” und der Variante 2 “Star”.

3.2 Erweiterte schema.org-Hierarchie

Die Kenntnis über das Vorkommen einzelner Klassen in den Rohdaten ist für eine umfassende Analyse der Verwendung von schema.org-Klassen nicht ausreichend. Daher ist die Einbeziehung der schema.org-Hierarchie notwendig. Für den Entwurf und die Umsetzung wurde die zum Zeitpunkt des Schreibens dieser Arbeit aktuelle schema.org-Hierarchie verwendet. Laufende Änderungen der schema.org-Hierarchie wurden außer Acht gelassen.

In den Rohdaten finden sich neben dem schema.org-Vokabular auch noch andere Vokabulare, die in dieser Arbeit nicht berücksichtigt werden. Werden Klassen eines anderen Vokabulars in den Daten gefunden, wird diesen Einträgen die Klasse *NA_NOTINSCHEMA* zugewiesen. Weiters werden all jene

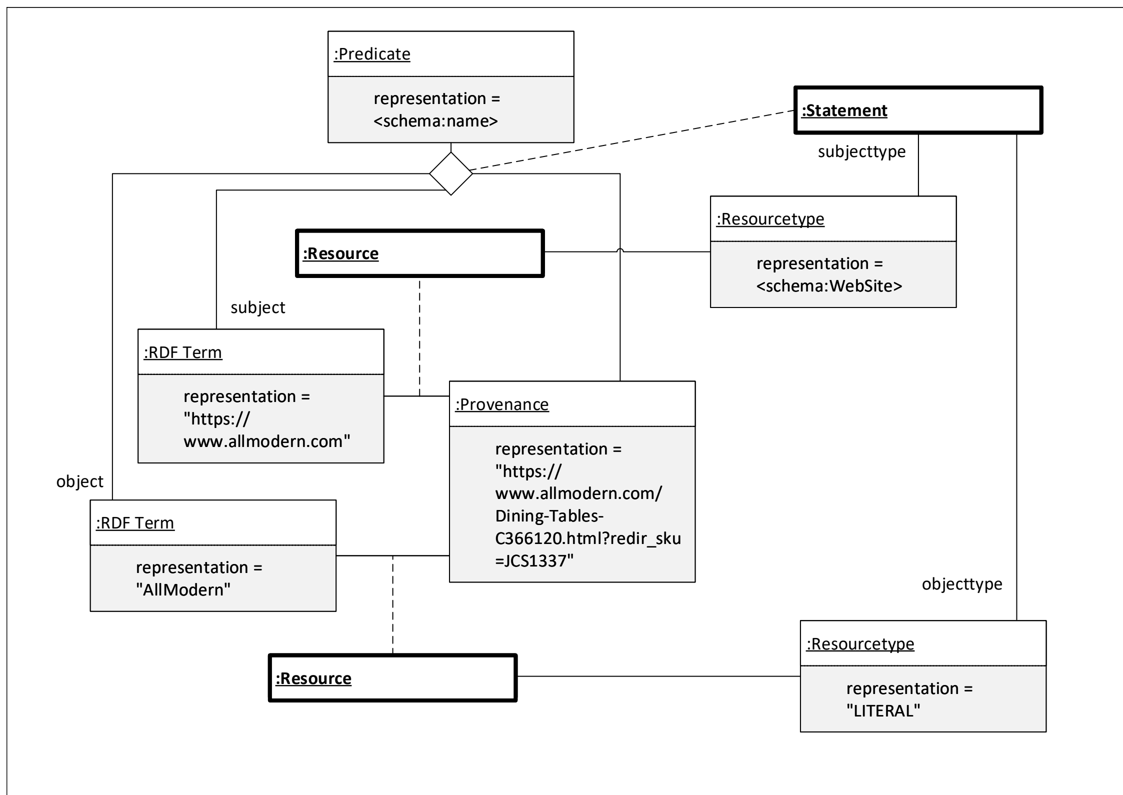


Abbildung 3.3: Beispiel zu Primärfakten als UML-Objektdiagramm

Klassen, die keinem Vokabular zuzuordnen sind, wie beispielsweise Produktbeschreibungen, deren Bilder im PNG-Format als Ressource in den Rohdaten vorkommen, mit dem Wert *NA_NORDFTYPE* gekennzeichnet. Eine letzte Transformation der Klassen erfolgt bei Datentypen wie beispielsweise String, Integer oder Double. Jene Klassen werden als *LITERAL* angeführt. Die neu eingeführten Klassen

- ANY
- NA_NORDFTYPE
- NA_NOTINSCHEMA
- LITERAL

resultieren in einer erweiterten schema.org-Hierarchie, welche in der Abbildung 3.4 dargestellt ist. Die Klasse *ANY* bildet den neuen Wurzelknoten des erweiterten Baumes.

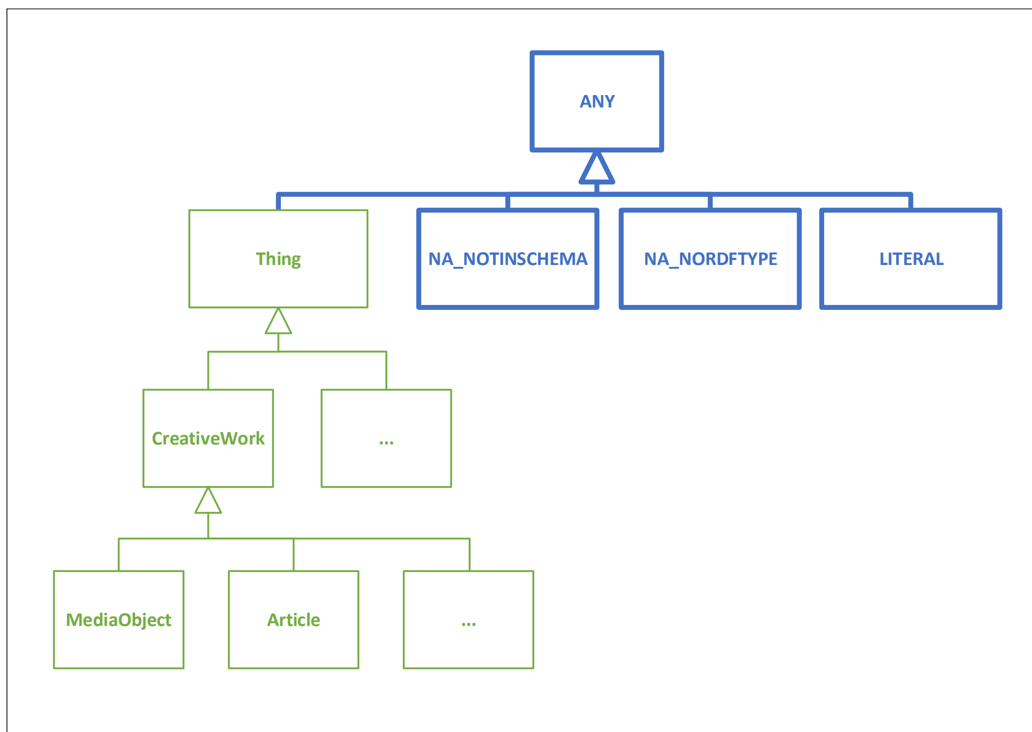


Abbildung 3.4: Erweiterte schema.org-Hierarchie

Die Einschränkungen zwischen den Klassen der erweiterten schema.org-Hierarchie ist in Abbildung 3.5 zu sehen. Die Klassen *Thing* und *NA_ NOTINSCHEMA* sind im Gegensatz zu den anderen Klassen nicht disjunkt, sondern überlappend. Eine Ressource der Klasse *Thing* kann auch der Klasse *NA_ NOTINSCHEMA* zugeordnet sein, wenn zum Beispiel für eine Ressource auch ein anderes Vokabular verwendet wird.

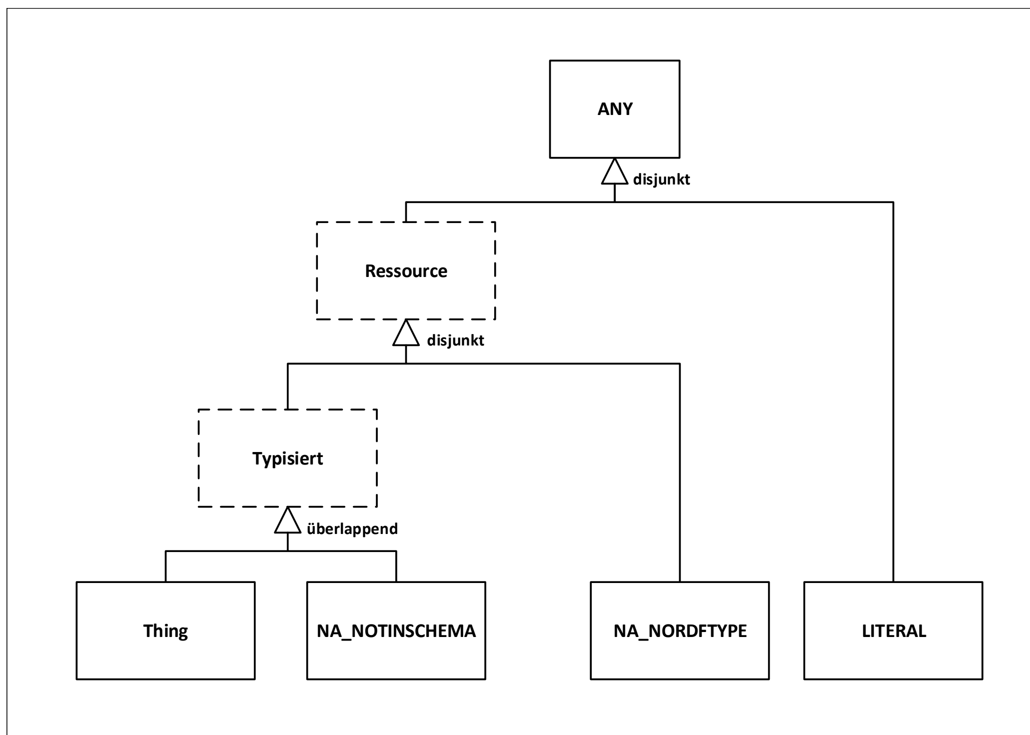


Abbildung 3.5: Einschränkungen der erweiterten schema.org-Hierarchie

Um die erweiterte schema.org-Hierarchie auch verarbeiten und in die Erzeugung der erweiterten Primärfakten und Sekundärfakten einfließen lassen zu können, wird die Hierarchie in einer eigenen Tabelle abgebildet.

Die erweiterte schema.org-Hierarchie ist für eine Aggregation über die semantischen Dimensionen *subjecttype, objecttype* und *resourcetype* der Primärfakten und Sekundärfakten noch nicht direkt geeignet. Für die Aggregation wird die transitiv-reflexive Hülle der erweiterten schema.org-Hierarchie gebildet, die in Abbildung 3.6 dargestellt ist.

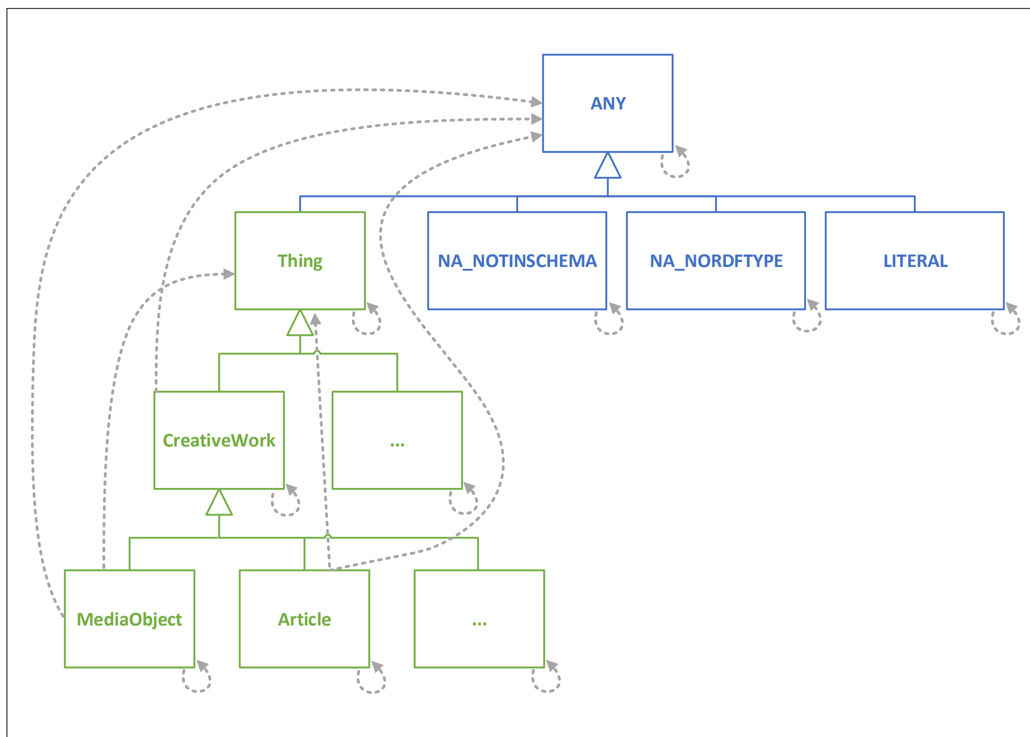


Abbildung 3.6: Transitiv-reflexive Hülle der erweiterten schema.org-Hierarchie

Durch die Bildung der transitiv-reflexiven Hülle der erweiterten schema.org-Hierarchie werden nicht nur die direkten Superklassen, sondern auch alle darüber liegenden Superklassen abgebildet.

Damit eine höhere Flexibilität bei Abfragen ermöglicht werden kann, wird neben der in Abschnitt 3.3 angeführten Variante 1 "Cube" die Variante 2 "Star", zu sehen in Abschnitt 3.4, wieder für Ressource-Fakten und Statement-Fakten, umgesetzt. Die dafür verwendete Hierarchie der semantischen Dimensionen unterscheidet sich von der transitiv-reflexiven Hierarchie, die bei der Implementierung der Variante 1 "Cube" verwendet wird. Die Interpretierbarkeit der Abfragen auf die erweiterten Primärfakten wird durch das Unterteilen der Hierarchie in Ebenen verbessert. Das Unterteilen der erweiterten Hierarchie, welche die zum Zeitpunkt des Schreibens aktuelle schema.org-Hierarchie enthält, in Ebenen resultiert in einer maximalen Tiefe von sieben Ebenen. Die für die Umsetzung der erweiterten Primärfakten verwendete Hierarchie der semantischen Dimensionen wird in Abbildung 3.7 veranschaulicht.

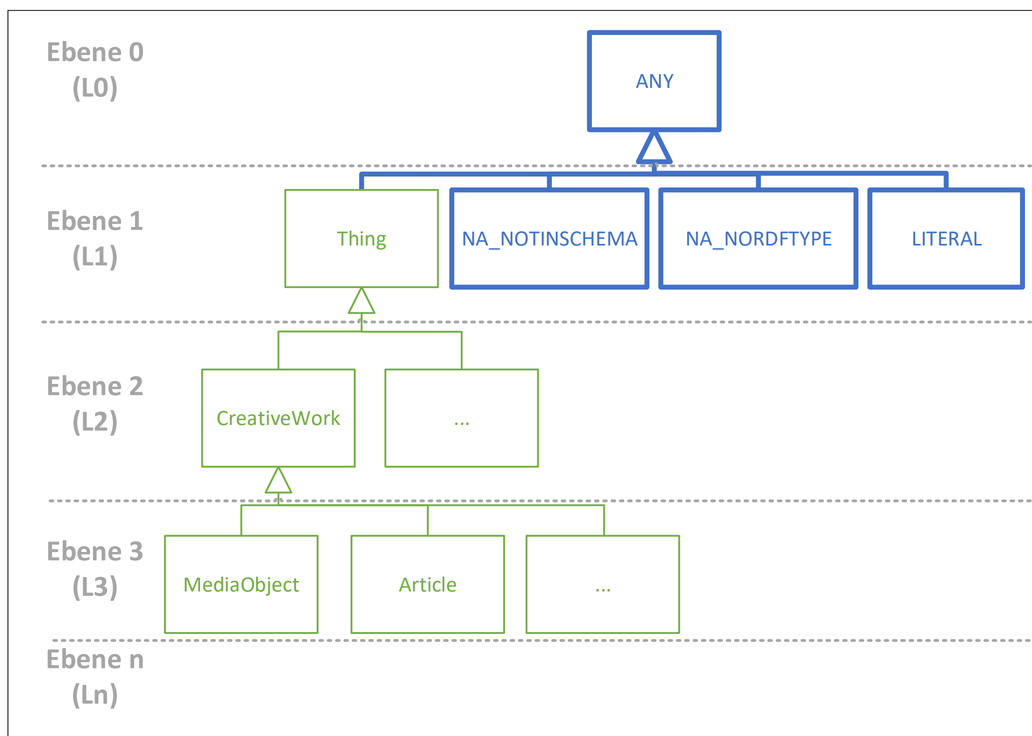


Abbildung 3.7: Erweiterte schema.org-Hierarchie unterteilt in Ebenen

Die Vorbereitung der Ressource-Fakten und Statement-Fakten sowie die Ergänzung der schema.org-Hierarchie um weitere Klassen stellen die Grundlage der Variante 1 "Cube" und der Variante 2 "Star" dar, deren Konzept in den Abschnitten 3.3 und 3.4 erläutert wird.

3.3 Variante 1 "Cube" - einfach und voraggregiert

Mit der Bereitstellung eines Cube werden die Daten von Web-Data-Commons für Analysen der verwendeten Klassen der erweiterten schema.org-Hierarchie aufbereitet. Im Cube werden alle darin vorkommenden Dimensionen auf deren Dimensionsebenen voraggregiert für Abfragen bereitgestellt [61]. Ausgehend von den bereits erstellten Ressource-Fakten und Statement-Fakten wird jeweils ein Cube erstellt. Der Ressource-Cube und der Statement-Cube bieten die Möglichkeit, sich rasch und unkompliziert einen Überblick über die für Ressourcen und Statements verwendeten Klassen der erweiterten schema.org-Hierarchie zu verschaffen, und weisen folgende Eigenschaften auf:

- Voraggregation der Daten
- Keine Dimensionslevels
- Superklassen werden immer inklusive indirekter Instanzen betrachtet
- Die Datenherkunft ist nicht im Ergebnis
- Keine Aggregation über Prädikate möglich (kein ALL-Level für die Dimension Predicate)

3.3.1 Sekundärfakten im Ressource-Cube

Abbildung 3.8 zeigt das konzeptuelle Modell des Ressource-Cube. Die Notation ist angelehnt an das Dimensional-Fact-Model (DFM) [60]. Dieser Cube stellt nur die Dimension *resourcetype* zur Verfügung, welche die verwendeten Klassen der erweiterten schema.org-Hierarchie aggregiert. Beim Erstellen des Ressource-Cube werden die eindeutigen Ressourcen und Datenherkünfte gezählt sowie deren Ressource-Klassen und die entsprechende Anzahl projiziert.

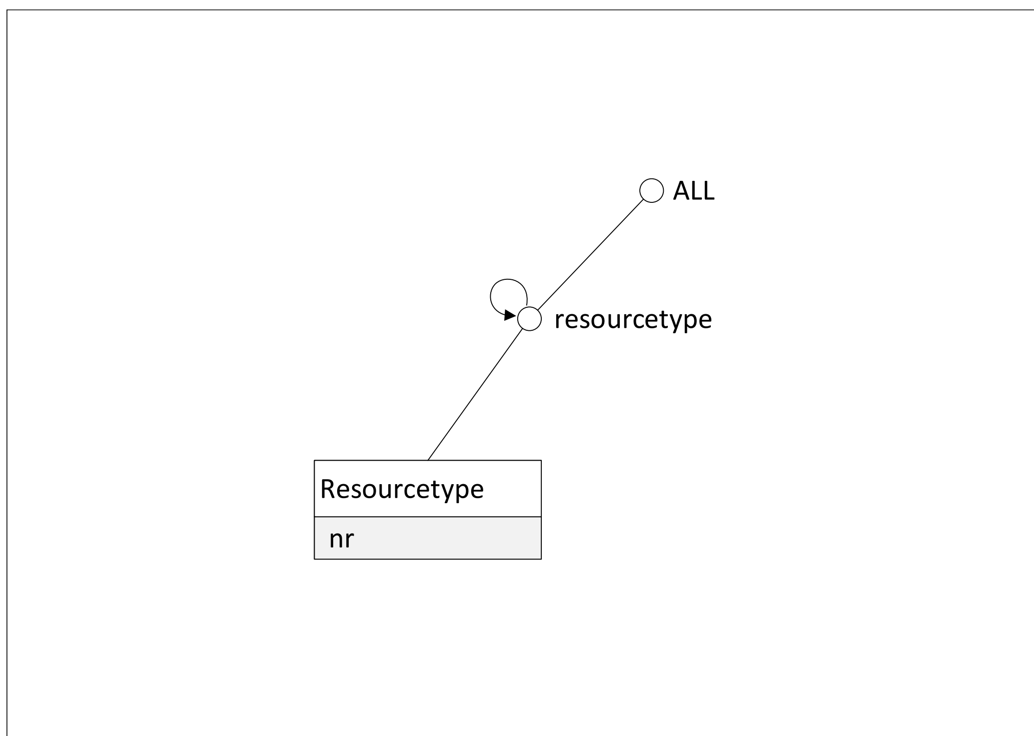


Abbildung 3.8: Konzeptuelles Modell Ressource-Cube (Variante 1)

3.3.2 Sekundärfakten im Statement-Cube

Für Analysen der Anzahl von verwendeten Subjekt- und Objektklassen und deren Verknüpfung über das Prädikat kann der Statement-Cube herangezogen werden. Das konzeptuelle Modell in Abbildung 3.9 zeigt die drei angeführten Dimensionen *subjecttype, objecttype* und *predicate* des Statement-Cube. Hier ist auch ersichtlich, dass eine Aggregation über die Dimension *predicate* nicht möglich ist.

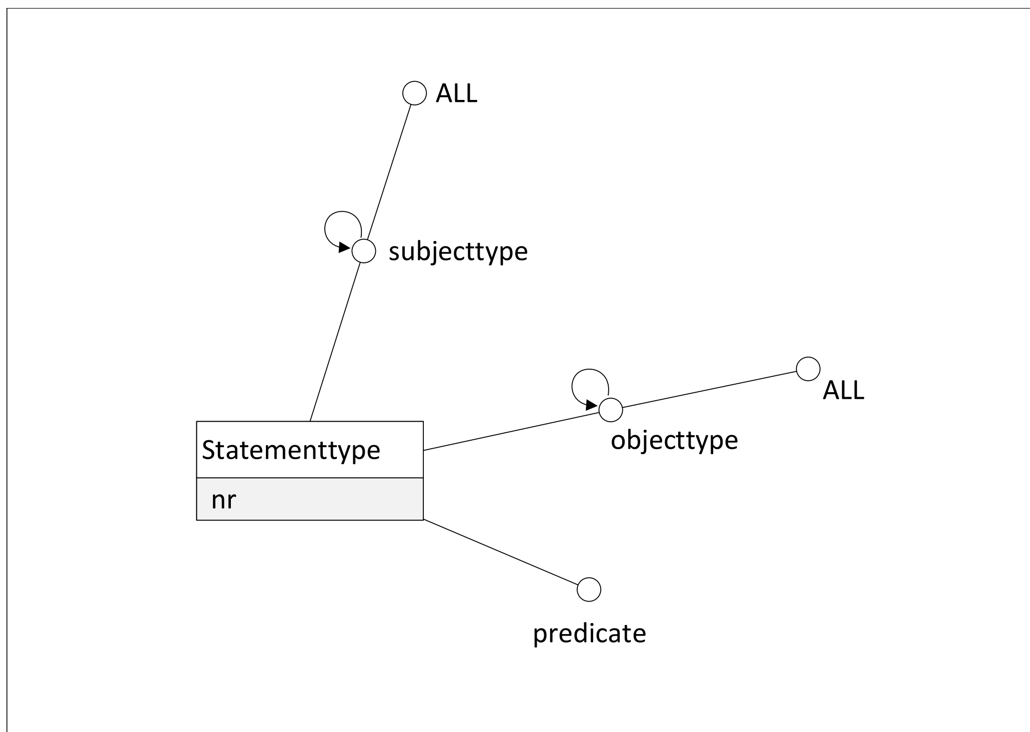


Abbildung 3.9: Konzeptuelles Modell Statement-Cube (Variante 1)

3.4 Variante 2 "Star" - für flexible Abfragen

Um umfangreiche Analysen der verwendeten Klassen der erweiterten schema.org-Hierarchie durchführen zu können, werden erweiterte Primärfakten in Form des Ressource-Star und Statement-Star erstellt, welche um die in Ebenen unterteilte erweiterte schema.org-Hierarchie ergänzt werden. Die Verwendung der in Ebenen unterteilten schema.org-Hierarchie und die Möglichkeit, die Datenherkunft in Analysen einbeziehen zu können, bietet eine höhere Flexibilität bei Abfragen. Der Ressource-Star und der Statement-Star weisen folgende Eigenschaften auf:

- Die Daten sind nicht voraggregiert. Die Fakten entsprechen erweiterten Rohdaten.
- Dimensionslevels sind vorhanden
- Unterscheidung zwischen:
  - Klassen ohne indirekte Instanzen
  - Klassen mit indirekten Instanzen
- Die Datenherkunft kann in Abfragen verwendet werden
- Aggregation über Prädikate möglich (ALL-Level implizit vorhanden)

3.4.1 Erweiterte Primärfakten im Ressource-Star

Der Ressource-Star wird durch die Erweiterung der Ressource-Fakten um die in Ebenen unterteilte erweiterte schema.org-Hierarchie erstellt. Das konzeptuelle Modell in Abbildung 3.10 zeigt, dass die Dimension *resourcetype* eine Aggregation der für Ressourcen verwendeten Klassen entlang der in Ebenen unterteilten erweiterten schema.org-Hierarchie zulässt. Zudem kann im Ressource-Star auch die Datenherkunft in Abfragen einbezogen werden.

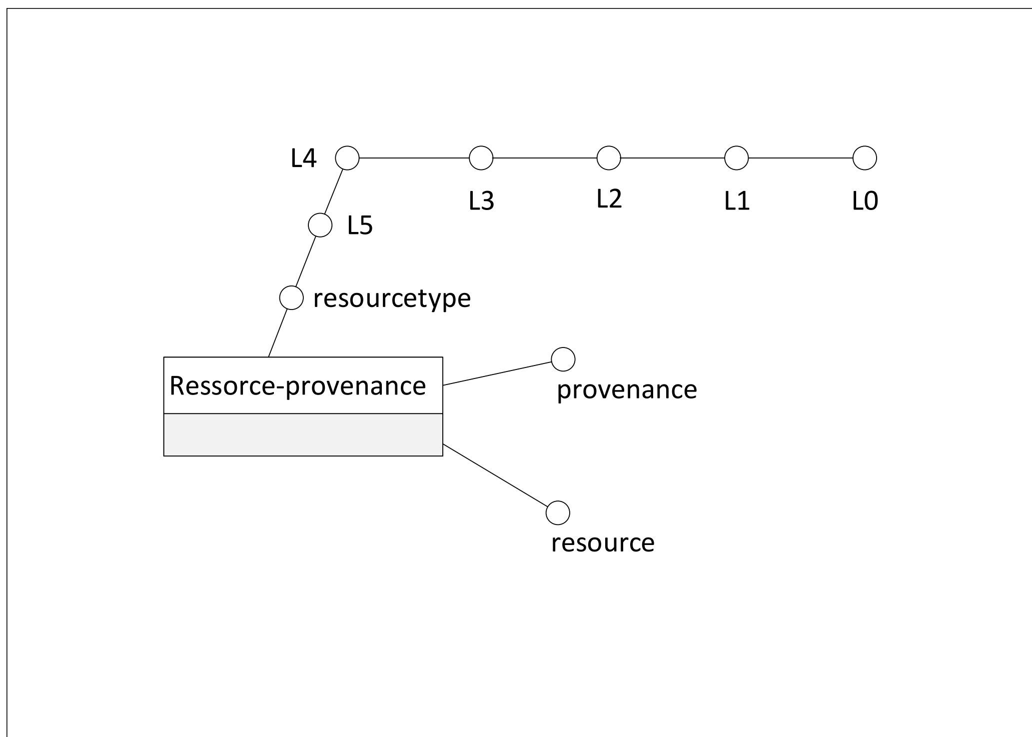


Abbildung 3.10: Konzeptuelles Modell Ressource-Star (Variante 2)

3.4.2 Erweiterte Primärfakten im Statement-Star

Als Grundlage dienen die Statement-Fakten, welche um die in Ebenen unterteilte erweiterte schema.org-Hierarchie für die semantischen Dimensionen *subjecttype* und *objecttype* erweitert werden. Der Statement-Star bietet die flexibelsten Abfrage- und umfangreichsten Analysemöglichkeiten der vorgestellten Schemata. Die Statements im Statement-Star entsprechen den RDF-Quads, die in den Rohdaten vorkommen. Der Statement-Star erlaubt neben der Verwendung der Datenherkunft in Abfragen die voneinander unabhängige Navigation über die Ebenen der semantischen Dimensionen *subjecttype* und *objecttype*.

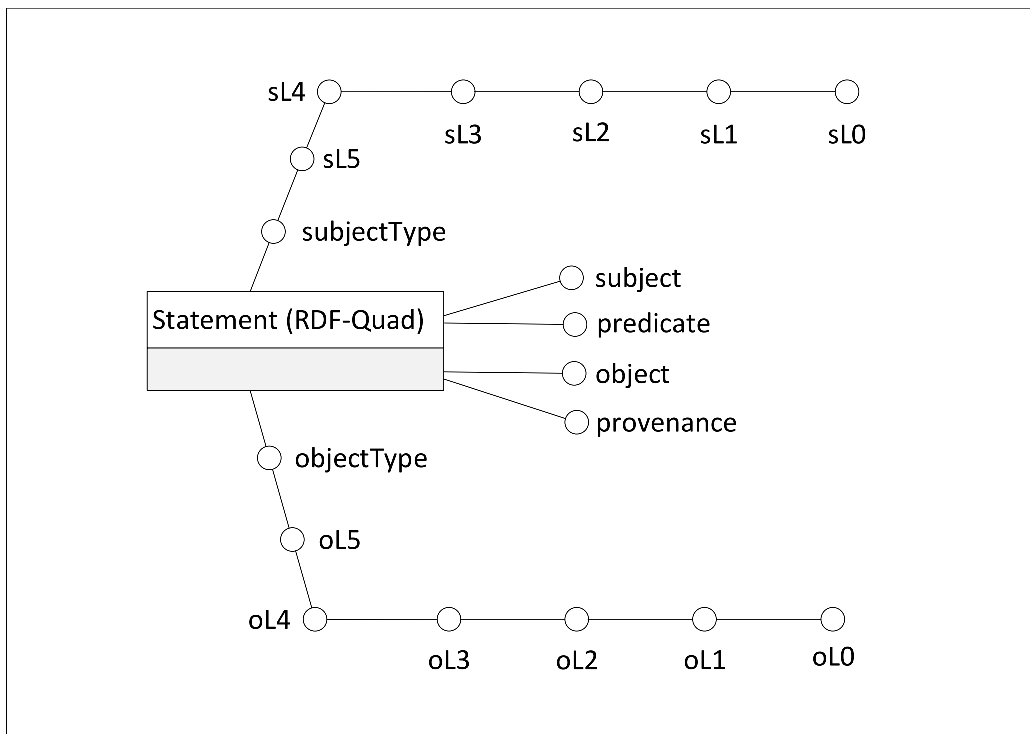


Abbildung 3.11: Konzeptuelles Modell Statement-Star (Variante 2)

3.5 Zusammenfassung

Dieses Kapitel zeigte den Entwurf für die Umsetzung der Aufgabenstellung dieser Arbeit. Ausgehend von Rohdaten, die von Web-Data-Commons [9] zur Verfügung gestellt werden, wurden Ressource-Fakten und Statement-Fakten erstellt. Nach der Erweiterung der schema.org-Hierarchie um die Klassen *NA_NOTINSCHEMA, NA_NORDFTYPE, LITERAL* und *ANY* sowie der Unterteilung der Hierarchie in Ebenen wurden die Ressource-Fakten, Statement-Fakten und die Hierarchie genutzt, um sowohl für Ressourcen als auch für Statements die Variante 1 "Cube" und die Variante 2 "Star" zu erstellen. Diese beiden Varianten können für Auswertungen bezüglich der Nutzung der schema.org-Klassen verwendet werden, wobei sich die beiden Varianten in ihrer Abfrageflexibilität unterscheiden. Abbildung 3.12 veranschaulicht den Unterschied zwischen Variante 1 "Cube" und Variante 2 "Star". Die in Variante 1 erstellten Cubes Ressource-Cube und Statement-Cube erlauben Abfragen bezüglich der Anzahl der verwendeten Klassen der erweiterten schema.org-Hierarchie. Die einzelnen Klassen werden dabei immer mit deren Subklassen betrachtet. Daher sind Abfragen bezüglich des tatsächlichen Vorkommens

einer Klasse ohne die Subklassen mitzuzählen nicht möglich. Für Abfragen dieser Art können der Ressource-Star oder der Statement-Star aus Variante 2 verwendet werden, die zusätzlich auch Abfragen auf einer bestimmten Ebene der in Ebenen unterteilten erweiterten schema.org-Hierarchie und die Einbeziehung der Datenherkunft erlauben.

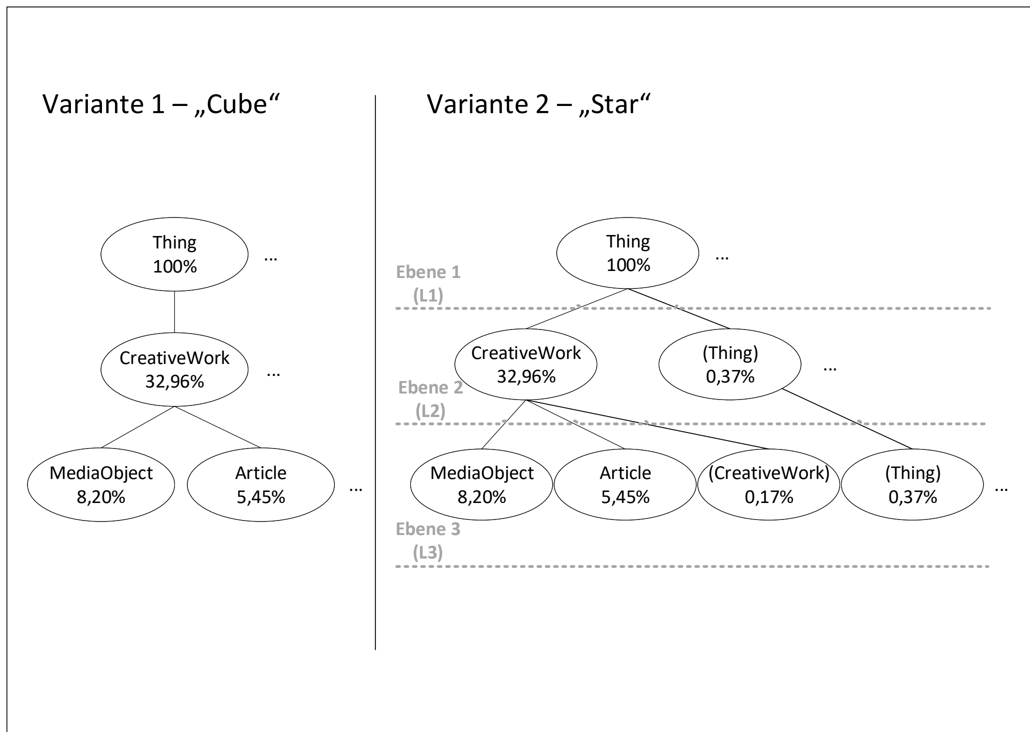


Abbildung 3.12: Vergleich von Variante 1 und Variante 2

Nachfolgendes Kapitel 4 beschreibt die Umsetzung des Entwurfes unter der Verwendung von Python, Spark [56] und Spark SQL [15].

Kapitel 4

Umsetzung

In diesem Kapitel wird die verwendete Proof-of-Concept-Umgebung erklärt und auf die Umsetzung des im vorigen Kapitel 3 gezeigten Entwurfes eingegangen.

4.1 Proof-of-Concept-Umgebung

Für die Entwicklung der Spark- und Spark-SQL-Skripte wurde eine eigene Proof-of-Concept-Umgebung (PoC-Umgebung) aufgebaut. Die in Abbildung 4.1 veranschaulichte PoC-Umgebung basiert auf vier handelsüblichen Computern mit je 16 GB RAM Arbeitsspeicher und 6 Cores. Als Betriebssystem dient CentOS Linux release 7.4.1708. Von den vier Servern werden einer als Master- und drei als Worker-Knoten verwendet. Die Worker-Knoten wurden mit je einer 3,7 TB Datendisk ausgestattet. Für die Installation der benötigten Komponenten HDFS, YARN und Spark wurde die Hadoop-Distribution von Cloudera in der Version CDH 5.12.1 Express eingesetzt. Diese CDH Version beinhaltet die Hadoop-Version 2.6.0. Da in CDH 5.12.1 nur die Spark-Version 1.6.0 zur Verfügung steht, wurde Spark 2.2.0 durch Cloudera Custom-Service-Discriptors installiert.

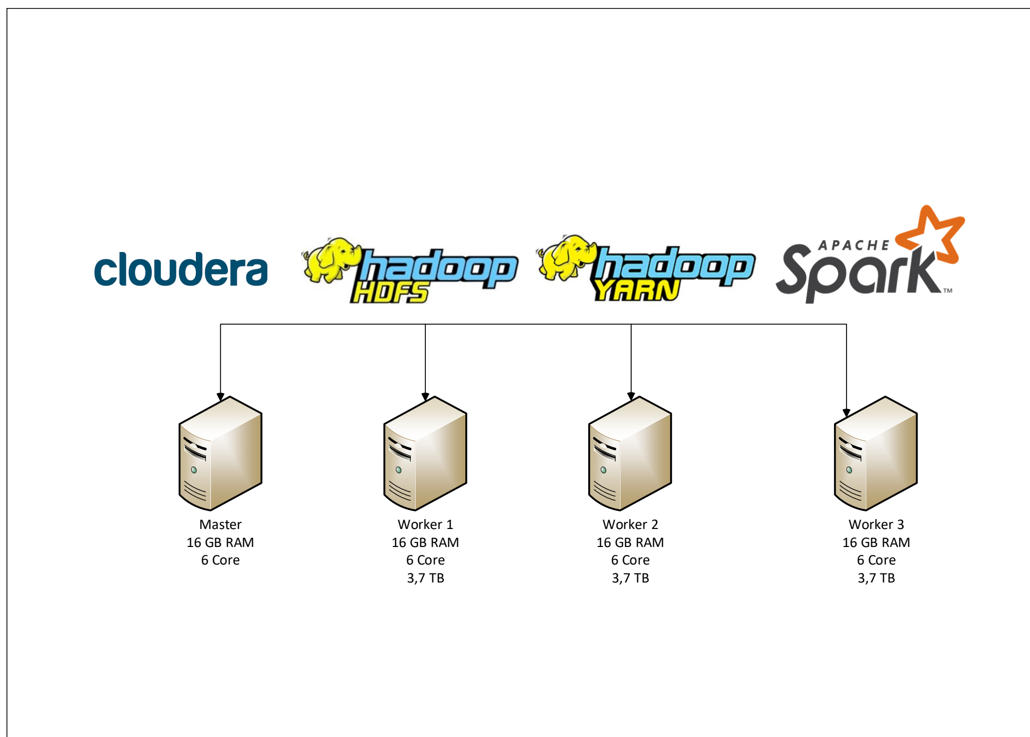


Abbildung 4.1: Proof-of-Concept-Umgebung

Für die in der Umsetzung implementierten PySpark-Skripte wird Python 2.7.13 eingesetzt. Damit die gängigsten Python-Libraries in der Umsetzung verwendet werden können, wurde die Python-Distribution Anaconda 4.3.1 über die von Cloudera zur Verfügung gestellten Installations-Pakete, so genannte Parcels, im Cluster installiert.

4.2 Vorbereitungen

Um die Daten von Web-Data-Commons im PoC-Cluster verarbeiten zu können, werden erst einzelne Dateien von Web-Data-Commons in das HDFS geladen. Dafür werden die von Web-Data-Commons veröffentlichten Links zu den einzelnen Dateien, welche RDF-Quads enthalten, verwendet und mittels dem Linux-Tool *wget* lokal auf den Master-Knoten des PoC-Clusters kopiert. Anschließend werden die heruntergeladenen gzip-Dateien entpackt und mit Hilfe der HDFS-Befehlszeilenschnittstelle in ein Verzeichnis im HDFS gelegt (siehe Listing 4.1).

Listing 4.1: Web-Data-Commons-Daten in das HDFS kopieren

```
1  cd /home/data_webDataCommons
2
3  wget http://data.dws.informatik.uni-mannheim.de
4  /structureddata/2017-12
5  /quads/dpef.html-embedded-jsonld.nq-00000.gz
6
7  unzip dpef.html-embedded-jsonld.nq-00000.gz
8
9  su -c 'hdfs dfs -put /home/data_webDataCommons
10 /dpef.html-embedded-jsonld.nq-00000
11 /data_webDataCommons' hdfs
```

In der Umsetzung wurden vier Skripte erstellt, welche die Daten von Web-Data-Commons aufbereiten und Abfragen auf die aufbereiteten Daten ausführen. Grundsätzlich können die vier Skripte unabhängig voneinander ausgeführt werden. Die Voraussetzung für die beiden Skripte, die Abfragen ausführen, sind verfügbare, entsprechend aufbereitete Daten. Die importierten Libraries und die Erzeugung der Spark-Umgebung sind für alle vier Skripte gleich und werden in Listing 4.2 angeführt.

Listing 4.2: Verwendete Python-Libraries und Spark-Umgebung

```
1  import pyspark
2
3  from pyspark.sql import SQLContext
4  from pyspark.sql import SparkSession
5  from pyspark.sql.types import *
6  from pyspark.sql import functions as F
7
8  spark = SparkSession.builder
9  .appName("pyspark2 RDF-Schema-Summarization-Cubes
10 DF Cluster")
11 .master('yarn')
12 .config('spark.executor.cores', '4')
13 .config('spark.executor.memory', '12g')
14 .config('spark.yarn.executor.memoryOverhead', '1g')
15 .config('spark.sql.shuffle.partitions', '30')
16 .config('spark.kryoserializer.buffer.max', '512')
17 .getOrCreate()
18
19 sc = spark.sparkContext
20 sqlContext = SQLContext(sc)
```

Die Spark-Umgebung wird mit einer SparkSession erzeugt. Bei der Erzeugung der SparkSession können Konfigurationsparameter übergeben werden. Die hier verwendeten Werte der Konfigurationsparameter beziehen sich auf den eingesetzten PoC-Cluster und können in anderen Umgebungen abweichen.

4.3 Erstellung von Faktentabellen

Um die RDF-Quads der Rohdaten in einem Spark-SQL-Data-Frame verwenden zu können, wird das in Listing 4.3 definierte Schema verwendet. Die Spalten des Schemas bestehen aus den in den RDF-Quads der Rohdaten vorkommenden Elementen *subject, predicate, object* und *provenance*. Das definierte Schema wird beim Erzeugen des Data-Frame *df_statement*, wo auch die Rohdaten aus dem HDFS eingelesen werden, mitgegeben. Abschließend ermöglicht die Erstellung einer sogenannten *TempView* mit dem Namen *statement*, die aus dem Data-Frame *df_statement* erzeugt wird, die Verwendung von SQL-Syntax bei Abfragen der Daten. Wegen der einfacheren Schreibweise bei der Formulierung von Abfragen durch SQL wird auch für alle anderen Data-Frames, die in diesem und den anderen drei Skripten folgen, eine *TempView* erstellt. Ein Auszug aus den Rohdaten im Data-Frame *df_statement* wird in Tabelle 4.1 auf Seite 64 gezeigt.

Listing 4.3: Schema des Data-Frame für die Rohdaten

```
1  df_schema = StructType([
2   StructField("subject", StringType(), False),
3   StructField("predicate", StringType(), False),
4   StructField("object", StringType(), False),
5   StructField("provenance", StringType(), False)])
6
7  df_statement = spark.read.csv(
8   path='hdfs:///data_webDataCommons
9   dpef.html-embedded-jsonld.nq-00000',
10  sep=' ',
11  header=False,
12  inferSchema=False,
13  schema=df_schema)
14
15 df_statement.createOrReplaceTempView("statement")
```

Nach diesem Verarbeitungsschritt sind die von Web-Data-Commons bereitgestellten Rohdaten in einem Spark-SQL-Data-Frame verfügbar. Um später die in den Rohdaten vorkommenden RDF-Klassen in den Faktentabellen

verwenden zu können, werden alle in den Rohdaten enthaltenen Ressourcen mit deren Klasse im Data-Frame *df_types* in Listing 4.4 gespeichert. Eine in SQL formulierte Abfrage auf das Data Frame *df_statement* liefert alle vorkommenden RDF-Klassen, deren Ergebnis im Data-Frame *df_types* zu finden ist.

Listing 4.4: Data-Frame der RDF-Klassifizierung

```
1  df_types = spark.sql("
2   SELECT subject AS resource, object AS type, provenance
3   FROM statement
4   WHERE predicate =
5    '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'")
6
7  df_types.createOrReplaceTempView("types")
```

Die Data-Frames *df_statement* und *df_types* bilden die Grundlage für die Erstellung der beiden im Entwurf in Kapitel 3 erklärten Ressource-Fakten und Statement-Fakten. Bevor die Faktentabelle *resourcetypes*, welche auf Ressource-Fakten basiert, erstellt werden kann, ist es notwendig, die im Data-Frame *df_statement* vorkommenden *resources* zu filtern und in das Data-Frame *df_resources* zu schreiben. Wie in der Abfrage für die Ermittlung der *resources* in Listing 4.5 ersichtlich, kann eine *resource* auch in der Spalte *object* vorkommen.

Listing 4.5: Data-Frame *df_resources*

```
1  df_resources =  spark.sql("
2   SELECT subject AS resource, provenance
3   FROM statement
4   UNION
5   SELECT object AS resource, provenance
6   FROM statement
7   WHERE predicate !=
8    '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
9   AND object is not null")
10
11 df_resources.createOrReplaceTempView("resources")
```

Das Data-Frame *df_resources* verknüpft mit dem Data-Frame *df_types* liefert die Faktentabelle *resourcetypes*, die durch das Data-Frame *df_resourcetypes* verwendet werden kann. Bei der Abfrage in Listing 4.6, mit der das Data-Frame *df_resourcetypes* generiert wird, erfolgt auch die Zuordnung der keiner RDF-Klasse zugeordneten Ressourcen zu den Klassen *LITERAL* und *NA_NORDFTYPE*. Durch die Funktion *coalesce()*, die in Listing 4.6 in Zeile 4 verwendet wird, werden Werte in der Spalte *resource* des Data-Frames

df_resourcetypes, bei denen ein Anführungszeichen am Anfang zu finden ist, als *LITERAL* erkannt. Alle anderen Ressourcen, die weder einer RDF-Klasse zugeordnet noch als *LITERAL* erkannt werden können, erhalten die Klasse *NA_NORDFTYPE*. Einen Ausschnitt der Faktentabelle *resourcetypes* ist in Tabelle 4.2 zu sehen.

Listing 4.6: Data-Frame *df_resourcetypes*

```
1  df_resourcetypes = spark.sql("
2   SELECT
3    r.resource,
4    coalesce(t.type,(CASE WHEN r.resource like '\"%'
5    THEN 'LITERAL' END),'NA_NORDFTYPE')
6    AS resourcetype,
7    r.provenance
8   FROM resources r
9   LEFT JOIN types t ON r.resource = t.resource
10  AND r.provenance = t.provenance")
11
12 df_resourcetypes.
13 createOrReplaceTempView("resourcetypes")
```

Für die Erzeugung der Faktentabelle *statementtypes* werden die Data-Frames *df_statement* und *df_types* herangezogen. In Listing 4.7 ist der Join zu sehen, mit dem die Spalten *subject* und *object* des Data-Frames *statementtypes* um die jeweilige Klasse erweitert werden. Einen Auszug der Faktentabelle *statementtypes* ist in Tabelle 4.3 abgebildet.

Listing 4.7: Data-Frame *df_statementtypes*

```
1  df_statementtypes = spark.sql("
2   SELECT
3    s.subject,
4    s.predicate,
5    s.object,
6    s.provenance,
7    subjT.resourcetype AS subjectType,
8    objT.resourcetype AS objectType
9   FROM statement s
10  JOIN resourcetypes subjT
11   ON subjT.resource = s.subject
12  AND subjT.provenance = s.provenance
13  JOIN resourcetypes objT
14   ON objT.resource = s.object
15  AND objT.provenance = s.provenance
16  WHERE s.predicate !=
```

```
17     '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'")
18
19  df_statementtypes.
20  createOrReplaceTempView("statementtypes")
```

Nach der Erstellung der beiden Faktentabellen *resourcetypes* und *statementtypes* werden die zugehörigen Data-Frames im HDFS unter der Verwendung des Dateiformates *Parquet*, wie in Listing 4.8 gezeigt, persistiert. Dadurch stehen die Faktentabellen für die Migration in andere Systeme, den Zugriff aus anderen Systemen oder den nachfolgenden Skripten der Umsetzung als Input-Daten im HDFS zur Verfügung.

Listing 4.8: Persistierung der Faktentabellen in Parquet-Dateien

```
1  df_resourcetypes.write.mode("overwrite").
2   parquet("hdfs:///results/resourcetypes.parquet")
3
4  df_statementtypes.write.mode("overwrite").
5   parquet("hdfs:///results/statementtypes.parquet")
```

Tabelle 4.1: Rohdaten von Web-Data-Commons

subject	**predicate**	**object**	**provenance**
_:genid2d61...2db0	rdf:type	schema:WebSite	<https://www.allmodern.com...
_:genid2d99...2db0	rdf:type	schema:BreadcrumbList	<https://www.allmodern.com...
_:genid2d99...2db1	rdf:type	schema:ListItem	<https://www.allmodern.com...
_:genid2d61...2db0	schema:url	<https://www.allmodern.com>	<https://www.allmodern.com...
_:genid2d61...2db0	schema:name	“AllModern”	<https://www.allmodern.com...
_:genid2d99...2db0	schema:ListElement	_:genid2d99...2db1	<https://www.allmodern.com...
_:genid2d99...2db1	schema:item	<https://www.allmodern.com...1866610.html>	<https://www.allmodern.com...
_:genid2d99...2db1	schema:postition	“0”^^xsd:integer	<https://www.allmodern.com...

Tabelle 4.2: Faktentabelle *resourcetypes*

resource	**resourcetype**	**provenance**
_:genid2d61...2db0	schema:WebSite	<https://www.allmodern.com...
_:genid2d99...2db0	schema:BreadcrumbList	<https://www.allmodern.com...
_:genid2d99...2db1	schema:ListItem	<https://www.allmodern.com...
<https://www.allmodern.com>	NA_NORDFTYPE	<https://www.allmodern.com...
“AllModern”	LITERAL	<https://www.allmodern.com...
<https://www.allmodern.com...1866610.html>	NA_NORDFTYPE	<https://www.allmodern.com...
“0”^^xsd:integer	LITERAL	<https://www.allmodern.com...

Tabelle 4.3: Faktentabelle *statementtypes*

subject	**predicate**	**object**	**provenance**	**subjectType**	**objectType**
_:genid2d61...2db0	schema:url	<https://www.allmodern.com>	<https://www.allmodern.com...	schema:WebSite	NA_NORDFTYPE
_:genid2d61...2db0	schema:name	“AllModern”	<https://www.allmodern.com...	schema:WebSite	LITERAL
_:genid2d99...2db0	schema:ListElement	_:genid2d99...2db1	<https://www.allmodern.com...	schema:BreadcrumbList	schema:ListItem
_:genid2d99...2db1	schema:item	<https://www.allmodern.com...1866610.html>	<https://www.allmodern.com...	schema:ListItem	NA_NORDFTYPE
_:genid2d99...2db1	schema:postition	“0”^^xsd:integer	<https://www.allmodern.com...	schema:ListItem	LITERAL

4.4 Erstellung von Hierarchietabellen

Für die Verwendung der schema.org-Hierarchie als semantische Dimensionen *resourcetype, subjecttype* und *objecttype* sowie für die Erweiterung der Hierarchie um die eingeführten Klassen *ANY, NA_NORDFTYPE, NA_NOTINSCHEMA* und *LITERAL* wird die öffentlich zugängliche Datei, welche die schema.org-Hierarchie enthält, wie in Listing 4.9 gezeigt, eingelesen und im Data-Frame *df_schemaorg* gespeichert. Analog zum Einlesen der Web-Data-Commons-Rohdaten wird auch für die schema.org-Hierarchie ein Schema definiert. Die Hierarchie-Daten liegen als RDF-Tripel vor, deren Elemente *subject, predicate* und *object* in das Schema übernommen werden.

Listing 4.9: Data-Frame *df_schemaorg* der schema.org-Hierarchie

```
1  df_schema_triple = StructType([
2   StructField("subject", StringType(), False),
3   StructField("predicate", StringType(), False),
4   StructField("object", StringType(), False)])
5
6  df_schemaorg = spark.read.csv(
7   path='hdfs:///schema.org_hierarchy/schema.nt',
8   sep=' ', header=False,
9   inferSchema=False,
10  schema=dfschema_triple)
11
12 df_schemaorg.createOrReplaceTempView("schemaorg")
```

In Listing 4.10 ist das SQL-Statement zu sehen, mit dem die erweiterte schema.org-Hierarchie aus Abbildung 3.4 erstellt und im Data-Frame *df_hierarchy* vorgehalten wird. Dabei werden alle *subjects*, die eine Subklasse des *objects* in einem RDF-Tripel sind, als *children* und das *object*, die übergeordneten Klassen, als *parent* ermittelt. Die schema.org-Datentypen, die im Data-Frame *df_schemaorg* vorkommen, werden bei der Hierarchieerstellung entfernt. Zudem werden die Klassen *Thing, NA_NORDFTYPE, NA_NOTINSCHEMA* und *LITERAL* unter den neuen Wurzelknoten *ANY* eingefügt.

Listing 4.10: Data-Frame *df_hierarchy* der erweiterten schema.org-Hierarchie

```
1 df_hierarchy = spark.sql("
2  SELECT
3   subject as ch,
4   object as pr
5  FROM schemaorg
6  WHERE predicate=
```

```
7     '<http://www.w3.org/2000/01/rdf-schema#subClassOf>'
8    AND subject not in (
9            '<http://schema.org/DataType>',
10           '<http://schema.org/Boolean>',
11           '<http://schema.org/False>',
12           '<http://schema.org/True>',
13           '<http://schema.org/Date>',
14           '<http://schema.org/DateTime>',
15           '<http://schema.org/Number>',
16           '<http://schema.org/Float>',
17           '<http://schema.org/Integer>',
18           '<http://schema.org/Text>',
19           '<http://schema.org/URL>',
20           '<http://schema.org/Time>'
21  )
22   UNION SELECT
23           '<http://schema.org/Thing>' as ch,
24           'ANY' AS pr
25   UNION SELECT
26           'NA_NOTINSCHEMA' as ch,
27           'ANY' AS pr
28   UNION SELECT
29           'NA_NORDFTYPE' as ch,
30           'ANY' AS pr
31   UNION SELECT
32           'LITERAL' as ch,
33           'ANY' AS pr
34  ")
35
36  df_hierarchy.createOrReplaceTempView("hierarchy")
```

Alleine mit der erweiterten schema.org-Hierarchie lassen sich noch nicht direkt Aggregationen der semantischen Dimensionen *resourcetype, subjecttype* und *objecttype* berechnen. Dafür ist die transitiv-reflexive Hülle der erweiterten schema.org-Hierarchie notwendig, die in Listing 4.11 gebildet wird.

Listing 4.11: Data-Frame *df_hierarchy_tr* der transitiv-reflexiven Hülle

```
1  df_hierarchy_tr = spark.sql("
2   SELECT ch, ch AS pr
3   FROM   hierarchy
4   UNION
5   SELECT *
6   FROM   hierarchy
7   UNION
```

```
 8   SELECT h1.ch, h2.pr
 9   FROM   hierarchy h1,
10          hierarchy h2
11   WHERE  h1.pr = h2.ch
12   UNION
13   SELECT h1.ch, h3.pr
14   FROM   hierarchy h1,
15          hierarchy h2,
16          hierarchy h3
17   WHERE  h1.pr = h2.ch
18   AND    h2.pr = h3.ch
19   UNION
20   SELECT h1.ch, h4.pr
21   FROM   hierarchy h1,
22          hierarchy h2,
23          hierarchy h3,
24          hierarchy h4
25   WHERE  h1.pr = h2.ch
26   AND    h2.pr = h3.ch
27   AND    h3.pr = h4.ch
28   UNION
29   SELECT h1.ch, h5.pr
30   FROM   hierarchy h1,
31          hierarchy h2,
32          hierarchy h3,
33          hierarchy h4,
34          hierarchy h5
35   WHERE  h1.pr = h2.ch
36   AND    h2.pr = h3.ch
37   AND    h3.pr = h4.ch
38   AND    h4.pr = h5.ch
39   UNION
40   SELECT h1.ch, h6.pr
41   FROM   hierarchy h1,
42          hierarchy h2,
43          hierarchy h3,
44          hierarchy h4,
45          hierarchy h5,
46          hierarchy h6
47   WHERE  h1.pr = h2.ch
48   AND    h2.pr = h3.ch
49   AND    h3.pr = h4.ch
50   AND    h4.pr = h5.ch
```

```
51  AND     h5.pr = h6.ch
52  ")
53
54  df_hierarchy_tr.createOrReplaceTempView("hierarchy_tr")
```

Um in weiterer Folge auch die einzelnen Ebenen der erweiterten schema.org-Hierarchie abfragen zu können, werden in Listing 4.12 die schema.org-Klassen und die eingeführten Klassen einer der Ebenen in der erweiterten schema.org-Hierarchie zugeordnet.

Listing 4.12: Data-Frame *df_hierarchy_leveled* der in Ebenen unterteilten Hierarchie

```
1   df_hierarchy_leveled = spark.sql("
2    SELECT h1.ch AS concept,
3           h1.ch AS L6,
4           h1.ch AS L5,
5           h1.ch AS L4,
6           h1.ch AS L3,
7           h1.ch AS L2,
8           h1.ch AS L1,
9           h1.pr AS L0
10   FROM   hierarchy h1
11   WHERE  h1.pr = 'ANY'
12   UNION
13   SELECT h2.ch AS concept,
14          h2.ch AS L6,
15          h2.ch AS L5,
16          h2.ch AS L4,
17          h2.ch AS L3,
18          h2.ch AS L2,
19          h1.ch AS L1,
20          h1.pr AS L0
21   FROM   hierarchy h1
22   JOIN hierarchy h2 ON h2.pr = h1.ch
23   WHERE  h1.pr = 'ANY'
24   UNION
25   SELECT h3.ch AS concept,
26          h3.ch AS L6,
27          h3.ch AS L5,
28          h3.ch AS L4,
29          h3.ch AS L3,
30          h2.ch AS L2,
31          h1.ch AS L1,
32          h1.pr AS L0
```

```
33  FROM    hierarchy h1
34   JOIN hierarchy h2 ON h2.pr = h1.ch
35   JOIN hierarchy h3 ON h3.pr = h2.ch
36  WHERE  h1.pr = 'ANY'
37  UNION
38  SELECT h4.ch AS concept,
39          h4.ch AS L6,
40          h4.ch AS L5,
41          h4.ch AS L4,
42          h3.ch AS L3,
43          h2.ch AS L2,
44          h1.ch AS L1,
45          h1.pr AS L0
46  FROM    hierarchy h1
47   JOIN hierarchy h2 ON h2.pr = h1.ch
48   JOIN hierarchy h3 ON h3.pr = h2.ch
49   JOIN hierarchy h4 ON h4.pr = h3.ch
50  WHERE  h1.pr = 'ANY'
51  UNION
52  SELECT h5.ch AS concept,
53          h5.ch AS L6,
54          h5.ch AS L5,
55          h4.ch AS L4,
56          h3.ch AS L3,
57          h2.ch AS L2,
58          h1.ch AS L1,
59          h1.pr AS L0
60  FROM    hierarchy h1
61   JOIN hierarchy h2 ON h2.pr = h1.ch
62   JOIN hierarchy h3 ON h3.pr = h2.ch
63   JOIN hierarchy h4 ON h4.pr = h3.ch
64   JOIN hierarchy h5 ON h5.pr = h4.ch
65  WHERE  h1.pr = 'ANY'
66  UNION
67  SELECT h6.ch AS concept,
68          h6.ch AS L6,
69          h5.ch AS L5,
70          h4.ch AS L4,
71          h3.ch AS L3,
72          h2.ch AS L2,
73          h1.ch AS L1,
74          h1.pr AS L0
75  FROM hierarchy h1
```

```
76   JOIN hierarchy h2 ON h2.pr = h1.ch
77   JOIN hierarchy h3 ON h3.pr = h2.ch
78   JOIN hierarchy h4 ON h4.pr = h3.ch
79   JOIN hierarchy h5 ON h5.pr = h4.ch
80   JOIN hierarchy h6 ON h6.pr = h5.ch
81  WHERE h1.pr = 'ANY'
82 ")
83
84 df_hierarchy_leveled.
85 createOrReplaceTempView("hierarchy_leveled")
```

Analog zu den Faktentabellen werden die Data-Frames der transitiv-reflexiven Hülle und der in Ebenen unterteilten, erweiterten schema.org-Hierarchie im HDFS im Dateiformat *Parquet* persistiert, siehe Listing 4.13, und für die weitere Verarbeitung bereitgestellt.

Listing 4.13: Persistierung der Hierarchien in Parquet-Dateien

```
1 df_hierarchy_tr.write.mode("overwrite").
2  parquet("hdfs:///results/hierarchy_tr.parquet")
3
4 df_hierarchy_leveled.write.mode("overwrite").
5  parquet("hdfs:///results/hierarchy_leveled.parquet")
```

4.5 Umsetzung von Variante 1

Ausgehend von den in den Abschnitten 4.3 und 4.4 aufbereiteten Daten werden zu Beginn der Cube-Generierung für *resources* und *statements* die zuvor im HDFS abgelegten Parquet-Dateien, wie in Listing 4.14 gezeigt, eingelesen.

Listing 4.14: Einlesen der Parquet-Dateien für die Cube-Generierung

```
1 df_statementtypes =
2  spark.read.parquet(
3  "hdfs:///results/statementtypes.parquet")
4
5 df_statementtypes.createOrReplaceTempView("
    statementtypes")
6
7 df_resourcetypes =
8  spark.read.parquet(
9  "hdfs:///results/resourcetypes.parquet")
10  df_resourcetypes.
11   createOrReplaceTempView("resourcetypes")
```

```
12
13 df_hierarchy_tr =
14  spark.read.parquet(
15  "hdfs:///results/hierarchy_tr.parquet")
16
17 df_hierarchy_tr.createOrReplaceTempView("hierarchy_tr")
18
19 df_hierarchy_leveled =
20  spark.read.parquet(
21  "hdfs:///results/hierarchy_leveled.parquet")
22
23 df_hierarchy_leveled.
24  createOrReplaceTempView("hierarchy_leveled")
```

Die eingelesenen Daten werden weiter zu den Cubes Ressource-Cube und Statement-Cube verarbeitet.

4.5.1 Erstellung des Ressource-Cube

In den Ressource-Fakten befinden sich in der Spalte *resourcetype* noch Werte, die keiner der Klassen der erweiterten schema.org-Hierarchie entsprechen. Diese Werte sind der Klasse *NA_NOTINSCHEMA* zuzuordnen und erhalten durch die SQL-Statements aus Listing 4.15 den entsprechenden Wert *NA_NOTINSCHEMA*. Im Data-Frame *df_resourcetypes_expanded_step1* wird der Wert *NA_NOTINSCHEMA* entsprechend zugeordnet. Da die Zuordnung der Klasse *NA_NOTINSCHEMA* zur Klasse *ANY* noch nicht abgebildet ist, wird im Dataframe *df_resourcetypes_expanded* für jedes Vorkommen der Klasse *NA_NOTINSCHEMA* in *df_resourcetypes_expanded_step1* ein Eintrag mit der Klasse *ANY* hinzugefügt.

Listing 4.15: Bereinigung nicht zugeordneter Ressourcen-Klassen für den Ressource-Cube

```
1 df_resourcetypes_expanded_step1 = spark.sql("
2 SELECT
3  r.resource,
4  r.provenance,
5  COALESCE(h.pr, 'NA_NOTINSCHEMA') AS resourcetype
6 FROM resourcetypes r
7 LEFT JOIN hierarchy_tr h ON r.resourcetype = h.ch
8 ")
9 df_resourcetypes_expanded_step1.
10 createOrReplaceTempView("resourcetypes_expanded_step1")
11
```

```
12
13 df_resourcetypes_expanded = spark.sql("
14 SELECT *
15 FROM resourcetypes_expanded_step1
16 UNION ALL
17 SELECT
18   r.resource,
19   r.provenance,
20   'ANY' AS r.resourcetype
21 FROM resourcetypes_expanded_step1 r
22 WHERE r.resourcetype = 'NA_NOTINSCHEMA'
23 ")
24 df_resourcetypes_expanded.
25 createOrReplaceTempView("resourcetypes_expanded")
```

Im Klassen-bereinigten Data-Frame *df_resourcetypes_expanded* sind alle *resourcetypes* einer Klasse der erweiterten schema.org-Hierarchie zugeordnet. Daher wird das Data-Frame mit der Bezeichnung *expanded* für die Erstellung des Ressource-Cube *cube_resources*, der wieder während der Laufzeit in einem Data-Frame gespeichert wird, verwendet. In Listing 4.16 wird der Cube für Ressourcen erstellt. Dabei wird das eindeutige Vorkommen von *resource* und *provenance* für den Ressource-Cube *cube_resources* gezählt und der *resourcetype* gruppiert.

Listing 4.16: Data-Frame des Ressource-Cube *df_cube_resources*

```
1 df_cube_resources = spark.sql("
2  SELECT
3    resourcetype,
4    COUNT(DISTINCT(resource,provenance)) AS nr
5  FROM resourcetypes_expanded
6  GROUP BY resourcetype
7  ORDER BY nr DESC")
8
9 df_cube_resources.
10  createOrReplaceTempView("cube_resources")
```

Die Tabelle 4.4 zeigt auszugsweise das Ergebnis des erstellten Ressource-Cube. Der generierte Cube kann wieder im Parquet-Format im HDFS persistiert oder direkt für Analysen genutzt werden. Abfragen auf den erstellten Ressource-Cube werden in Kapitel 5 gezeigt.

Tabelle 4.4: Auszug aus dem Ressource-Cube *cube_resources*

resourcetype	**nr**
ANY	4479306
NA_NORDFTYPE	2684934
<http://schema.org/Thing>	1333384
LITERAL	448662
<http://schema.org/CreativeWork>	439519
<http://schema.org/Intangible>	393055
<http://schema.org/ListItem>	200864
<http://schema.org/WebSite>	190912
<http://schema.org/Organization>	172321
<http://schema.org/Action>	167872
<http://schema.org/SearchAction>	161912
<http://schema.org/MediaObject>	109431
<http://schema.org/ImageObject>	100963
<http://schema.org/Person>	98470
<http://schema.org/StructuredValue>	76969

4.5.2 Erstellung des Statement-Cube

Analog zum Ressource-Cube werden bei der Erstellung des Statement-Cube noch nicht zugeordnete Klassen bereinigt. Im Statement-Cube erfolgt das einfügen der Klasse *NA_NOTINSCHEMA* und deren entsprechende Zuordnung zur Klasse *ANY* in den beiden Dimensionen *subjecttype* und *objecttype* mit den in Listing 4.17 angeführten SQL-Statements. Die bereinigten Daten sind im Data-Frame *df_statementtypes_expanded* enthalten.

Listing 4.17: Bereinigung nicht zugeordneter Ressourcen-Klassen für den Statement-Cube

```
1  df_statementtypes_expanded_step1 = spark.sql("
2  SELECT
3    s.subject,
4    s.predicate,
5    s.object,
6    s.provenance,
7    COALESCE(hs.pr, 'NA_NOTINSCHEMA') AS subjectType,
8    COALESCE(ho.pr, 'NA_NOTINSCHEMA') AS objectType
9  FROM statementtypes s
10 LEFT JOIN hierarchy_tr hs ON s.subjectType = hs.ch
11 LEFT JOIN hierarchy_tr ho ON s.objectType = ho.ch ")
12 df_statementtypes_expanded_step1.
13 createOrReplaceTempView("statementtypes_expanded_step1")
```

```
14
15
16 df_statementtypes_expanded = spark.sql("
17 SELECT *
18 FROM statementtypes_expanded_step1
19 UNION ALL
20 SELECT
21   s.subject,
22   s.predicate,
23   s.object,
24   s.provenance,
25   'ANY' AS s.subjectType,
26   s.objectType
27 FROM statementtypes_expanded_step1 s
28 WHERE subjectType = 'NA_NOTINSCHEMA'
29 UNION ALL
30 SELECT
31   s.subject,
32   s.predicate,
33   s.object,
34   s.provenance,
35   s.subjectType,
36   'ANY' AS s.objectType,
37 FROM statementtypes_expanded_step1 s
38 WHERE objectType = 'NA_NOTINSCHEMA'
39     ")
40 df_statementtypes_expanded.
41 createOrReplaceTempView("statementtypes_expanded")
```

In Listing 4.18 wird das Data-Frame *statementtypes_ expanded* dazu verwendet, um den Statement-Cube zu erstellen. Bei der Erstellung werden die eindeutigen Werte von *subject, predicate, object* und *provenance* gezählt und nach *subjecttype, predicate* und *objecttype* gruppiert.

Listing 4.18: Data-Frame des Statement-Cube *df_ cube_ statements*

```
1  df_cube_statements = spark.sql("
2  SELECT
3    subjecttype,
4    predicate,
5    objecttype,
6    COUNT(DISTINCT(subject,predicate,object,provenance))
       AS nr
7  FROM statementtypes_expanded
8  GROUP BY subjecttype, predicate, objecttype
```

```
 9   ORDER BY nr DESC")
10
11 df_cube_statements.
12   createOrReplaceTempView("cube_statements")
```

Das Ergebnis der Statment-Cube-Generierung wird auszugsweise in der Tabelle 4.5 angeführt.

Tabelle 4.5: Auszug aus dem Statement-Cube *cube_statements*

subjecttype	**predicate**	**objecttype**	**nr**
ANY	<http://schema.org/name>	ANY	740171
ANY	<http://schema.org/name>	NA_NORDFTYPE	739478
<http://schema.org/Thing>	<http://schema.org/name>	ANY	561627
<http://schema.org/Thing>	<http://schema.org/name>	NA_NORDFTYPE	561043
ANY	<http://schema.org/url>	ANY	532292
<http://schema.org/Thing>	<http://schema.org/url>	ANY	522169
ANY	<http://schema.org/url>	NA_NORDFTYPE	513108
<http://schema.org/Thing>	<http://schema.org/url>	NA_NORDFTYPE	504103
<http://schema.org/CreativeWork>	<http://schema.org/url>	ANY	359785
<http://schema.org/CreativeWork>	<http://schema.org/url>	NA_NORDFTYPE	346394
ANY	<http://schema.org/sameAs>	ANY	345410
ANY	<http://schema.org/sameAs>	NA_NORDFTYPE	345343
<http://schema.org/Thing>	<http://schema.org/sameAs>	ANY	340194
<http://schema.org/Thing>	<http://schema.org/sameAs>	NA_NORDFTYPE	340127
<http://schema.org/Organization>	<http://schema.org/sameAs>	ANY	265286

4.6 Umsetzung von Variante 2

Auch für die Erzeugung der Stars werden die Ressource-Fakten *resourcetypes* und Statement-Fakten *statementtypes* sowie die in Ebenen unterteilte Hierarchie *hierarchy_leveled*, die aufbereitet im HDFS bereitgestellt werden, verwendet. Daher werden im ersten Schritt der Star-Generierung mit Hilfe der Anweisungen in Listing 4.19 die Daten aus dem HDFS gelesen und in den Data-Frames *df_resourcetypes, df_statementtypes* und *df_hierarchy_leveled* für die weitere Verarbeitung vorgehalten.

Listing 4.19: Einlesen der Parquet-Dateien für die Star-Generierung

```
1 df_statementtypes = spark.read.parquet(
2   "hdfs:///results/statementtypes.parquet")
3   df_statementtypes.
4   createOrReplaceTempView("statementtypes")
```

```
5
6 df_resourcetypes = spark.read.parquet(
7  "hdfs:///results/resourcetypes.parquet")
8  df_resourcetypes.
9  createOrReplaceTempView("resourcetypes")
10
11 df_hierarchy_leveled = spark.read.parquet(
12  "hdfs:///results/hierarchy_leveled.parquet")
13  df_hierarchy_leveled.
14  createOrReplaceTempView("hierarchy_leveled")
```

4.6.1 Erstellung des Ressource-Star

Bei der Umsetzung des Ressource-Star werden die Ressource-Fakten mit der in Ebenen unterteilten Hierarchie verbunden, zu sehen in Listing 4.20, damit bei Abfragen durch die jeweiligen Ebenen der Hierarchie navigiert werden kann. Zusätzlich werden, wie auch schon bei den Cubes, alle noch nicht einer Klasse der erweiterten schema.org-Hierarchie zugeordneten Werte mit dem Wert *NA_NOTINSCHEMA* ersetzt. Der erstellte Ressource-Star wird im Data-Frame *star_resourcetypes* für Abfragen zur Laufzeit zur Verfügung gestellt.

Listing 4.20: Data-Frame *df_star_resourcetypes*

```
1 df_star_resourcetypes = spark.sql("
2  SELECT
3    r.*,
4    'ANY' AS L0,
5    COALESCE(h.L1, 'NA_NOTINSCHEMA') AS L1,
6    COALESCE(h.L2, 'NA_NOTINSCHEMA') AS L2,
7    COALESCE(h.L3, 'NA_NOTINSCHEMA') AS L3,
8    COALESCE(h.L4, 'NA_NOTINSCHEMA') AS L4,
9    COALESCE(h.L5, 'NA_NOTINSCHEMA') AS L5,
10   COALESCE(h.L6, 'NA_NOTINSCHEMA') AS L6
11  FROM resourcetypes r
12  LEFT JOIN hierarchy_leveled h
13  ON r.resourcetype = h.concept")
14
15 df_star_resourcetypes.
16  createOrReplaceTempView("star_resourcetypes")
```

Einen Auszug des erstellten Ressource-Star zeigen die Tabellen 4.6 und 4.7. Beim Ressource-Star ist zu beachten, dass auch falsch verwendete schema.org-Klassen den Wert *NA_NOTINSCHEMA* zugewiesen bekommen. Da

schema.org-Klassen mit einem Großbuchstaben beginnen, ist zum Beispiel im Ressource-Star *star_resourcetypes* der Wert *<http://schema.org/professionalService>* in der Spalte *resourcetype* falsch verwendet. Daher ist in den Spalten *L1 - L6* die Klasse *NA_NOTINSCHEMA* zu finden.

Tabelle 4.6: Auszug aus dem Ressource-Star *star_resourcetypes* (Teil 1)

L0	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**
ANY	schema:Thing	schema:Organization	schema:LocalBusiness	schema:FinancialService	schema:AccountingService	schema:AccountingService
ANY	schema:Thing	schema:Place	schema:LocalBusiness	schema:FinancialService	schema:AccountingService	schema:AccountingService
ANY	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA
ANY	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA
ANY	schema:Thing	schema:Place	schema:LocalBusiness	schema:Store	schema:GardenStore	schema:GardenStore
ANY	schema:Thing	schema:Organization	schema:LocalBusiness	schema:Store	schema:GardenStore	schema:GardenStore
ANY	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA

Tabelle 4.7: Auszug aus dem Ressource-Star *star_resourcetypes* (Teil 2)

resource	**resourcetype**	**provenance**
_:genid2db4b2fccc458b417d93da03cb3cfb1ef22db0	<http://schema.org/AccountingService>	<https://leatherhead.cylex-uk.co.uk/company/roy-herath-ltd-26156585.html>
_:genid2db4b2fccc458b417d93da03cb3cfb1ef22db0	<http://schema.org/AccountingService>	<https://leatherhead.cylex-uk.co.uk/company/roy-herath-ltd-26156585.html>
_:genid2d10bccb16c18f4c9f9685273bcff939692db0	<http://schema.org/Children..Earrings>	<http://www.shakeela.com/jewelry/Earrings.shtml>
_:genid2dd9f2d28d565544a6825fdd33e3545dff2db0	<http://schema.org/localBusiness>	<http://www.olivepromotions.com/blog/engaging-all-5-senses-with-promo-products>
_:genid2d46a32da6fbe24483bbba04fc1bf297c32db0	<http://schema.org/GardenStore>	<https://duns.cylex-uk.co.uk/company/the-lawnmower-centre-duns-17859560.html>
_:genid2d46a32da6fbe24483bbba04fc1bf297c32db0	<http://schema.org/GardenStore>	<https://duns.cylex-uk.co.uk/company/the-lawnmower-centre-duns-17859560.html>
_:genid2db0b39fa96e4d4854a71e8fe85968eac42db0	<http://schema.org/professionalService>	<http://surelocknetworkauctions.com/aboutus/uk-property-auction-podcasts/>

4.6.2 Erstellung des Statement-Star

Die Verknüpfung der Statement-Fakten *statementtypes* mit der in Ebenen unterteilten erweiterten schema.org-Hierarchie für die semantischen Dimensionen *subjecttype* und *objecttype* resultiert im Statement-Star. Das SQL-Statement für die Erstellung des Statement-Star ist in Listing 4.21 zu sehen.

Listing 4.21: Data-Frame *df_star_statementtypes*

```
1  df_star_statementtypes = spark.sql("
2   SELECT
3     s.*,
4     'ANY' AS sL0,
5     COALESCE(hSubj.L1, 'NA_NOTINSCHEMA') AS sL1,
6     COALESCE(hSubj.L2, 'NA_NOTINSCHEMA') AS sL2,
7     COALESCE(hSubj.L3, 'NA_NOTINSCHEMA') AS sL3,
8     COALESCE(hSubj.L4, 'NA_NOTINSCHEMA') AS sL4,
9     COALESCE(hSubj.L5, 'NA_NOTINSCHEMA') AS sL5,
10    COALESCE(hSubj.L6, 'NA_NOTINSCHEMA') AS sL6,
11    'ANY' AS oL0,
12    COALESCE(hObj.L1, 'NA_NOTINSCHEMA') AS oL1,
13    COALESCE(hObj.L2, 'NA_NOTINSCHEMA') AS oL2,
14    COALESCE(hObj.L3, 'NA_NOTINSCHEMA') AS oL3,
15    COALESCE(hObj.L4, 'NA_NOTINSCHEMA') AS oL4,
16    COALESCE(hObj.L5, 'NA_NOTINSCHEMA') AS oL5,
17    COALESCE(hObj.L6, 'NA_NOTINSCHEMA') AS oL6
18  FROM statementtypes s
19  LEFT JOIN hierarchy_leveled hSubj
20  ON s.subjectType = hSubj.concept
21  LEFT JOIN hierarchy_leveled hObj
22  ON s.objectType = hObj.concept")
23
24 df_star_statementtypes.
25  createOrReplaceTempView("star_statementtypes")
```

Die Tabellen 4.8, 4.9 und 4.10 zeigen einen Auszug aus dem erstellten Statement-Star. Auch in den semantischen Dimensionen des Statement-Star können, so wie in den erstellten Cubes, falsch verwendete schema.org-Klassen vorkommen, die ebenfalls den Wert *NA_NOTINSCHEMA* zugewiesen bekommen.

Tabelle 4.8: Auszug aus dem Statement-Star *star_ statementtypes* (Teil 1)

sL0	sL1	sL2	sL3	sL4	sL5	sL6
ANY	schema:Thing	schema:Organization	schema:Organization	schema:Organization	schema:Organization	schema:Organization
ANY	schema:Thing	schema:Product	schema:Product	schema:Product	schema:Product	schema:Product
ANY	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA
ANY	schema:Thing	schema:Person	schema:Person	schema:Person	schema:Person	schema:Person
ANY	schema:Thing	schema:Person	schema:Person	schema:Person	schema:Person	schema:Person
ANY	schema:Thing	schema:CreativeWork	schema:MusicComposition	schema:MusicComposition	schema:MusicComposition	schema:MusicComposition
ANY	schema:Thing	schema:Organization	schema:PerformingGroup	schema:MusicGroup	schema:MusicGroup	schema:MusicGroup

Tabelle 4.9: Auszug aus dem Statement-Star *star_ statementtypes* (Teil 2)

subject	predicate	object	provenance	subjectType	objectType
_:genid2..db0	schema:aggregateRating	_:genid2..db2	<https://www.bark.com/../ruislip/>	schema:Organization	schema:aggregateRating
_:genid2..db8	schema:aggregateRating	_:genid2..db9	<https://www.betzold.at/prod/53098/>	schema:Product	schema:aggregateRating
<http://www.finance..%20Loans>	<http://purl.org/../#includes>	_:genid2..db1	<http://www.silverfinance.net/personal-loans>	<http://purl.org/../#Offering>	<http://purl.org/../#Individual>
_:genid2..db0	schema:homeLocation	_:genid2..db1	<https://www.expertissim.com/.../12205242>	schema:Person	schema:City
_:genid2..b88	schema:Weight	_:genid2..b91	<http://www.maxpreps.com/../schedule.htm>	schema:Person	schema:QuantitativeValue
<https://yeut../116147>	schema:sameAs	<https://yeut../116147>	<https://yeutienganh.com/../116147>	schema:MusicComposition	schema:MusicComposition
<https://www.letr../dido/>	schema:track	<https://www.letr..dido/1857442/>	<https://www.letras.com/mc-dido/>	schema:MusicGroup	schema:MusicRecording

Tabelle 4.10: Auszug aus dem Statement-Star *star_ statementtypes* (Teil 3)

oL0	oL1	oL2	oL3	oL4	oL5	oL6
ANY	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA
ANY	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA
ANY	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA	NA_NOTINSCHEMA
ANY	schema:Thing	schema:Place	schema:AdministrativeArea	schema:City	schema:City	schema:City
ANY	schema:Thing	schema:Intangible	schema:StructuredValue	schema:QuantitativeValue	schema:QuantitativeValue	schema:QuantitativeValue
ANY	schema:Thing	schema:CreativeWork	schema:MusicComposition	schema:MusicComposition	schema:MusicComposition	schema:MusicComposition
ANY	schema:Thing	schema:CreativeWork	schema:MusicRecording	schema:MusicRecording	schema:MusicRecording	schema:MusicRecording

4.7 Zusammenfassung

In diesem Kapitel wurde zu Beginn die PoC-Umgebung erklärt, in der die Spark-Skripte in Python entwickelt und getestet wurden. Ausgehend von den Vorbereitungen, zu denen das Herunterladen und die Bereitstellung der Daten sowie die Konfiguration der SparkSession zählen, wurden die einzelnen Verarbeitungsschritte anhand von Code-Listings gezeigt. Die Basis für die erstellten Cubes und Stars bilden zwei Faktentabellen, die zum einen für in den Rohdaten vorkommende Ressourcen und zum anderen für vorkommende Statements erstellt wurden. Für die Verwendung der schema.org-Hierarchie als semantische Dimensionen in den Cubes und Stars war die Erweiterung um eigene Klassen, die Erzeugung der transitiv-reflexiven Hülle der erweiterten schema.org-Hierarchie und deren Unterteilung in verschiedene Ebenen notwendig. Aufbauend auf den Ressource- und Statement-Fakten sowie den erstellten Hierarchien wurden die Cubes und Stars für Ressourcen und Statements generiert, die erste Auswertungen bezüglich der Verwendung der schema.org-Hierarchie in verschiedenen Rohdatenformaten, die in Kapitel 5 durchgeführt werden, ermöglichen.

Kapitel 5

Auswertung und Evaluierung

Dieses Kapitel beschreibt eine vorläufige Evaluierung von Funktionalität und Skalierbarkeit des Ansatzes. In Abschnitt 5.1 wird die Plausibilität der Abfrageergebnisse überprüft, indem die Ergebnisse von Variante 1 und Variante 2 verglichen werden. Die Ergebnisse sind in beiden Varianten identisch, was als Hinweis auf die Plausibilität eben dieser gewertet werden kann. In Abschnitt 5.2 wird der Prototyp auf jeweils einem kleinen Ausschnitt aus Web-Data-Commons für drei verschiedene Web-Datenformate (JSON-LD, RDFa, Microdata) ausgeführt und ein vergleichender Überblick über die Verwendung in diesen drei Formaten gegeben. Für jedes der drei Formate wird die jeweilige Datei mit der ID 00000 verwendet, die unter `http://data.dws.informatik.uni-mannheim.de/structureddata/2017-12/quads/` zu finden ist. Für den Vergleich der drei Formate wird der Cube *cube_resources* herangezogen. Weitere Beispiele zu Abfragen und Abfrageergebnissen in Abschnitt 5.3 illustrieren die Verwendung von RDF-Summarization-Cubes. Die Inspektion der Abfrageergebnisse in den Abschnitten 5.2 und 5.3 stellt eine weitere Plausibilitätsprüfung dar. Eine weitergehende Evaluierung der Funktionalität des Ansatzes sowie eine weitergehende Validierung der Abfrageergebnisse geht über den Rahmen dieser Arbeit hinaus. Um in Abschnitt 5.4 Skalierbarkeit und Portierbarkeit zu zeigen, wird der Prototyp in der Microsoft-Azure-Cloud auf eine größere Datenmenge ausgeführt.

Zu Beginn dieser Arbeit war der Web-Data-Commons-Korpus 2016-10 aktuell, mit dem auch die ersten Evaluierungen durchgeführt wurden. Da mittlerweile ein neuer Korpus, 2017-12, erstellt wurde, werden auch die bereits durchgeführten Auswertungen mit dem neuen Korpus durchgeführt, um für alle drei untersuchten Formate aktuelle Daten zu verwenden.

5.1 Vergleich der Ergebnisse von Variante 1 und Variante 2

Als erste Plausibilitätsprüfung werden Abfrageergebnisse auf die erstellten Cubes und Stars miteinander verglichen. Die Ergebnisse sollen in beiden Varianten die geleichen sein. Für die Plausibilitätsprüfung des Cube *cube_resources* und des Star *star_resourcetypes* werden die auf Ebene L1 vorkommenden Klassen der erweiterten schema.org-Hierarchie sowie die Klassen *<http://schema.org/CreativeWork>* auf Ebene L2 und *<http://schema.org/AboutPage>* auf Ebene L4 gezählt. Das Ergebnis der Anzahl der vorkommenden Instanzen einer Klasse im Cube *cube_resources* muss mit der Anzahl der vorkommenden Instanzen im Star *star_resourcetypes* auf Ebene L1 beziehungsweise auf Ebene L2 und Ebene L4 für die Klassen *<http://schema.org/CreativeWork>* und *<http://schema.org/AboutPage>* übereinstimmen. Listing 5.1 zeigt die beiden Abfragen für die Validierung des Cube *cube_resources* und des Star *star_resourcetypes*, in welche die jeweiligen Klassen der erweiterten schema.org-Hierarchie für die Validierung eingesetzt werden. Aufgrund der leichteren Lesbarkeit werden bei den folgenden Beispielen nur die SQL-Statements in den Listings angeführt. Die Ergebnisse der Plausibilitätsprüfung, die mit JSON-LD-Daten durchgeführt wurde, sind in Tabelle 5.1 gegenübergestellt und zeigen, dass die Anzahl der geprüften Instanzen im Cube und im Star der Ressourcen übereinstimmen.

Listing 5.1: Abfrage 1 zum Ergebnisvergleich

```
1  SELECT nr
2  FROM cube_resources
3  WHERE resourcetype = '<http://schema.org/Thing>'
4
5  SELECT COUNT(DISTINCT(resource, provenance)) as nr
6  FROM star_resourcetypes
7  WHERE L1 = '<http://schema.org/Thing>'
```

Tabelle 5.1: Vergleichsergebnis zwischen Ressource-Cube und Ressource-Star

resourcetype	**Cube**	**Star**
<http://schema.org/Thing>	1333384	1333384
NA_NORDFTYPE	2684934	2684934
NA_NOTINSCHEMA	12480	12480
LITERAL	448662	448662
<http://schema.org/CreativeWork>	439519	439519
<http://schema.org/AboutPage>	2	2

Die in Listing 5.2 angeführten Abfragen werden für die Validierung des Cube *cube_statements* und des Star *star_statementtypes* verwendet. Analog zur Validierung des Cube und Star der Ressourcen, muss auch bei den Statements die Anzahl der Instanzen der geprüften Klassen der erweiterten schema.org-Hierarchie übereinstimmen. Im Gegensatz zur Validierung der Ressourcen muss bei der Validierung der Statements die Anzahl der *subjecttype, predicate* und *objecttype* Kombinationen des Cube *cube_statements* mit der im Star *star_statementtypes*, auf der entsprechenden Ebene der erweiterten schema.org-Hierarchie, übereinstimmen. Im in Listing 5.2 angeführten Beispiel werden Kombinationen, die *ANY* als Subjektklasse, *<http://schema.org/name>* als Prädikat und *ANY* als Objektklasse aufweisen, abgefragt. Da sich *ANY* in der erweiterten schema.org-Hierarchie auf Ebene L0 befindet, werden die Felder *sL0* für die Subjektklasse und *oL0* für die Objektklasse im Star *star_statementtypes* abgefragt. Tabelle 5.2 zeigt die für die Validierung verwendeten *subjecttype, predicate* und *objecttype* Kombinationen und die Ergebnisse der Validierung. Die Anzahl der überprüften *subjecttype, predicate* und *objecttype* Kombinationen des Cube *cube_statements* stimmt mit den entsprechenden *subjecttype, predicate* und *objecttype* Kombinationen im Star *star_statementtypes* überein.

Listing 5.2: Abfrage 2 zum Ergebnisvergleich

```
1  SELECT *
2  FROM cube_statements
3  WHERE subjecttype = 'ANY'
4    AND predicate = '<http://schema.org/name>'
5    AND objecttype = 'ANY'
6
7  SELECT COUNT(DISTINCT(subject,predicate,object,
     provenance))
8  FROM star_statementtypes
9  WHERE sL0 = 'ANY'
10   AND predicate = '<http://schema.org/name>'
11   AND oL0 = 'ANY'
```

Tabelle 5.2: Vergleichsergebnis zwischen Statement-Cube und Statement-Star

subjecttype	**predicate**	**resourcetype**	**Cube**	**Star**
ANY	<http://schema.org/name>	ANY	740171	740171
<http://schema.org/SocialMediaPosting>	<http://schema.org/comment>	<http://schema.org/CreativeWork>	4958	4958
<http://schema.org/Organization>	<http://schema.org/sameAs>	NA_NORDFTYPE	265261	265261
<http://schema.org/BreadcrumbList>	<http://schema.org/itemListElement>	<http://schema.org/ListItem>	187526	187526
<http://schema.org/Event>	<http://schema.org/location>	<http://schema.org/Place>	6083	6083
<http://schema.org/NewsArticle>	<http://schema.org/datePublished>	LITERAL	30659	30659

5.2 Verwendung von schema.org-Klassen in Microdata, JSON-LD und RDFa

In diesem Abschnitt wird die Auswertung der Verwendung der schema.org-Klassen beschrieben. Für die Auswertung wird je eine Rohdatendatei von Web-Data-Commons [8] der gängigen Formate von strukurierten Web-Daten Microdata, JSON-LD und RDFa herangezogen und auf das Vorkommen und die Häufigkeit von schema.org-Klassen untersucht.

5.2.1 Microdata

Das Ergebnis der Abfrage des mit einer Microdata-Datei erstellten Cube aus Listing 5.3 in Tabelle 5.3 zeigt, dass 2812715 Klasseninstanzen im Cube enthalten sind. Ein Großteil der Klassen sind keine RDF-Klassen und Literale. In Summe befinden sich 552588 schema.org-Instanzen in der ausgewerteten Microdata-Datei. 253011 Einträge müssen der Klasse *NA_NOTINSCHEMA* zugeordnet werden.

Listing 5.3: Abfrage des Ressource-Cube zur Ermittlung der 20 häufigsten Klassen

```
1 SELECT *
2 FROM cube_resources
3 ORDER BY nr DESC
4 LIMIT 20
```

Tabelle 5.3: Ergebnis der Abfrage der 20 häufigsten Klassen in einer Microdata-Datei

resourcetype	**nr**
ANY	2812715
NA_NORDFTYPE	1146899
LITERAL	860219
<http://schema.org/Thing>	552588
NA_NOTINSCHEMA	253011
<http://schema.org/Intangible>	202221
<http://schema.org/CreativeWork>	157571
<http://schema.org/Product>	75357
<http://schema.org/Offer>	66871
<http://schema.org/Organization>	50178
<http://schema.org/Article>	45847
<http://schema.org/ListItem>	45088
<http://schema.org/StructuredValue>	39286
<http://schema.org/Person>	39239
<http://schema.org/Place>	38207
<http://schema.org/WebPageElement>	31304
<http://schema.org/ContactPoint>	28718
<http://schema.org/PostalAddress>	28529
<http://schema.org/MediaObject>	26363
<http://schema.org/LocalBusiness>	23896

Bei Microdata fällt auf, dass die Klasse *LITERAL* im Vergleich zu <http://schema.org/Thing> häufiger vorkommt. Außerdem sind viele Einträge in den Rohdaten, die keiner schema.org-Klasse und damit der Klasse *NA_NOTINSCHEMA* zugeordnet sind.

5.2.2 JSON-LD

Der mit den aktuellen JSON-LD-Daten erstellte Cube *cube_resources* beinhaltet die in Tabelle 5.4 ersichtliche Verteilung der Klassen der erweiterten schema.org-Hierarchie. Die Abfrage ist in Listing 5.3 angeführt. Die insgesamt 4479306 Einträge verteilen sich auf 2684934 NA_NORDFTYPE, 1333384 <http://schema.org/Thing>, dem Wurzelknoten der nicht erweiterten schema.org-Hierarchie, und 448662 Literale. Die Klasse NA_NOTINSCHEMA ist nicht unter den 20 am häufigsten auftretenden Klassen.

Tabelle 5.4: Ergebnis der Abfrage der 20 häufigsten Klassen in einer JSON-LD-Datei

resourcetype	**nr**
ANY	4479306
NA_NORDFTYPE	2684934
<http://schema.org/Thing>	1333384
LITERAL	448662
<http://schema.org/CreativeWork>	439519
<http://schema.org/Intangible>	393055
<http://schema.org/ListItem>	200864
<http://schema.org/WebSite>	190912
<http://schema.org/Organization>	172321
<http://schema.org/Action>	167872
<http://schema.org/SearchAction>	161912
<http://schema.org/MediaObject>	109431
<http://schema.org/ImageObject>	100963
<http://schema.org/Person>	98470
<http://schema.org/StructuredValue>	76969
<http://schema.org/Article>	72780
<http://schema.org/ItemList>	56491
<http://schema.org/BreadcrumbList>	53215
<http://schema.org/Place>	41801
<http://schema.org/WebPage>	40747

5.2.3 RDFa

Wie auch für Microdata und JSON-LD wird für die Auswertung von RDFa-Daten ein eigener Cube *cube_resources* erstellt. In Tabelle 5.5 ist die Auswertung der 20 meistverwendeten Klassen der erweiterten schema.org-Hierarchie in einer RDFa Rohdatendatei zu sehen.

Tabelle 5.5: Ergebnis der Abfrage der 20 häufigsten Klassen in einer RDFa-Datei

resourcetype	**nr**
ANY	3346382
NA_NORDFTYPE	2558369
NA_NOTINSCHEMA	731594
LITERAL	56066
<http://schema.org/Thing>	2920
<http://schema.org/CreativeWork>	2427
<http://schema.org/Article>	1425
<http://schema.org/Blog>	669
<http://schema.org/SocialMediaPosting>	661
<http://schema.org/BlogPosting>	661
<http://schema.org/NewsArticle>	629
<http://schema.org/Person>	171
<http://schema.org/Organization>	120
<http://schema.org/Place>	119
<http://schema.org/LocalBusiness>	114
<http://schema.org/Library>	110
<http://schema.org/MediaObject>	108
<http://schema.org/VideoObject>	106
<http://schema.org/Event>	90
<http://schema.org/Intangible>	79

Die Auswertung der RDFa Rohdaten zeigt eine sehr geringe Nutzung des schema.org-Vokabulars. Von insgesamt 3346382 Instanzen der erweiterten schema.org-Hierarchie fallen nur 2920 auf <http://schema.org/Thing>. Weit vor dem Wurzelknoten der nicht erweiterten schema.org-Hierarchie liegen die Klassen *NA_NORDFTYPE, NA_NOTINSCHEMA* und *LITERAL*.

5.2.4 Vergleich der verwendeten schema.org-Klassen in den Formaten Microdata, JSON-LD und RDFa

Bei der durchgeführten Auswertung wurden die in der Umsetzung gezeigten Skripten dazu verwendet, um einen Einblick in die Verwendung der schema.org-Hierarchie zu bekommen. Dazu wurde je eine Rohdatendatei der strukturierten Web-Daten-Formate Microdata, JSON-LD und RDFa untersucht. Tabelle 5.6 zeigt eine Gegenüberstellung der Formate und wie häufig die vier auf Ebene L1 der erweiterten schema.org-Hierarchie zu findenden Klassen *<http://schema.org/Thing>, NA_NORDFTYP, NA_NOTINSCHEMA* und *LITERAL* vorkommen. Zusätzlich sind in der Tabelle 5.6 eine

Auflistung der drei am öftesten vorkommenden Klassen auf Ebene L2, Ebene L3 und auf Ebene L4 des *star_resourcetypes* sowie die drei am häufigsten auftretenden *resourcetypes, subjecttypes* und *objecttypes* je untersuchtem Format angeführt.

Tabelle 5.6: Vergleich der Verwendung von schema.org-Klassen in den Formaten Microdata, JSON-LD und RDFa

	Microdata	**JSON-LD**	**RDFa**
Anteil schema:Thing	19,6 %	29,8 %	0,1 %
Anteil NA_NORDFTYPE	40,8 %	60,0 %	76,5 %
Anteil NA_NOTINSCHEMA	9,0 %	0,3 %	21,9 %
Anteil LITERAL	30,6 %	10,0 %	1,7 %
Häufigste Klassen für Ressourcen	1. Intangible	1. CreativeWork	1. CreativeWork
	2. CreativeWork	2. Intangible	2. Article
	3. Product	3. ListItem	3. Blog
Häufigste Klassen auf L2	1. CreativeWork	1. CreativeWork	1. CreativeWork
	2. Intangible	2. Intangible	2. Person
	3. Organization	3. Organization	3. Organization
Häufigste Klassen auf L3	1. Product	1. ListItem	1. Article
	2. Offer	2. WebSite	2. Blog
	3. LocalBusiness	3. SearchAction	3. Person
Häufigste Klassen auf L4	1. Product	1. ListItem	1. Blog
	2. Offer	2. WebSite	2. SocialMediaPosting
	3. ListItem	3. SearchAction	3. NewsArticle
Häufigste Klassen für Subjekte	1. Product	1. Organization	1. BlogPosting
	2. ProgramMembership	2. WebSite	2. NewsArticle
	3. Offer	3. ListItem	3. CreativeWork
Häufigste Klassen für Objekte	1. Organization	1. ListItem	1. Person
	2. Product	2. SearchAction	2. NewsArticle
	3. Offer	3. ImageObject	3. CreativeWork

Aufgrund des Ergebnisses des Vergleiches der Formate Microdata, JSON-LD und RDFa bezüglich deren Verwendung von schema.org-Klassen werden im folgenden Abschnitt Beispiel-Abfragen auf die Cubes und Stars des Formates JSON-LD gezeigt.

5.2.5 Exkurs zu Duplicate-Blank-Node-Identifier in JSON-LD-Daten

Die ersten Auswertungen JSON-LD betreffend wurden bereits mit dem Web-Data-Commons-Korpus 2016-10 durchgeführt. Dabei wurde ein Problem der Datenqualität in den JSON-LD-Rohdaten des Korpus 2016-10 erkannt und

eine mögliche Lösung erarbeitet.

Nach den ersten Analysen der Daten mit den erstellten Cubes und Stars stellte sich heraus, dass ein Problem der Datenqualität in den verwendeten JSON-LD-Rohdaten besteht. Das Problem ist bereits nach der Filterung der vorkommenden RDF-Klassen in den Rohdaten ersichtlich. Die Selektion der bisher als Beispiel verwendeten Datenherkunft *https://www.allmodern.com/-Dining-Tables-C366120.html?redir_sku=JCS1337* zeigt in Tabelle 5.7, dass für diese Datenherkunft der Blank-Node *_:b0* mehrere, inhaltlich keinen Sinn ergebende RDF-Klassen aufweist. Der Abfrage zufolge ist der Blank-Node *_:b0* von der Klasse *schema:WebSite*, *schema:Product* und *schema:BreadcrumbList*.

Tabelle 5.7: Problem durch Duplicate-Blank-Node-Identifier in JSON-LD-Rohdaten

resource	**type**	**provenance**
_:b0	<http://schema.org/WebSite>	<https://www.allmodern.com/Dining-Tables-C366120...
_:b0	<http://schema.org/Product>	<https://www.allmodern.com/Dining-Tables-C366120...
_:b1	<http://schema.org/AggregateRating>	<https://www.allmodern.com/Dining-Tables-C366120...
_:b2	<http://schema.org/Offer>	<https://www.allmodern.com/Dining-Tables-C366120...
_:b0	<http://schema.org/BreadcrumbList>	<https://www.allmodern.com/Dining-Tables-C366120...
_:b1	<http://schema.org/AggregateRating>	<https://www.allmodern.com/Dining-Tables-C366120...
_:b2	<http://schema.org/Offer>	<https://www.allmodern.com/Dining-Tables-C366120...
_:b0	<http://schema.org/Product>	<https://www.allmodern.com/Dining-Tables-C366120...

Die Ermittlung der Ursache für dieses Problem hat ergeben, dass die Blank-Node-Identifier in JSON-LD-Rohdaten je Datenherkunft nicht eindeutig sind. Für die Verifizierung des Problems wurde eine weitere Datenherkunft, *https://www.theguardian.com/technology/series/dorktalk+books/douglasadams*, betrachtet. Pro Datenherkunft können mehrere JSON-LD-Skript-Tags vorkommen, die für sich einen eigenständigen RDF-Graphen darstellen. In der Abbildung 5.1 sind die beiden vorkommenden JSON-LD-Skript-Tags im HTML-Code der Datenherkunft *https://www.theguardian.com/technology/series/dorktalk+books/douglasadams* mit deren Übersetzung nach RDF ersichtlich.

```
<script data-schema="Organization" type="application/ld+json">{
  "name":"The Guardian",
  "url":"http://www.theguardian.com/",
  "logo":"https://assets.guim.co.uk/images/...png",
  "sameAs":["https://www.facebook.com/theguardian","https://twitter.com/guardian",..."],
  "@type":"Organization",
  "@context":"http://schema.org"
}</script>

_:b0 <http://schema.org/logo> <https://assets.guim.co.uk/images/...png> .
_:b0 <http://schema.org/name> "The Guardian" .
_:b0 <http://schema.org/sameAs> <https://plus.google.com/+theGuardian> .
_:b0 <http://schema.org/sameAs> <https://twitter.com/guardian> .
_:b0 <http://schema.org/sameAs> <https://www.facebook.com/theguardian> .
_:b0 <http://schema.org/sameAs> <https://www.youtube.com/user/TheGuardian> .
_:b0 <http://schema.org/url> <http://www.theguardian.com/> .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Organization> .


<script data-schema="WebPage" type="application/ld+json">{
  "@id":"https://www.theguardian.com/technology/series/dorktalk+books/douglasadams",
  "potentialAction":{"@type":"ViewAction","target":"android-app://com.guardian/../douglasadams"},
  "@type":"WebPage",
  "@context":"http://schema.org"}
</script>

<https://www.theguardian.com/technology/../douglasadams> <http://schema.org/potentialAction> _:b0 .
<https://www.theguardian.com/technology/../douglasadams> rdf:type <http://schema.org/WebPage> .
_:b0 <http://schema.org/target> "android-app://com.guardian/../technology/../douglasadams" .
_:b0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/ViewAction> .
```

Abbildung 5.1: JSON-LD eingebettet in HTML und entsprechende RDF-Tripel

In beiden Übersetzungen nach RDF ist zu erkennen, dass der Wert *_:b0* als Blank-Node-Identifier verwendet wird. Beide RDF-Graphen für sich betrachtet ermöglichen die Verwendung des gleichen Blank-Node-Identifiers, da die RDF-Graphen eigenständig sind und unterschiedliche Informationen enthalten, wie in den Abbildungen 5.2 und 5.3 zu sehen ist. Der Graph in 5.2 beschreibt eine Organisation mit dem Namen *"The Guardien"* und Abbildung 5.3 eine Webseite. Beide Graphen weisen unterschiedliche RDF-Klassen und -Attribute auf.

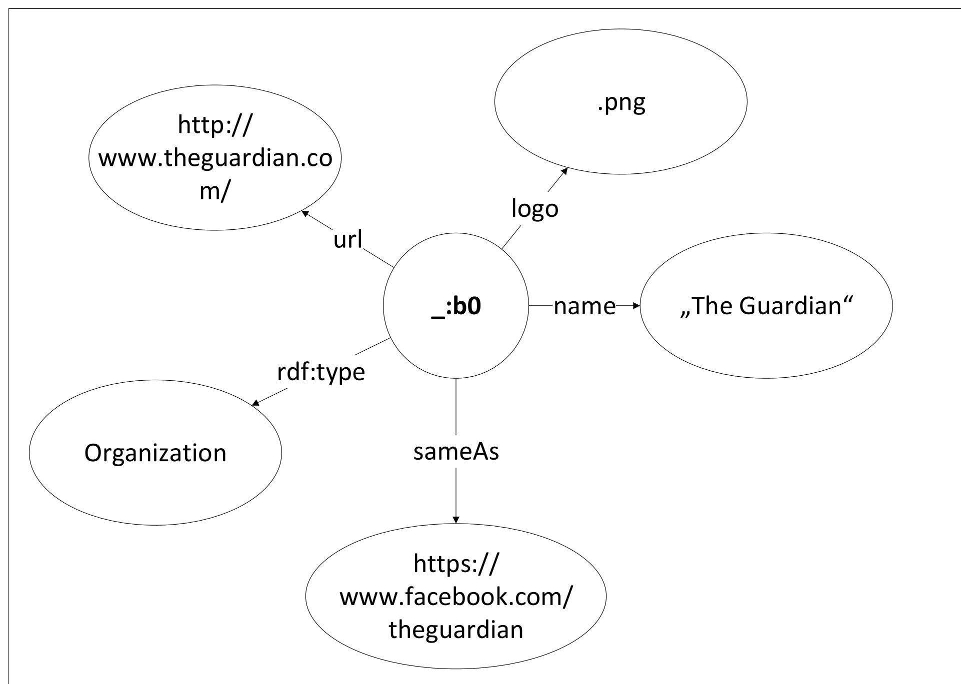


Abbildung 5.2: JSON-LD als RDF-Graph (Teil 1)

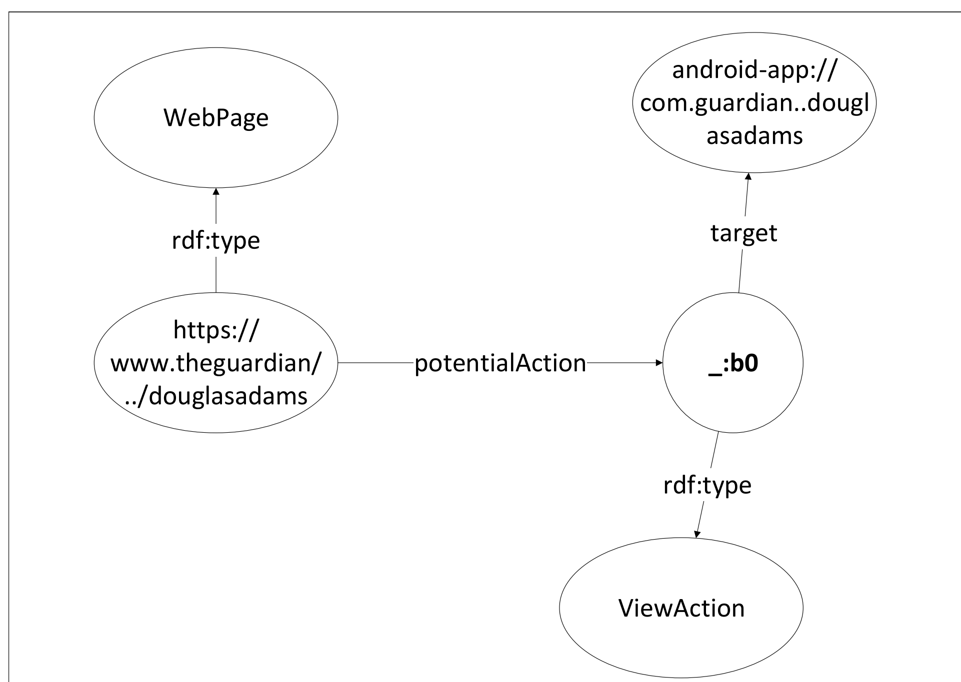


Abbildung 5.3: JSON-LD als RDF-Graph (Teil 2)

Das Problem der Datenqualität entsteht durch das Zusammenführen der Ergebnisse, nachdem die JSON-LD-Skripts getrennt voneinander nach RDF übersetzt wurden. Dadurch entsteht aus den beiden angeführten unabhängigen RDF-Graphen ein einziger Graph, der den Blank-Node-Identifier *_:b0* verwendet. Das Resultat des Zusammenführens ist in Darstellung 5.4 abgebildet. Die Vereinigung der beiden unabhängigen RDF-Graphen führt inhaltlich zu falschen Informationen in den JSON-LD-Rohdaten. Nach den Informationen, die in den JSON-LD-Rohdaten enthalten sind, ist der Blank-Node *_:b0* von der RDF-Klasse *schema:Organization* und *schema:ViewAction*, was inhaltlich nicht korrekt und auch auf der Webseite in dieser Form nicht zu finden ist.

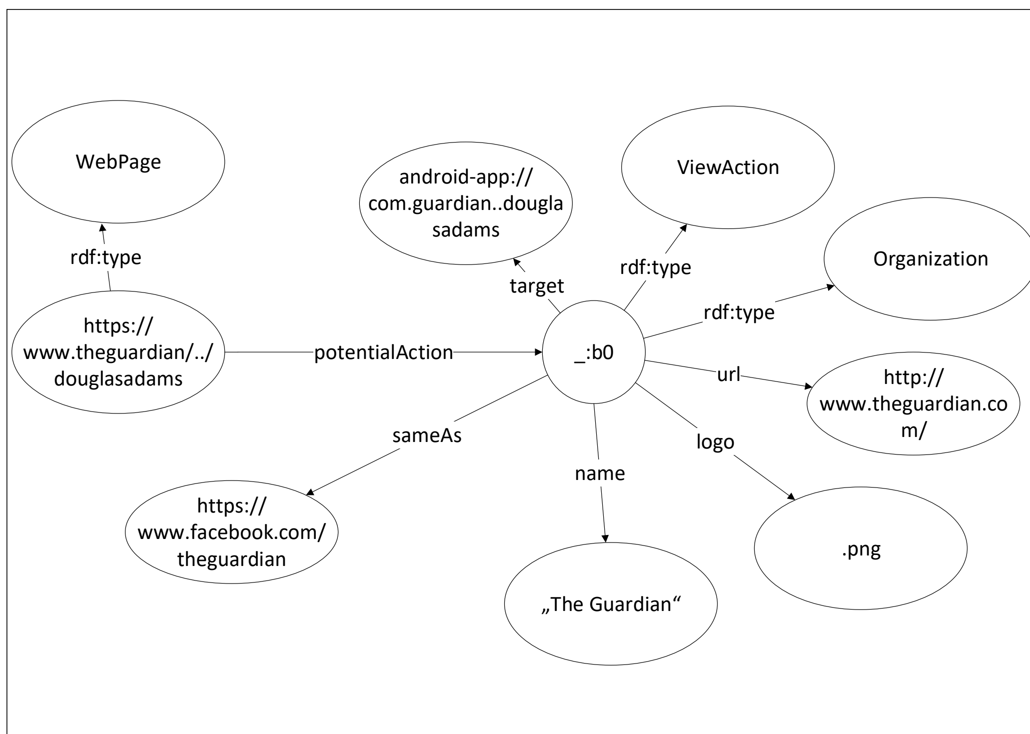


Abbildung 5.4: Visualisierung der Duplicate-Blank-Node-Identifier

Eine mögliche Lösung des Problems wäre, die Blank-Node-Identifier mit einer Referenz zum zugehörigen Skript-Tag zu ergänzen. Wie in Abbildung 5.5 erkennbar, wären durch die Ergänzung des Blank-Node-Identifiers diese für eine Datenherkunft eindeutig. Dadurch wären die vorkommenden Blank-Nodes der JSON-LD-Skript-Tags eindeutig, was in zwei anstatt einem RDF-Graphen resultiert. Dadurch wären auch die Rohdaten inhaltlich richtig für weitere Verarbeitungsschritte verfügbar.

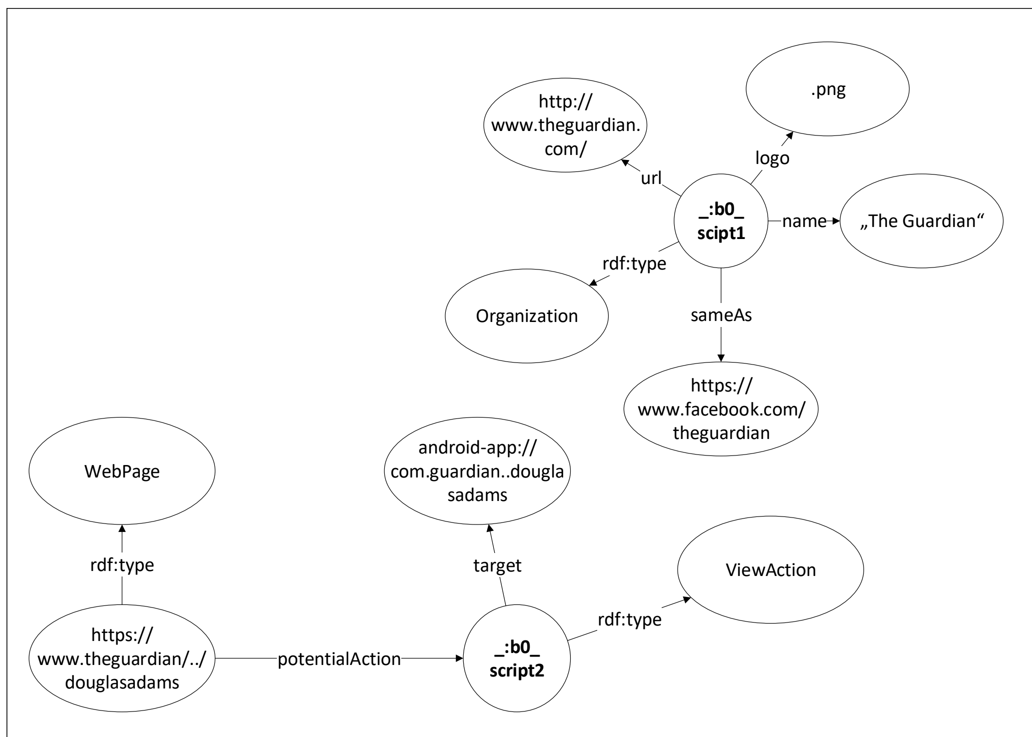


Abbildung 5.5: Lösungsvorschlag für die Duplicate-Blank-Node-Identifier

Um zu prüfen, ob die gleichen Probleme auch in anderen, von Web-Data-Commons extrahierten strukturierten Web-Daten vorkommen, wurde eine Microdata-Datei mit einer Stichprobe überprüft. Die Ausgabe der RDF-Klassen für eine zufällig gewählte Datenherkunft, *http://24.hu/elet-stilus/2011/04/21/ungvari_ aranyermes* zeigt, dass Blank-Nodes mit einem weiteren Identifier, wie die Stichprobe in Tabelle 5.8 veranschaulicht, verwendet werden. Auch die Anzahl der eindeutigen Werte des Feldes *resources* in der Tabelle *types* ist wesentlich höher (861606 von 868444 Rows), als das bei JSON-LD-Rohdaten der Fall ist (49066 von 1377182 Rows).

Tabelle 5.8: Stichprobe der Blank-Node-Identifier in einer Microdata-Datei

resource	**type**	**provenance**
_:node3953efbf2422666d9816c559f617bbf	<http://schema.org/Article>	<http://24.hu/elet-stilus/2011/...
_:nodeb855435c68da642f91be5693c476a30	<http://www.schema.org/SiteNavigationElement>	<http://24.hu/elet-stilus/2011/...
_:nodeda36508aebae3d79e6bb95262496c1	<http://www.schema.org/SiteNavigationElement>	<http://24.hu/elet-stilus/2011/...
_:nodec88f401ad2ae9dd886b4f83f55448f85	<http://schema.org/MobileApplication>	<http://24.hu/elet-stilus/2011/...
_:node1bceaaa64c5e5d1cb9fe46938d1c3d6	<http://schema.org/Offer>	<http://24.hu/elet-stilus/2011/...
_:node8fa285db52cb95cbf520738116ddc5ce	<http://schema.org/MobileApplication>	<http://24.hu/elet-stilus/2011/...
_:nodef14135731b2235377c563d4b86e485c3	<http://schema.org/Offer>	<http://24.hu/elet-stilus/2011/...
_:node8a10ed71374d50f2b24aea4d6356ee6d	<http://schema.org/MobileApplication>	<http://24.hu/elet-stilus/2011/...
_:nodea5e54a92803ce6a4b035cffd525d66d9	<http://schema.org/Offer>	<http://24.hu/elet-stilus/2011/...

Nach der neuerlichen Auswertung der JSON-LD-Rohdaten wegen des neuen Korpus 2017-12 konnte festgestellt werden, dass in den aktuellen Rohdaten-Dateien von JSON-LD andere, eindeutige Identifier für Blank-Nodes verwendet werden. Die im vorherigen Beispiel verwendete Datenherkunft *https://www. theguardian.com/technology/series/dorktalk+books/douglasadams* ist nicht in der verwendeten JSON-LD-Rohdaten-Datei des Korpus 2017-12 enthalten. Für die Stichprobe aus dem neuen Korpus, um die Blank-Node-Identifier zu prüfen, wurde die Datenherkunft <*https://www.theguardian.com-/politics/blog/live/2016/nov/21/theresa-may-cbi-speech-signals-plan-to-put--workers-on-company-boards-being-watered-down-politics-live?page=with:block-5832f4abe4b0e0ad2d75bc1f*>, ebenfalls von der Webseite *The Guardian*, ausgewählt. Die Suche nach dieser Datenherkunft im Data-Frame *df_statement* liefert die in Tabelle 5.9 aufgelisteten Einträge der Spalten *subject, predicate* und *object*. Für jeden RDF-Graphen dieser Datenherkunft wird ein eindeutiger Blank-Node-Identifier verwendet, was das zuvor beschriebene Problem der nicht eindeutigen Blank-Nodes löst.

Tabelle 5.9: Kontrolle der Blank-Node-Identifier des JSON-LD-Korpus 2017-12

subject	**predicate**	**object**
_:genid2d73a41bea7f6348818588ed3bde2a42672db0	rdf:type	<http://schema.org/Organization>
_:genid2d73a41bea7f6348818588ed3bde2a42672db0	<http://schema.org/logo>	<https://assets.guim.co.uk/images/../152x152.png>
_:genid2d73a41bea7f6348818588ed3bde2a42672db0	<http://schema.org/name>	The Guardian
_:genid2d73a41bea7f6348818588ed3bde2a42672db0	<http://schema.org/sameAs>	<https://www.facebook.com/theguardian>
_:genid2d73a41bea7f6348818588ed3bde2a42672db0	<http://schema.org/sameAs>	<https://twitter.com/guardian>
_:genid2d73a41bea7f6348818588ed3bde2a42672db0	<http://schema.org/sameAs>	<https://plus.google.com/+theGuardian>
_:genid2d73a41bea7f6348818588ed3bde2a42672db0	<http://schema.org/sameAs>	<https://www.youtube.com/user/TheGuardian>
_:genid2d73a41bea7f6348818588ed3bde2a42672db0	<http://schema.org/url>	<http://www.theguardian.com/>
_:genid2d00c6ba6200e3465091528123dfd900e52db0	rdf:type	<http://schema.org/ViewAction>
_:genid2d00c6ba6200e3465091528123dfd900e52db0	<http://schema.org/target>	android-app://com.guardian/https/..
_:genid2dbe9050be6edc4084b871a04e3dece89a2db0	rdf:type	<http://schema.org/ImageObject>
_:genid2dbe9050be6edc4084b871a04e3dece89a2db0	<http://schema.org/height>	"60"^^xsd:integer>
_:genid2dbe9050be6edc4084b871a04e3dece89a2db0	<http://schema.org/url>	<https://static.guim.co.uk/../Guardiantitle.png>
_:genid2dbe9050be6edc4084b871a04e3dece89a2db0	<http://schema.org/width>	"300"^^xsd:integer>
_:genid2d1be7e1600535414eb518d44c89ed33042db0	rdf:type	<http://schema.org/LiveBlogPosting>

5.3 Beispiel-Abfragen auf RDF-Summarization-Cubes

In den JSON-LD-Daten ist im Vergleich aus Abschnitt 5.2 die größte Gesamtzahl an Klassenausprägungen zu finden. Zudem kommen die Klasse <http://schema.org/Thing> und ihre Subklassen sehr oft vor, was einer häufigen Nutzung von schema.org-Klassen entspricht. Auffällig ist auch, dass die Klasse *NA_NOTINSCHEMA* nicht unter den Top 20 vorkommt.

Um neben der Anzahl der vorkommenden Klassen, was mit Hilfe des Cube *cube_resources* möglich ist, auch Erkenntnisse bezüglich der Anzahl verschiedener *subject, predicate* und *object* Klassenkombinationen zu erlangen, kann der Cube *cube_statements* verwendet werden. Dieser Cube erlaubt Abfragen auf bestimmte Objekt- und Subjektklassen oder, unter Einbeziehung der Tabelle der in Ebenen unterteilten Hierarchie in einer Unterabfrage, die Verwendung einer bestimmten Ebene der erweiterten schema.org-Hierarchie. Listing 5.4 zeigt eine Abfrage mit deren Ergebnis in Tabelle 5.10, die auf die Subjektklasse *ANY* und die Ebene L2 der Hierarchie als Objektklasse abfragt.

Listing 5.4: Abfrage 1 des Statement-Cube

```
1 SELECT *
2 FROM cube_statements
3 WHERE subjectType = 'ANY'
4 AND objectType in (SELECT L2
5                    FROM hierarchy_leveled)
```

Tabelle 5.10: Ergebnis Abfrage 1 des Statement-Cube

subjecttype	**predicate**	**objecttype**	**nr**
ANY	<http://schema.org/name>	NA_NORDFTYPE	739478
ANY	<http://schema.org/url>	NA_NORDFTYPE	513108
ANY	<http://schema.org/sameAs>	NA_NORDFTYPE	345343
ANY	<http://schema.org/itemListElement>	<http://schema.org/Thing>	206301
ANY	<http://schema.org/itemListElement>	<http://schema.org/Intangible>	205113
ANY	<http://schema.org/position>	LITERAL	195634
ANY	<http://schema.org/item>	NA_NORDFTYPE	175484
ANY	<http://schema.org/target>	NA_NORDFTYPE	166242

Eine weitere Möglichkeit des Cube *cube_statement* besteht in der Abfrage eines bestimmten Teilbaumes der Hierarchie wie in Listing 5.5 gezeigt. Das Ergebnis der Abfrage in Tabelle 5.11 enthält Einträge, die als Subjektklasse

eine Klasse der Ebene L3 der Hierarchie haben und bei denen die Objektklasse der Ebene L3 die Klasse *<http://schema.org/Intangible>* als Superklasse haben.

Listing 5.5: Abfrage 2 des Statement-Cube

```
1 SELECT *
2 FROM cube_statements
3 WHERE subjectType in (SELECT L3
4                       FROM hierarchy_leveled)
5 AND   objectType  in (SELECT L3
6                       FROM hierarchy_leveled
7                       WHERE L2=
8                        '<http://schema.org/Intangible>')
```

Tabelle 5.11: Ergebnis Abfrage 2 des Statement-Cube

subjecttype	**predicate**	**objecttype**	**nr**
schema:Thing	schema:itemListElement	schema:Intangible	204897
schema:Intangible	schema:itemListElement	schema:Intangible	203775
schema:ItemList	schema:itemListElement	schema:Intangible	203775
schema:Thing	schema:itemListElement	schema:ListItem	200339
schema:ItemList	schema:itemListElement	schema:ListItem	199217
schema:Intangible	schema:itemListElement	schema:ListItem	199217
schema:Thing	schema:offers	schema:Offer	28039
schema:Thing	schema:offers	schema:Intangible	28039

Informationen über die Vorkommen von Klassen im Verhältnis zu einer anderen Klasse können auch mit Hilfe des Cube *cube_statements* gewonnen werden. Die Ausführung der Abfrage in Listing 5.6 liefert die Klasse *<http://schema.org/Thing>* als 100% und im Verhältnis dazu die Vorkommen der anderen Klassen, zu sehen in Tabelle 5.12

Listing 5.6: Abfrage 3 des Statement-Cube

```
1 SELECT
2 *,
3 (nr/(SELECT nr
4      FROM cube_resources
5      WHERE resourcetype=
6       '<http://schema.org/Thing>') * 100) AS ratio
7 FROM cube_resources
8 ORDER BY ratio DESC
```

Tabelle 5.12: Ergebnis Abfrage 3 des Statement-Cube

resourcetype	**nr**	**ratio**
<http://schema.org/Thing>	1333384	100
<http://schema.org/CreativeWork>	439519	32.964622344350914
<http://schema.org/Intangible>	393055	29.46450534879674
<http://schema.org/ListItem>	200864	15.064227559352744
<http://schema.org/WebSite>	190912	14.317855921474983
<http://schema.org/Organization>	172321	12.923583903811656
<http://schema.org/Action>	167872	12.589921582979846
<http://schema.org/SearchAction>	161912	12.142938568334403

Analysen, die sich auf eine bestimmte Datenherkunft beziehen oder bei welchen die Datenherkunft Teil des Ergebnisses sein soll, sind mit den gezeigten Cubes nicht möglich, da die Datenherkunft in keinem der beiden Cubes enthalten ist. Für Untersuchungen dieser Art können die beiden Stars herangezogen werden. Durch Abfragen des Star *star_resourcetypes* können Analysen über die verwendeten Klassen von Ressourcen unter der Einbeziehung der in Ebenen unterteilten erweiterten schema.org-Hierarchie durchgeführt werden. Die Anzahl der verwendeten Klassen, die Subklassen von *<http://schema.org/CreativeWork>* sind, kann in Tabelle 5.13 nachgelesen werden, die durch die Abfrage in Listing 5.7 erstellt wurde.

Listing 5.7: Abfrage des Ressource-Star

```
1 SELECT L3, COUNT(DISTINCT(resource,provenance)) AS nr
2 FROM star_resourcetypes
3 WHERE L2 = '<http://schema.org/CreativeWork>'
4 GROUP BY L3
5 ORDER BY nr DESC
```

Tabelle 5.13: Ergebnis einer Abfrage des Ressource-Star

L3	**nr**
<http://schema.org/WebSite>	190912
<http://schema.org/MediaObject>	109431
<http://schema.org/Article>	72780
<http://schema.org/WebPage>	40747
<http://schema.org/MusicRecording>	6155
<http://schema.org/Comment>	5002
<http://schema.org/Review>	2851
<http://schema.org/MusicPlaylist>	2479

Listing 5.8 zeigt eine Beispielabfrage, bei der die Datenherkunft einbezogen wird. Dabei werden wieder jene Klassen der Ebene L3 im *star_resourcetypes* abgefragt, die eine Subklassen von *<http://schema.org/CreativeWork>* sind. Um die Anzahl dieser Klassen je Datenherkunft abzufragen, wird zusätzlich zu L3 auch die Datenherkunft gruppiert. Das Abfrageergebnis, auszugsweise in Tabelle 5.14 zu sehen, zeigt jene Klassen, die öfter als 10 mal je Datenherkunft vorkommen.

Listing 5.8: Abfrage des Ressource-Star mit Datenherkunft

```
1 SELECT provenance, L3, COUNT(DISTINCT(resource,
     provenance)) AS nr
2 FROM star_resourcetypes
3 WHERE L2 = '<http://schema.org/CreativeWork>'
4 GROUP BY L3, provenance
5 HAVING nr > 10
6 ORDER BY provenance
```

Tabelle 5.14: Ergebnis einer Abfrage des Ressource-Star mit Datenherkunft

provenance	**L3**	**nr**
<http://advantagemedicalprofessionals.com/about>	<http://schema.org/MediaObject>	27
<http://aolradio.slacker.com/artist/breaking%20benjamin>	<http://schema.org/MusicRecording>	69
<http://aolradio.slacker.com/artist/breaking%20benjamin>	<http://schema.org/MusicPlaylist>	87
<http://aolradio.slacker.com/artist/buju+banton>	<http://schema.org/MusicPlaylist>	251
<http://aolradio.slacker.com/artist/buju+banton>	<http://schema.org/MusicRecording>	200
<http://aolradio.slacker.com/artist/cat+power>	<http://schema.org/MusicRecording>	104
<http://aolradio.slacker.com/artist/cat+power>	<http://schema.org/MusicPlaylist>	119
<http://aolradio.slacker.com/artist/kid%20frost>	<http://schema.org/MusicRecording>	29

Die umfangreichsten Analysen erlaubt jedoch der Star *star_statementtypes*, der die Rohdaten erweitert um *subjecttype* und *objecttype* mit deren Hierarchie enthält. Dieser Star lässt Abfragen auf unterschiedlichen Ebenen der *subjecttypes* und *objecttypes*, sowie die Auswertung von bestimmten Werten des Feldes *provenances* zu. Eine solche Abfrage ist in Listing 5.9 zu sehen, die das Subjekt auf Ebene L2 (*sL2*) und das Objekt auf Ebene L1 (*oL1*) aufrollt, eingeschränkt auf Webseiten (*provenance*), deren URLs die Zeichenkette *//www.theguardian.com* enthalten.

Listing 5.9: Abfrage des Statement-Star

```
1 SELECT
2 sL2,
3 predicate,
```

```
4 oL1,
5 COUNT(DISTINCT(subject,predicate,object,provenance)) AS
    nr
6 FROM star_statementtypes
7 WHERE provenance like '%//www.theguardian.com%'
8 GROUP BY sL2, predicate, oL1
9 ORDER BY nr DESC
```

Tabelle 5.15: Ergebnis einer Abfrage des Statement-Star

sL2	**predicate**	**oL1**	**nr**
schema:Organization	schema:sameAs	NA_NORDFTYPE	1541
schema:Organization	schema:name	NA_NORDFTYPE	387
schema:Organization	schema:logo	NA_NORDFTYPE	385
schema:Organization	schema:url	NA_NORDFTYPE	385
schema:CreativeWork	schema:potentialAction	schema:Thing	384
schema:Action	schema:target	NA_NORDFTYPE	384
schema:CreativeWork	schema:url	NA_NORDFTYPE	12
schema:CreativeWork	schema:datePublished	LITERAL	12

5.4 Skalierbarkeit - Umsetzung in der Cloud mit Microsoft Azure und Databricks

Da die PoC-Umgebung nicht in der Lage ist, größere Datenmengen zu verarbeiten, und bisher jeweils nur eine Datei von Web-Data-Commons [9] verarbeitet wurde, wurde nach einer Lösung gesucht, um zu zeigen, dass die umgesetzten Skripten skaliert und auf größere Datenmengen angewendet werden können. Für die Ausführung der Skripten auf mehreren Rohdaten-Dateien bot sich Microsoft Azure mit seinen zahlreichen Services im Bereich Big Data und Analytics an. Da die Umsetzung des in Kapitel 3 gezeigten Entwurfes in Spark erfolgte, wurde die in Azure angebotene Plattform Databricks für die Ausführung der Skripten gewählt. Databricks bietet die Möglichkeit, Spark-Skripte über eine Schnittstelle, einem sogenannten Notebook, in einem Spark-Cluster, der durch Benutzer konfiguriert werden kann, auszuführen. Um Databricks in Microsoft Azure nutzen zu können, wurde eine Pay-As-You-Go-Subscription verwendet, bei der je nach Verwendung von Ressourcen (verwendete Komponenten, Speicher, CPU, und Datenmenge) abgerechnet wird. Damit die Rohdaten von den Skripten gelesen werden können, wurde ein Azure-Blob-Store angelegt, in dem zehn JSON-LD-Rohdatendateien in

gzip-komprimierter Form und die schema.org-Hierarchie in einer Datei abgelegt wurden, so wie in Abbildung 5.6 gezeigt wird.

Hochladen Aktualisieren Löschen Lease abrufen Lease abbrechen ··· Mehr

Speicherort: containermaster

Blobs nach Präfix durchsuchen (Beachtung von Groß-/Kleinschreibung) Gelöschte Blobs anzeigen

NAME	GEÄNDERT	BLOB-TYP	GRÖSSE	LEASEZU...	
dpef.html-embedded-jsonld.nq-00000.gz	3.6.2018, 9:49:40 vorm.	Blockblob	100.08 M...	Verfügbar	...
dpef.html-embedded-jsonld.nq-00001.gz	3.6.2018, 12:07:40 nac...	Blockblob	100.7 MiB	Verfügbar	...
dpef.html-embedded-jsonld.nq-00002.gz	3.6.2018, 12:07:27 nac...	Blockblob	100.35 M...	Verfügbar	...
dpef.html-embedded-jsonld.nq-00003.gz	3.6.2018, 12:07:21 nac...	Blockblob	100.23 M...	Verfügbar	...
dpef.html-embedded-jsonld.nq-00004.gz	3.6.2018, 12:07:35 nac...	Blockblob	100.39 M...	Verfügbar	...
dpef.html-embedded-jsonld.nq-00005.gz	3.6.2018, 12:07:41 nac...	Blockblob	100.65 M...	Verfügbar	...
dpef.html-embedded-jsonld.nq-00006.gz	3.6.2018, 12:07:24 nac...	Blockblob	100.14 M...	Verfügbar	...
dpef.html-embedded-jsonld.nq-00007.gz	3.6.2018, 12:07:35 nac...	Blockblob	100.65 M...	Verfügbar	...
dpef.html-embedded-jsonld.nq-00008.gz	3.6.2018, 12:07:24 nac...	Blockblob	100.61 M...	Verfügbar	...
dpef.html-embedded-jsonld.nq-00009.gz	3.6.2018, 12:07:35 nac...	Blockblob	100.47 M...	Verfügbar	...
schema.nt	3.6.2018, 10:51:16 vorm.	Blockblob	1.01 MiB	Verfügbar	...

Abbildung 5.6: Screenshot - Persistierung der Rohdaten und der schema.org-Hierarchie im Azure-Blob-Store

Anschließend wurde die Plattform Databricks, in Abbildung 5.7 dargestellt, mit den Konfigurationen in Abbildung 5.8 erstellt.

Nach der Erstellung und Konfiguration von Databricks kann die in Abbildung 5.9 abgebildete graphische Benutzerschnittstelle verwendet werden. Diese Schnittstelle ermöglicht die Durchführung weiterer Konfigurationen und die Anlage eines Spark-Clusters für die Ausführung von den erstellten Skripten.

Bevor die Skripten im Databricks-Notebook ausgeführt werden können, muss der Spark-Cluster in Abbildung 5.10 eingerichtet werden.

Für die Server des Spark-Clusters wurde die Kategorie mit den wenigsten Ressourcen (Speicher und CPU) *Standard_ DS3_ v2* Server mit 14 GB RAM und 4 Cores sowohl für den Server, auf dem der Spark-Treiber ausgeführt wird, als auch für die Worker-Knoten, auf denen die Skripten ausgeführt werden, verwendet. Die Autoscaling-Konfiguration startet den Cluster mit

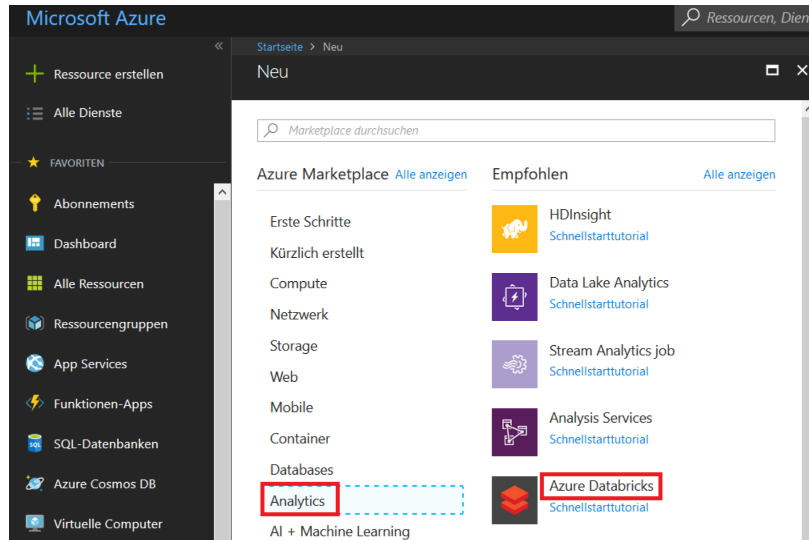


Abbildung 5.7: Screenshot - Erstellung der Databricks-Umgebung

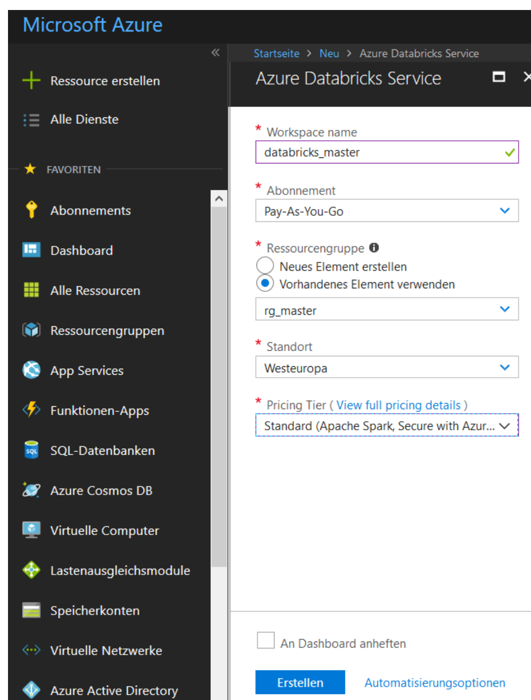


Abbildung 5.8: Screenshot - Databricks-Konfiguration

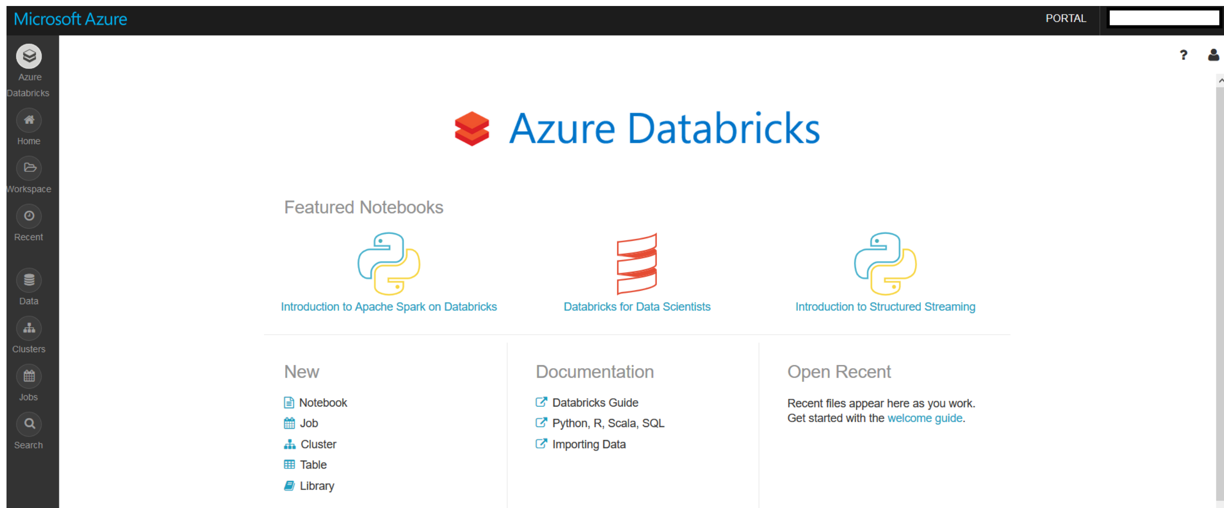


Abbildung 5.9: Screenshot - Graphische Databricks-Benutzerschnittstelle

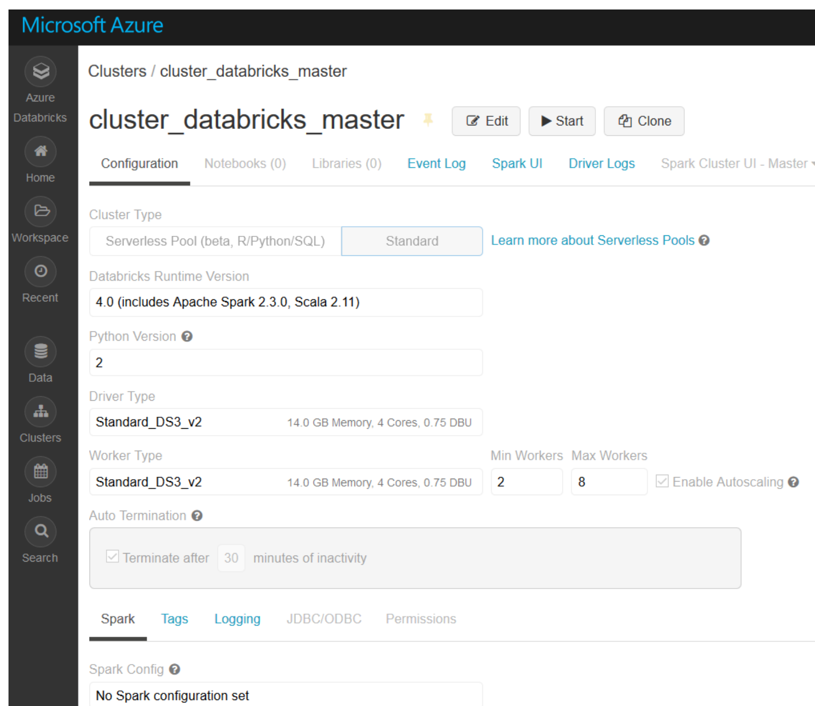


Abbildung 5.10: Screenshot - Databricks-Spark-Cluster

zwei Knoten und skaliert bei Bedarf auf bis zu acht Knoten. Die Server-Ressourcen sind etwas geringer als die in der PoC-Umgebung. Die Anzahl der verfügbaren Server ist bei Bedarf jedoch doppelt so hoch. Außerdem kann der im Cluster verwendete Server-Typ jederzeit auf Server mit mehr Ressourcen geändert werden.

Nach dem Cluster-Start wurden die in der Umsetzung in Kapitel 4 erstellten Skripten in das in Abbildung 5.11 von Databricks zur Verfügung gestellte Notebook migriert. Im Unterschied zur Umsetzung im PoC-Cluster finden sich in Databricks alle vier Skripten in einem Notebook und es werden keine Zwischenergebnisse persistiert, da das für den Test der Ausführbarkeit auf mehrere Rohdatendateien nicht notwendig ist. Eine weitere Änderung war für das Einlesen der Rohdatendateien notwendig. Listing 5.10 zeigt die Konfiguration des Azure-Blob-Stores und den geänderten Pfad zu den Rohdatendateien.

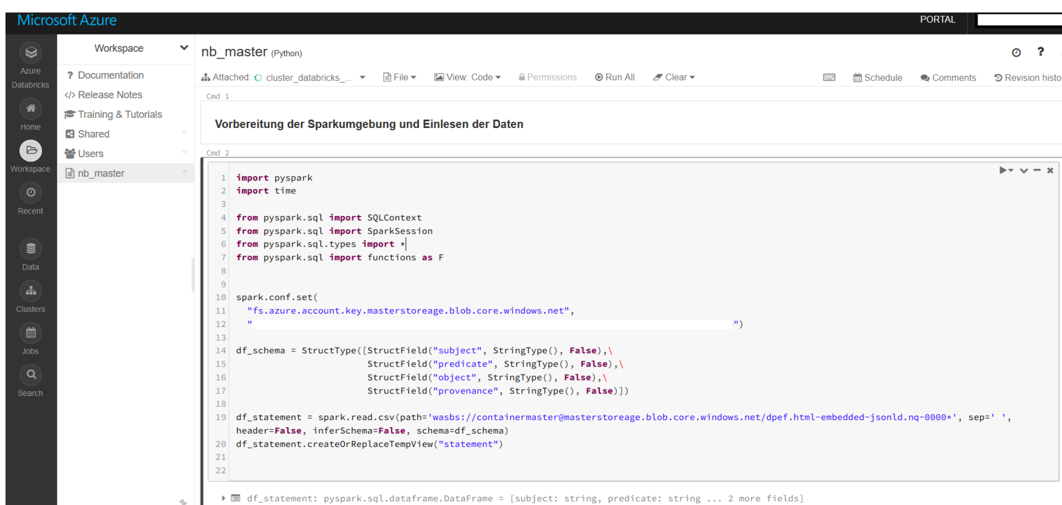


Abbildung 5.11: Screenshot - Databricks-Notebook

Listing 5.10: Konfiguration und Verwendung des Azure-Blob-Stores

```
1 spark.conf.set(
2 "fs.azure.account.key
3 .masterstoreage.blob.core.windows.net",
4 "ACCESS_KEY")
5
6 df_statement =
7 spark.read.csv(
8 path='wasbs://containermaster@masterstoreage
9 .blob.core.windows.net
```

```
10  /dpef.html-embedded-jsonld.nq-0000*',
11  sep=' ',
12  header=False,
13  inferSchema=False,
14  schema=df_schema)
```

Für den Test, ob die Skripten auch auf mehreren Rohdatendateien ausgeführt werden können, wurden anstatt einer zehn JSON-LD-Rohdatendateien verarbeitet. Auch mit der geringsten Server-Kategorie konnten die Skripten erfolgreich ausgeführt werden. Das Data-Frame *df_statement* enthält nach dem Einlesen aller zehn Rohdatendateien, wie in den Abbildungen 5.14 und 5.13 zu sehen ist, 60054812 Einträge im Vergleich zu einer Rohdatendatei mit 5992471 Einträgen.

Abbildung 5.12: Screenshot - Anzahl von Einträgen in einer Rohdatendatei

Abbildung 5.13: Screenshot - Anzahl von Einträgen in zehn Rohdatendateien

Auch der Vergleich in den Abbildungen 5.14 und 5.15 der beiden Cubes *cube_resources* zeigt, dass die Anzahl der jeweiligen *resourcetypes* wesentlich höher ist als zuvor.

Der Test hat gezeigt, dass der umgesetzte Entwurf in Spark auch mit einer größere Datenmenge, als in der PoC-Umgebung verwendet, umgehen kann. Je nach zu verarbeitender Datenmenge muss der Cluster entsprechend dimensioniert werden. Die Verwendung von Databricks in Microsoft Azure

```
1 %sql
2 select * from cube_resources
```

(5) Spark Jobs

resourcetype	nr
ANY	4499037
NA_NORDFTYPE	2716991
<http://schema.org/Thing>	1333384
LITERAL	448662
<http://schema.org/CreativeWork>	439545
<http://schema.org/Intangible>	392875
<http://schema.org/ListItem>	200864
<http://schema.org/WebSite>	190912
<http://schema.org/Organization>	172321

Command took 2.56 seconds -- by rbuschberger@gmx.at at 3.6.2018, 19:37:36 on cluster_databricks_master

Abbildung 5.14: Screenshot - Anzahl von Einträgen in *cube_resources* mit einer Rohdatendatei

```
1 %sql
2 select * from cube_resources
```

(5) Spark Jobs

resourcetype	nr
ANY	45087458
NA_NORDFTYPE	27239815
<http://schema.org/Thing>	13360065
LITERAL	4487578
<http://schema.org/CreativeWork>	4375080
<http://schema.org/Intangible>	3887529
<http://schema.org/ListItem>	1991130
<http://schema.org/WebSite>	1919526
<http://schema.org/Organization>	1717218

Command took 2.30 seconds -- by rbuschberger@gmx.at at 10.6.2018, 16:28:29 on databricks_cluster_master2

Abbildung 5.15: Screenshot - Anzahl von Einträgen in *cube_resources* mit zehn Rohdatendateien

bietet eine gute Möglichkeit, den Spark-Cluster mit Hilfe von Scale-Up und Scale-Out zu skalieren.

5.5 Zusammenfassung

Da in der Rohdatendatei des Formates JSON-LD im Verhältnis am meisten Ressourcen mit schema.org-Klassen vorkommen und die Untersuchung der Verwendung der schema.org-Klassen im Fokus dieser Arbeit liegt, wurden auf Basis der JSON-LD-Daten Beispiele für die Verwendung der erstellten Cubes und Stars gezeigt. Dabei wurden unterschiedliche Abfragen angeführt, mit denen Analysen der verwendeten Klassen der erweiterten schema.org-Hierarchie und die Beziehungen zwischen Subjekt- und Objektklassen möglich sind. Zudem können Subjekt- und Objektklassen auch unabhängig voneinander auf unterschiedlichen Ebenen der erweiterten schema.org-Hierarchie betrachtet und bestimmte Datenherkünfte untersucht werden.

Wegen der begrenzten Datenmenge, die in der PoC-Umgebung verarbeitet werden kann, wurde mit der Ausführung der Skripten in Databricks, einer in Microsoft Azure verfügbaren Plattform, gezeigt, dass eine skalierbare Ausführung des Spark-Codes möglich ist und damit auch größere Datenmengen verarbeitet werden können.

Kapitel 6

Fazit

6.1 Zusammenfassung und Schlussfolgerung

Bevor Analysen auf Datenbestände einzelner oder auch verschiedener miteinander zu integrierenden Datenquellen durchgeführt werden können, ist die Kenntnis über deren Beschaffenheit Voraussetzung. Die Aktivitäten zur Erhebung der Datenbeschaffenheit, zu der unter anderem die Struktur, die verwendeten Datentypen und verschiedene Muster von Werten zu zählen sind, werden unter dem Begriff Data-Profiling zusammengefasst und sind bereits aus traditionellen Datenbanksystemen bekannt.

Die Anforderung, auch semistrukturierte Daten zu analysieren, welche durch die rasanten Entwicklungen im Bereich Informations- und Kommunikationstechnologien und der damit verbundenen Vernetzung von mobilen Geräten in einem hohen Ausmaß erzeugt werden, bringt neue Herausforderungen für Datenanalysen und Data-Profiling mit sich. Neben den umfangreichen semistrukturierten Datenmengen stellt die Bedeutung von im Web veröffentlichten Inhalten und Seiten in maschinenlesbarer Form für automatisierte Auswertungen bereitzustellen, eine große Herausforderung dar. Für die Beschreibung der Bedeutung einer Webseite wurden unterschiedliche strukturierte Web-Daten-Formate wie Microformats, Microdata, JSON-LD oder RDFa entwickelt. Zudem ist ein einheitlich verwendetes Vokabular notwendig, um Inhalte vergleichen zu können. Da anfänglich strukturierte Web-Daten falsch oder nicht verwendet wurden, riefen die Suchmaschinenbetreiber Bing, Google, Yahoo! und Yandex das Projekt schema.org ins Leben, das ein einheitliches, einfach zu verwendendes Vokabular für die Beschreibung von Inhalten zur Verfügung stellt.

Um ein schema.org-bezogenes Data-Profiling zu ermöglichen und einen besseren Einblick in die Verwendung des schema.org-Vokabulars zu erhal-

ten, wurden im Zuge dieser Arbeit Cubes und Stars (zusammengefasst als RDF-Summarization-Cubes) erstellt, mit denen die Häufigkeit der Verwendung der im schema.org-Vokabular definierten Klassen und Properties abgefragt werden kann. Die von Web-Data-Commons extrahierten strukturierten Web-Daten wurden mit der schema.org-Hierarchie zu Ressource- und Statement-Fakten verknüpft, um darauf aufbauend die Cubes und Stars für die eigentliche Analyse der Verwendung des schema.org-Vokabulars zu erzeugen.

Da die Rohdaten in großem Umfang zur Verfügung stehen, ist die Skalierbarkeit der hier gezeigten Umsetzung ein wichtiger Aspekt, um auch umfassende Auswertungen durchführen zu können. Die Skalierbarkeit der implementierten Skripten wurde unter Zuhilfenahme von Microsoft Azure und der in Azure verfügbaren Plattform Databricks getestet. Das Ergebnis zeigt die Skalierbarkeit und, dass die Ausführung der Skripten auch auf größere Datenmengen als in der PoC-Umgebung verwendet, erfolgreiche Resultate liefert. Zu beachten ist, dass die Größe des Spark-Clusters der zu verarbeitenden Datenmenge entsprechen muss, um die Skripten erfolgreich ausführen zu können.

6.2 Ausblick

Die Aufgabenstellung dieser Arbeit bezog sich auf die Erstellung und Umsetzung eines Entwurfs für die Analyse der Verwendung des schema.org-Vokabulars. In weiterer Folge können Analysen auf umfangreichere Datenmengen durchgeführt werden, um bessere Erkenntnisse über die Verwendung von schema.org-Klassen zu erlangen. Zudem können Schnittstellen definiert werden, damit der Zugriff auf die Ressource- und Statement-Fakten sowie die Cubes und Stars auch aus anderen Systemen ermöglicht wird. Da die Visualisierung im Bereich Data-Profiling nicht zu vernachlässigen ist, können Visualisierungsmöglichkeiten für die Abfrageergebnisse evaluiert oder implementiert werden. Aufgrund der laufenden Weiterentwicklung des schema.org-Projektes sollte das schema.org-Vokabular in generischer Form in die Skripten eingebunden werden, um bei Abfragen die aktuelle Version des schema.org-Vokabulars verwenden zu können. Die Umsetzung der angeführten, noch offenen Implementierungen und die Verwendung des gezeigten Entwurfes erlauben, umfangreiche, in entsprechender Form visualisierte Analysen um Erkenntnisse über die Verwendung des schema.org-Vokabulars zu erlangen und dessen Weiterentwicklung zu unterstützen.

Literaturverzeichnis

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. “Profiling relational data: a survey”. In: *VLDB Journal — The International Journal on Very Large Data Bases* 24.4 (Aug. 2015), pp. 557–581. ISSN: 1066-8888. DOI: 10.1007/s00778-015-0389-y.

[2] Theodore Johnson. “Data Profiling”. In: *Encyclopedia of Database Systems*. 2009, pp. 604–608. DOI: 10.1007/978-0-387-39940-9_601. URL: https://doi.org/10.1007/978-0-387-39940-9_601.

[3] C. Böhm, F. Naumann, Z. Abedjan, et al. “Profiling linked open data with ProLOD”. In: *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)* (Mar. 2010), pp. 175–178. DOI: 10.1109/ICDEW.2010.5452762.

[4] Jonas Freiknecht. *Big Data in der Praxis: Lösungen mit Hadoop, HBase und Hive. Daten speichern, aufbereiten, visualisieren*. Carl Hanser Verlag GmbH & Co. KG, Oct. 2014. ISBN: 978-344643959-7.

[5] Douglas Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Tech. rep. META Group, Feb. 2001. URL: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[6] Christian Bizer, Tom Heath, and Tim Berners-Lee. “Linked Data - The Story So Far”. In: *Int. J. Semantic Web Inf. Syst.* 5 (2009), pp. 1–22.

[7] *Introduction to Structured Data | Search*. [Online; accessed 23. Apr. 2018]. Sept. 2017. URL: https://developers.google.com/search/docs/guides/intro-structured-data.

[8] Robert Meusel, Petar Petrovski, and Christian Bizer. “The WebDataCommons Microdata, RDFa and Microformat Dataset Series”. In: *SpringerLink* (Oct. 2014), pp. 277–292. DOI: 10.1007/978-3-319-11964-9_18.

[9] *Web Data Commons.* [Online; accessed 19. Apr. 2018]. Jan. 2018. URL: http://webdatacommons.org.

[10] R. V. Guha, Dan Brickley, and Steve MacBeth. “Schema.org: Evolution of Structured Data on the Web”. In: *Queue* 13.9 (Nov. 2015), pp. 10–37. ISSN: 1542-7730. DOI: 10.1145/2857274.2857276.

[11] Bernd Neumayr, Christoph G. Schuetz, and Michael Schrefl. “Towards Ontology-Driven RDF Analytics”. In: *SpringerLink* (Oct. 2015), pp. 210–219. DOI: 10.1007/978-3-319-25747-1_21.

[12] *Common Crawl.* [Online; accessed 19. Apr. 2018]. Apr. 2018. URL: http://commoncrawl.org.

[13] Stefan Anderlik, Bernd Neumayr, and Michael Schrefl. “Using Domain Ontologies as Semantic Dimensions in Data Warehouses”. In: *SpringerLink* (Oct. 2012), pp. 88–101. DOI: 10.1007/978-3-642-34002-4_7.

[14] Matei Zaharia, Reynold S. Xin, Patrick Wendell, et al. “Apache Spark: a unified engine for big data processing”. In: *Commun. ACM* 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782. DOI: 10.1145/2934664.

[15] Michael Armbrust, Reynold S. Xin, Cheng Lian, et al. *Spark SQL: Relational Data Processing in Spark.* ACM, May 2015. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2742797.

[16] Felix Naumann. “Data profiling revisited”. In: *SIGMOD Rec.* 42.4 (Feb. 2014), pp. 40–49. ISSN: 0163-5808. DOI: 10.1145/2590989.2590995.

[17] Ziawasch Abedjan and Felix Naumann. “Synonym Analysis for Predicate Expansion”. In: *SpringerLink* (May 2013), pp. 140–154. DOI: 10.1007/978-3-642-38288-8_10.

[18] Z. Abedjan, T. Gruetze, A. Jentzsch, et al. “Profiling and mining RDF data with ProLOD++”. In: *2014 IEEE 30th International Conference on Data Engineering* (2014), pp. 1198–1201. ISSN: 1063-6382. DOI: 10.1109/ICDE.2014.6816740.

[19] *Home - schema.org.* [Online; accessed 23. Apr. 2018]. Apr. 2018. URL: http://schema.org.

[20] *JSON-LD - JSON for Linking Data.* [Online; accessed 23. Apr. 2018]. Dec. 2017. URL: https://json-ld.org.

[21] The Apache Software Foundation. *Apache Any23 – Apache Any23 - Introduction.* [Online; accessed 1. Jun. 2018]. Mar. 2018. URL: https://any23.apache.org.

[22] Heiko Paulheim. "What the Adoption of schema.org Tells About Linked Open Data". In: *Joint Proceedings of the 5th International Workshop on Using the Web in the Age of Data (USEWOD '15) and the 2nd International Workshop on Dataset PROFIling and fEderated Search for Linked Data (PROFILES '15) co-located with the 12th European Semantic Web Conference (ESWC 2015), Portorož, Slovenia, May 31 - June 1, 2015.* 2015, pp. 85–90. URL: http://ceur-ws.org/Vol-1362/PROFILES2015_paper6.pdf.

[23] Kiumars Farkisch. *Data-Warehouse-Systeme kompakt: Aufbau, Architektur, Grundfunktionen (Xpert.press)*. Springer, July 2011. ISBN: 978-364221532-2.

[24] Hsinchun Chen, Roger H. L. Chiang, and Veda C. Storey. "Business intelligence and analytics: from big data to big impact". In: *MIS Q.* 36.4 (Dec. 2012), pp. 1165–1188. ISSN: 0276-7783. URL: http://dl.acm.org/citation.cfm?id=2481674.2481683.

[25] Jasmine Zakir, Tom Seymour, and Kristi Berg. "Big Data Analytics". In: *Issues in Information Systems* (July 2015), pp. 81–90.

[26] Michael Cox and David Ellsworth. "Application-controlled demand paging for out-of-core visualization". In: IEEE Computer Society Press, Oct. 1997. ISBN: 978-1-58113-011-9. URL: http://dl.acm.org/citation.cfm?id=266989.267068.

[27] Gil Press. *A Very Short History of Big Data*. [Online; accessed 9. May 2018]. June 2012. URL: https://whatsthebigdata.com/2012/06/06/a-very-short-history-of-big-data.

[28] Tom White. *Hadoop: The Definitive Guide*. O'Reilly and Associates, June 2012. ISBN: 978-144931152-0.

[29] *Gartner's 2011 Hype Cycle Special Report Evaluates the Maturity of 1,900 Technologies*. [Online; accessed 7. Jul. 2018]. Aug. 2011. URL: https://www.gartner.com/newsroom/id/1763814.

[30] Gil Press. *A Very Short History of Big Data*. [Online; accessed 7. Jul. 2018]. June 2012. URL: https://whatsthebigdata.com/2012/06/06/a-very-short-history-of-big-data.

[31] Pavlo Baron. *Big Data für IT-Entscheider: Riesige Datenmengen und moderne Technologien gewinnbringend nutzen (Print-on-Demand)*. Carl Hanser Verlag GmbH & Co. KG, Apr. 2013. ISBN: 978-344643339-7.

[32] Mike Loukides. "Big data is dead, long live big data: Thoughts heading to Strata". In: *O'Reilly Radar* (Feb. 2013). URL: http://radar.oreilly.com/2013/02/big-data-hype-and-longevity.html.

[33] Elena Geanina ULARU, Florina Camelia PUICAN, Anca APOSTU, et al. “Perspectives on Big Data and Big Data Analytics”. In: *Database Systems Journal* 3.4 (Dec. 2012), pp. 3–14. URL: https://ideas.repec.org/a/aes/dbjour/v3y2012i4p3-14.html.

[34] Paul Zikopoulos, Dirk deRoos, Krishnan Parasuraman, et al. *Harness the Power of Big Data The IBM Big Data Platform*. McGraw-Hill Education, Nov. 2012. ISBN: 978-007180817-0.

[35] Bernard Marr. *Why only one of the 5 Vs of big data really matters*. [Online; accessed 12. May 2018]. May 2018. URL: http://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters.

[36] *Welcome to Apache™ Hadoop®!* [Online; accessed 7. Jul. 2018]. June 2018. URL: http://hadoop.apache.org.

[37] *HDFS Architecture Guide*. [Online; accessed 7. Jul. 2018]. Aug. 2013. URL: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[38] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. *The Google file system*. Vol. 37. 5. ACM, Oct. 2003. ISBN: 978-1-58113-757-6. DOI: 10.1145/945445.945450.

[39] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492.

[40] *Apache Hadoop 2.9.1 – Apache Hadoop YARN*. [Online; accessed 7. Jul. 2018]. May 2018. URL: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[41] Eric Sammer. *Hadoop Operations: A Guide for Developers and Administrators*. O’Reilly Media, Oct. 2012. ISBN: 978-144932705-7.

[42] *Sqoop -*. [Online; accessed 13. May 2018]. Feb. 2018. URL: http://sqoop.apache.org.

[43] *Welcome to Apache Flume — Apache Flume*. [Online; accessed 13. May 2018]. Oct. 2017. URL: https://flume.apache.org.

[44] *Apache Kafka*. [Online; accessed 13. May 2018]. May 2018. URL: https://kafka.apache.org.

[45] *Apache Kudu - Fast Analytics on Fast Data*. [Online; accessed 13. May 2018]. May 2018. URL: https://kudu.apache.org.

[46] *Apache HBase – Apache HBase™ Home*. [Online; accessed 13. May 2018]. May 2018. URL: https://hbase.apache.org.

[47] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, et al. "Bigtable: A Distributed Storage System for Structured Data". In: *ACM Trans. Comput. Syst.* 26.2 (June 2008), pp. 1–26. ISSN: 0734-2071. DOI: 10.1145/1365815.1365816.

[48] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services". In: *In ACM SIGACT News*. 2002, p. 2002.

[49] *Apache Hive TM*. [Online; accessed 13. May 2018]. May 2016. URL: https://hive.apache.org.

[50] *Impala*. [Online; accessed 13. May 2018]. May 2018. URL: https://impala.apache.org.

[51] *Apache Lucene - Welcome to Apache Lucene*. [Online; accessed 13. May 2018]. Apr. 2018. URL: http://lucene.apache.org/index.html.

[52] *Apache Solr -*. [Online; accessed 13. May 2018]. Apr. 2018. URL: http://lucene.apache.org/solr.

[53] *Oozie - Apache Oozie Workflow Scheduler for Hadoop*. [Online; accessed 13. May 2018]. Apr. 2018. URL: http://oozie.apache.org.

[54] *Welcome to Apache Pig!* [Online; accessed 13. May 2018]. Apr. 2018. URL: https://pig.apache.org.

[55] *Apache Mahout*. [Online; accessed 13. May 2018]. Dec. 2017. URL: https://mahout.apache.org.

[56] *Apache Spark^TM - Unified Analytics Engine for Big Data*. [Online; accessed 13. May 2018]. Apr. 2018. URL: https://spark.apache.org.

[57] *Apache ZooKeeper - Home*. [Online; accessed 13. May 2018]. May 2018. URL: https://zookeeper.apache.org.

[58] *Amazon Simple Storage Service S3 – Cloud Online-Speicher*. [Online; accessed 17. May 2018]. May 2018. URL: https://aws.amazon.com/de/s3/?sc_channel=PS&sc_campaign=acquisition_AT&sc_publisher=google&sc_medium=english_s3_b&sc_content=s3_e&sc_detail=amazon%20s3&sc_category=s3&sc_segment=192093006232&sc_matchtype=e&sc_country=AT&s_kwcid=AL!4422!3!192093006232!e!!g!!amazon%20s3&ef_id=WvKwwgAAAoM12AH6:20180517153506:s.

[59] *Apache Cassandra*. [Online; accessed 17. May 2018]. Feb. 2018. URL: http://cassandra.apache.org.

[60] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. “The Dimensional Fact Model: A Conceptual Model for Data Warehouses.” In: *Int. J. Cooperative Inf. Syst* 7.2 (June 1998), pp. 215–247. ISSN: 0218-8430. DOI: 10.1142/S0218843098000118.

[61] Jim Gray, Surajit Chaudhuri, Adam Bosworth, et al. “Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals”. In: *Data Mining and Knowledge Discovery* 1.1 (Mar. 1997), pp. 29–53. ISSN: 1573-756X. DOI: 10.1023/A:1009726021843.