INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Extraktion und Erweiterung eines Knowledge Graphen für Publikationen und Zitationen

Timo Stroick

Bachelorarbeit

Beginn der Arbeit:	17. Juli 2020
Abgabe der Arbeit:	19. Oktober 2020
Gutachter:	Prof. Dr. Stefan Conrad
	Prof. Dr. Michael Leuschel

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 19. Oktober 2020

Timo Stroick

Zusammenfassung

In der folgenden Arbeit wird der Frage nachgegangen, ob es möglich ist, mehrere Knowledge Graphen in einer Datenbank zu vereinen. Dazu wird ein Parser erstellt, näher erklärt und beschrieben. Dieser soll die Daten von einem Knowledge Graphen, in diesem Fall die *Digital Biblography & Library Project* der Universität Trier, extrahieren. Dieser Graph enthält Daten über Arbeiten, Konferenzen und Autoren. Für diese Daten wird eine Datenbank mit den typischen Maßnahmen für die Modellierung entworfen. Diese beinhalten das Erstellen eines ER-Modells, die Übertragung in ein relationales Schema und die darauffolgende Verschmelzung. Hierzu werden charakteristische Fehler dargelegt und wie man sie mittels Normalformen auflöst. Durch diese Normalisierungen wird festgestellt, dass es sich bei der Datenbank um die 3. Normalform handelt. Daraufhin wird die Datenbank mit Daten gefüllt.

Die Daten für die Zitierungen werden aus dem *Microsoft Academic Knowledge Graph* geholt und benutzt. Dafür wird die zur Verfügung gestellte API verwendet, mit der erste Zitate von den bereits gespeicherten Daten gesucht werden. Folglich werden zu den Zitaten die entsprechenden Titel herausgesucht, bis schließlich die Daten in der Datenbank vereint werden können.

Diese Vorgehensweise zeigt, dass die Vereinigung solcher Datenbanken möglich ist. Allerdings werden nicht immer sicher die richtigen Überschneidungen getroffen, da Publikationen keine eindeutige Identifikation haben.

Inhaltsverzeichnis

1 Einleitung

Weltweit werden in fast allen Programmen, die wir alltäglich nutzen, Datenbanken verwendet. Überall treffen wir auf sie – sei es innerhalb unserer Freundesliste, die unsere Kontakte abspeichert, auf Websiten, die unsere Daten verwalten oder bei Streamingportalen, die unsere Verläufe sichern.

Es gibt diverse Formen von Datenbanken. Neben der hierarchischen oder objektorientierten wird innerhalb dieser Bachelorarbeit insbesondere eine relationale Datenbank genutzt, in die ein Knowlegde Graph umgewandelt und anschließend mit einem weiteren Graphen erweitert wird. Dies soll dazu dienen, zwei Datenquellen miteinander zu vereinen, um die Möglichkeit zu schaffen, sich aus mehreren Quellen zu bedienen. Dafür werden die Datenbanken wie das Digital Biblography & Library Project, kurz DBLP, der Universität Trier und der Microsoft Academic Knowledge Graph verwendet. Beide Graphen beinhalten Publikationen, Facharbeiten, Konferenzen und Fachbücher, von denen letzteres Zitationen umfasst. Die Datenbank der Universität Trier umschließt nur die Daten im Bereich Informatik, welches diese um einiges verringert. Innerhalb dessen werden 5,2 Millionen Puplikationen gespeichert. Im Gegensatz dazu umfasst der Microsoft Graph keinen Fachbereich und hat mit 209,7 Millionen eine größere Auswahl an Puplikationen als die DBLP. (Fußnote von wann die Zahlen sind)

Zu Beginn dieser Arbeit wird zunächst die verwendeten Knowledge Graphen erklärt, die für das gesamte Vorhaben benötigt werden. Anschließend wird einer dieser Graphen, die DBLP, extrahiert und in einer eigens angelegten relationalen Datenbank gespeichert. Dazu wird ein eigener Parser entwickelt und eine Datenbank konstruiert, um diese universell - für Daten im Bereich von Publikation und Zitation – zu nutzen. Nach der Extraktion werden schließlich die extrahierten Daten vom Microsoft Academic Knowledge Graph erweitert. Das darauffolgende Ergebnis dieser Anwendung soll eine Datenbank zeigen, die alle Publikationen im Bereich Informatik und die dazugehörigen Zitate enthält.

Das Resultat dieser Arbeit wird im Fazit reflektiert und könnte innerhalb des Ausblicks auf zukünftige Anwendungen optimiert werden.

2 Knowledge Graph

Um den Knowledge Graphen zu erklären, müssen vorab Graphen erklärt werden. Mit einem Graph wird in diesem Fall kein Funktionsgraph gemeint, der eine Kurve oder eine Gerade darstellt. Hierbei handelt es sich um Graphen aus der Mathematik oder Informatik, die abstrakt eine Menge von Objekten mit ihren Verbindungen repräsentieren. Diese Objekte werden als Knoten, und deren Verbindungen als Kanten bezeichnet. Es gibt dabei zwei Arten von Graphen: Zunächst den ungerichteten, bei dem die Kanten nur als Verbindung zwischen zwei Objekten gesehen werden. Zum Anderen gibt es den gerichteten Graphen, indem die Kanten Richtungen besitzen, die damit einen Weg von einem Knoten zum nächsten erzeugen können. Dabei ist darauf zu achten, dass Knoten beliebig viele Kanten haben können und dass in einem gerichteten Graphen Knoten auch auf sich selbst zeigen können.(vgl. Wikipedia, o.j.(a))

Ein Knowledge Graph ist in diesem Fall ein gerichteter Graph. Dieser repräsentiert eine Menge von miteinander verbundenen Beschreibungen von Objekten.(vgl. ontotext, o.j.) Diese Beschreibungen sind Knoten und enthalten die Daten zu einem Objekt. Diese Objekte besitzen Verweise, die sie mit anderen verbinden. Diese Verbindungen sind die Kanten und sind gerichtet, da sie nur einseitige Verlinkungen besitzen. So entsteht eine Form von Datenbank, die ein Netzwerk aus Informationen auslesen und verändern kann.

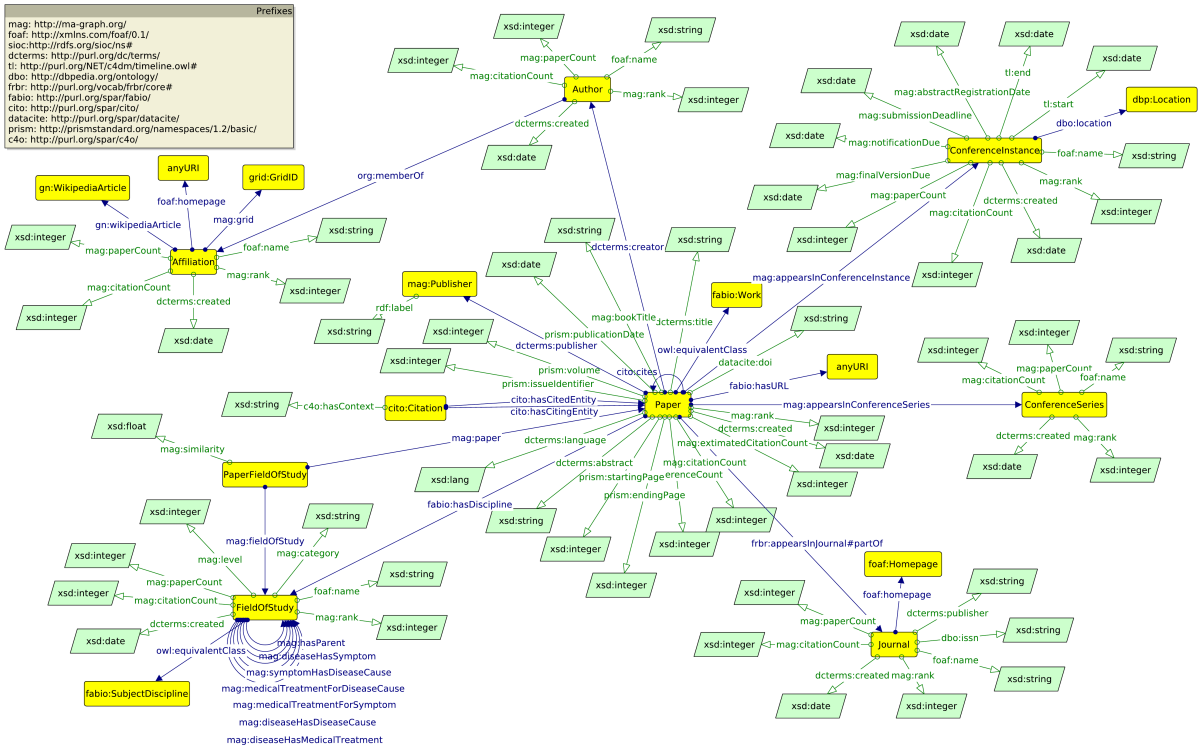

Abbildung 1: Schema vom *Microsoft Academic Knowledge Graph*
(Quelle:http://ma-graph.org/schema-linked-dataset-descriptions/)

Am Beispiel des *Microsoft Academic Knowledge Graphen* von Abbildung 1. wird kenntlich, dass die Objekte oder Knoten in Kästen dargestellt werden. Die gelben Kästen geben ein Objekt oder eine Entität an, was auf andere verweist, was damit in Verbindung steht. Diese Verweise oder auch Verknüpfungen werden mit Pfeilen dargestellt, wobei das die Kanten aus dem Graphen sind. Da diese Pfeile in eine Richtung zeigen, handelt es sich hierbei um einen gerichteten Graphen. Die grünen Kästen stellen Attribute oder Werte dar. Durch den Namen des Attributes, der in der Verknüpfung steht, erhält die Verlinkung ihren Sinn und ihren Namen. Diese Attribute sind Endpunkte oder auch Blätter in dem Graphen und haben keine ausgehenden Kanten, denn sie repräsentieren die Werte der Entität. Die blauen Pfeile verbinden Entitäten und setzen diese in einen Kontext. Beispielsweise ist der ‚Author' ein Ersteller eines ‚Paper's, also verweist ein ‚Paper' auf die eigenen Autoren, um diese zu verbinden. So spiegelt sich die typische Struktur eines Graphen wider.

Die Datenbank der *DBLP* basiert auf dem selben Prinzip wie die des Knowledge Graphen. Nur hat dieser Graph kein gegebenes Schema oder eine spezifische Beschreibung des Aufbaus, da diese in der Dokumentation nicht vorhanden sind. Dies macht das extrahieren und das Designen der eigenen Datenbank um einiges aufwendiger, da dadurch die Daten selbst den Datenaufbau herausfinden.

Im Gegensatz zu diesem Schema steht das relationale Datenbankmodell. Dieses Modell basiert nicht auf einem Graphen, sondern auf Objekten mit Eigenschaften. Diese Objekte werden auch Entitäten genannt. Hier gibt es auch Verweise, wobei der Hauptfokus auf den Entitäten und – wie der Name schon sagt – auf der Relation zwischen diesen Entitäten liegt. Es werden ebenfalls keine Attribute verwiesen, sondern gehören diese direkt zur Entität dazu. Dieses wird im weiteren Verlauf dieser Arbeit näher erläutert, da eine solche Datenbank selbst geplant und erstellt wird.

3 Extraktion

Der vorliegende Abschnitt bezieht sich auf das Extrahieren von Daten. Dieser Ablauf unterteilt sich in Parsen, um die Daten zu Beginn von der DBLP zu extrahieren. Dabei werden diese in passende Entitäten umgewandelt, um sie für die Datenbank nutzen zu können. Um diese danach zu speichern, wird die Datenbank mit diesen Entitäten entworfen.

Im ersten Teil werden die Daten von der *DBLP* extrahiert. Dafür werden zunächst die Daten, die zu extrahieren sind, analysiert. Die *DBLP* bietet dafür zwei Möglichkeiten an: Die erste ist eine Request API bzw. Schnittstelle.

Durch diese API können Anfragen an den Server gesendet werden. Dieser reagiert auf die Anfrage und schickt die Informationen zurück, die durch die Anfrage verlangt wurden. Damit ist es möglich, Daten von der DBLP zu erhalten. Nun kann die Anfrage in dieser API nur bestimmte Informationen abfragen, wie beispielsweise bestimmte Artikel, Autoren und Arbeiten von Autoren. Damit sind nur reduzierte Datenausgaben möglich.

Für diesen Fall ist die zweite Möglichkeit besser geeignet, auf Grund der Tatsache, dass ein Dump File (dblp.xml) genutzt wird. Ein Dump File ist ein Ausdruck, von dem gesamten Inhalt eines Speichers – hier von einer Datenbank. Das File enthält Daten, die die Datenbank zudem Zeitpunkt besitzt. Da sich die Datenbank regelmäßig ändert oder erweitert, erstellt die DBLP regelmäßig neue Dump Files. Das File was benutzt wird, ist vom 13. Mai 2020. Es kann viele Dateiformate besitzen, wobei es sich hier um ein XML-Format handelt. Hierzu gibt es eine Dokumentation, die die Formatierung des XML-Formates beschreibt. Da diese schon „DBLP - Some Lessons Learned“(Ley, 2009) heißt, lässt es darauf schließen, dass einige Fehler bei dem ursprünglichen Design aufgetreten sind und einige Änderungen gemacht wurden. Die erste Dokumentation der DBLP ist vom 18. Juni 2009. Zu diesem Zeitpunkt waren nur 532MP in der XML-Datei. Im Gegensatz dazu sind nun 2.806MB in der Datei enthalten, demnach das Fünffache der ursprünglichen Datei. Durch das Wachstum wurden einige Anpassungen getätigt, die von der Dokumentation abweichen. Somit muss die Formatierung der Datenstrukturen an Hand der Datei selbst herausgefunden werden, da nicht genau festgelegt war, was eine Publikation als Daten enthalten kann. Dies führt zu vielen optionalen Eigenschaften, da nicht alle Daten vom selbem Type die selben Eigenschaften besitzen.

Für das Extrahieren der Daten wird daraufhin ein Parser benötigt, der die XML-Datei ausliest. Die meisten Parser lesen die Datei komplett ein und geben die XML-Daten dann heraus. Da die Datei nun 2,8 GB groß ist, laufen die meisten Parser an ihre Grenzen. Dazu kommt, dass das Dump File in den nächsten Jahren beliebig groß werden kann. Deshalb sind Stream basierte Parser die einzige Lösung. Es gibt damit zwar Parser, mit denen es funktionieren würde, aber in diesem Fall wurde sich bewusst dagegen entschieden.

Daher wird ein eigener Parser geschrieben, um dieses Problem zu umgehen. Durch diesen regulären Ausdruck aus der Dokumentation wird gezeigt, dass es nur eine bestimmte Anzahl an Elementen gibt, die verwendet werden.

<!ELEMENT dblp (article | inproceedings |
proceedings | book | incollection | phdthesis | mastersthesis | www)*>

Das ist ein Ausschnitt aus der Dokumentation(vgl. Ley, 2009, S. 2) und beschreibt das Format der XML. Dieser zeigt alle Elemente (Tags), die in der Datei vorhanden sind. Nach dem testen traten keine anderen Elemente auf. So werden die benötigten Fähigkeiten, die der Parser haben muss, genau festgelegt.

3.1 Parser

Dieses Programm wurde in Python geschrieben. Python ist eine interpretierte, interaktive und objektorientierte Programmsprache.(vgl. Python, 2020) Diese Sprache basiert nicht wie üblich auf Semikola als Trennung, sondern basiert auf Spalten. Damit hat jedes Einrücken eine Bedeutung.

Alle Klassen folgen dem Single Responsibility Prinzip, welches besagt, dass jede Klasse nur eine Aufgabe verfolgt. So werden auch alle Klassen angelegt und geschrieben. Demnach extrahiert die Parser-Klasse die Daten, die DBParser-Klasse teilt die Daten in Entitäten auf und die Insert-Klasse fügt die Daten in die Datenbank ein. So ist das Programm besser wartbar und gibt ein sauberes Programmbild ab.

Bevor der Parser erklärt wird, muss zuerst das Dateiformat XML weiter ausgeführt werden. XML bedeutet Extensible Markup Language und ist eine erweiterbare Auszeichnungssprache, mit deren Hilfe Daten im Textformat gespeichert werden können.

Dies funktioniert, indem mit einem Tag einem bestimmtem Element einen Namen oder eine Kategorie zugeteilt wird, wie bspw. ‚<author> Timo Stroick </author>'. Hierbei bildet alles zusammen ein Element, und das eingeklammerte ‚author' ein Tag. Dabei ist darauf zu achten, dass Elemente immer mit dem dazugehörigen Endtag schließen muss, wie anhand des Beispiels gezeigt. Eine weitere Möglichkeit ist es, Elemente in Elemente zu schreiben. Hier gibt es eine besondere Regel, die XML ausmacht. Elemente, die in andere Elemente geschrieben sind, müssen erst geschlossen werden, bevor die äußeren geschlossen werden dürfen. Durch diese Regeln entsteht die typische hierarchische Struktur des XML-Dateiformats. In der dblp.xml sind alle Daten in ein Tag eingeschlossen, welches dblp heißt. So fängt die Datei bei <dblp> an und enden bei </dblp>. Somit sind alle wichtigen Daten zwischen den beiden Tags.

Der Parser ist in einer eigenen Klasse. Um mit dem Parser zu beginnen, werden die Daten zeilenweise in der Klasse eingelesen, um das Problem mit der Dateigröße zu umgehen. Denn würde alles auf einmal eingelesen werden, würde der Speicher des Programms schnell voll werden und damit den Computer überlasten. Nun wird in den Zeilen nach den Tags gesucht. Ist ein Tag gefunden worden, wird dessen Name überprüft und reagiert mit einem bestimmten Fall. Durch das zeilenweise Einlesen ist dies allerdings kein richtiger Streamparser, denn dieser würde Zeichen für Zeichen einlesen, um die Datei zu parsen.

Da durch die Dokumentation alle Tags bekannt sind, die in der Datei auftreten können, müssen nur acht Fälle behandelt werden. Somit haben wir nur acht Methoden, mit denen wir jeden Fall abdecken. Alle von denen arbeiten genau ein Element ab. Jede Methode liest weitere Zeilen und Daten ein, bis das Endtag des gewissen Elements, die Methode beendet. Das zählt natürlich auch für das dblp-Endtag, welchen den Parser beendet. Die in der Zeit eingelesenen Tags und Werte sind die Attribute des entsprechenden Elements,

```
<article mdate="2020-03-12">
        <author>Moritz Lipp</author>
        <author>Michael Schwarz 0001</author>
        <author>Daniel Genkin</author>
        <title>Meltdown</title>
        <journal>meltdownattack.com</journal>
        <ee>meltdownattack.com/meltdown.pdf</ee>
        <year>2018</year>
</article>
```

Abbildung 2: XML-Ausschnitt
Ein Beispiel Ausschnitt wie ein typischer Eintrag in der dblp.xml aussieht. Dieser wurde verkürzt und angepasst.

wie bspw. bei ‚article', bei dem es der Autor sein kann, der eingelesen wird.

Die Daten werden nun in eine weitere Klasse übergeben. Diese Klasse ist der DBParser, in diesem die Daten nochmals angepasst werden. Da die erste Klasse die Daten aus der XML-Datei in Daten in das Programm wandelt, wird diese Klasse nun die Daten so anpassen, sodass sie für die Datenbank nutzbar werden. Um nachher die Datenbank ordentlich verwenden zu können, werden die acht Elemente in Entitäten gewandelt, die vorteilhafter für das Datenbankdesign sind. Entitäten sind Objekte oder Wesen, die in der Datenbank mit Attributen und Werten befüllt werden können. Diese Einteilung wird im weiteren Verlauf noch gezeigt und erläutert.

Zum Schluss werden in der Insert Klasse die Daten nur noch in die Datenbank eingefügt. Somit sind alle Daten von einem Element so gespeichert, dass der Arbeitsspeicher erneut benutzt werden kann. Damit wird dafür gesorgt, dass nur ein Element gleichzeitig bearbeitet wird und kein Speicherüberlauf auftritt.

Abbildung 3 zeigt einen typischen Programmverlauf eines Parsers. In diesem Beispielablauf wird ein ‚article' Element eingelesen. Zunächst wird die Parser-Klasse mit der startParser-Funktion in der Main-Methode aufgerufen und liest einen Tag nach dem anderen, bis er auf Treffer stößt. In diesem Fall ist das Tag ‚article' ein Treffer. Nun wird die parseArticle-Methode in der selben Klasse aufgerufen. In der Methode werden die darauffolgenden Elemente ausgelesen und deren Daten zwischengespeichert. Dies geschieht solange, bis das Endtag von ‚article' wieder eingelesen wird. Jetzt werden alle Daten an die saveArticle-Methode der DBParser-Klasse übergeben. In dieser Klasse werden nun die Daten an die Datenbank angepasst. Denn in dieser Datenbank gibt es keine Entität, die Artikel heißt. Das wird erst im nächsten Schritt erklärt. Im Allgemeinen wird der Artikel sowohl in Publikation, Autoren, elektronische Version und Fachzeitschrift als auch in die Beziehungen zwischen diesen Entitäten zerteilt. Da die Daten richtig zerlegt sind, werden sie an die Insert-Klasse einzeln übergeben. Das geschieht über die insertPublikation-Methode.

Jetzt sind alle Daten vorhanden und richtig zugewiesen. Daraufhin werden sie mit SQL

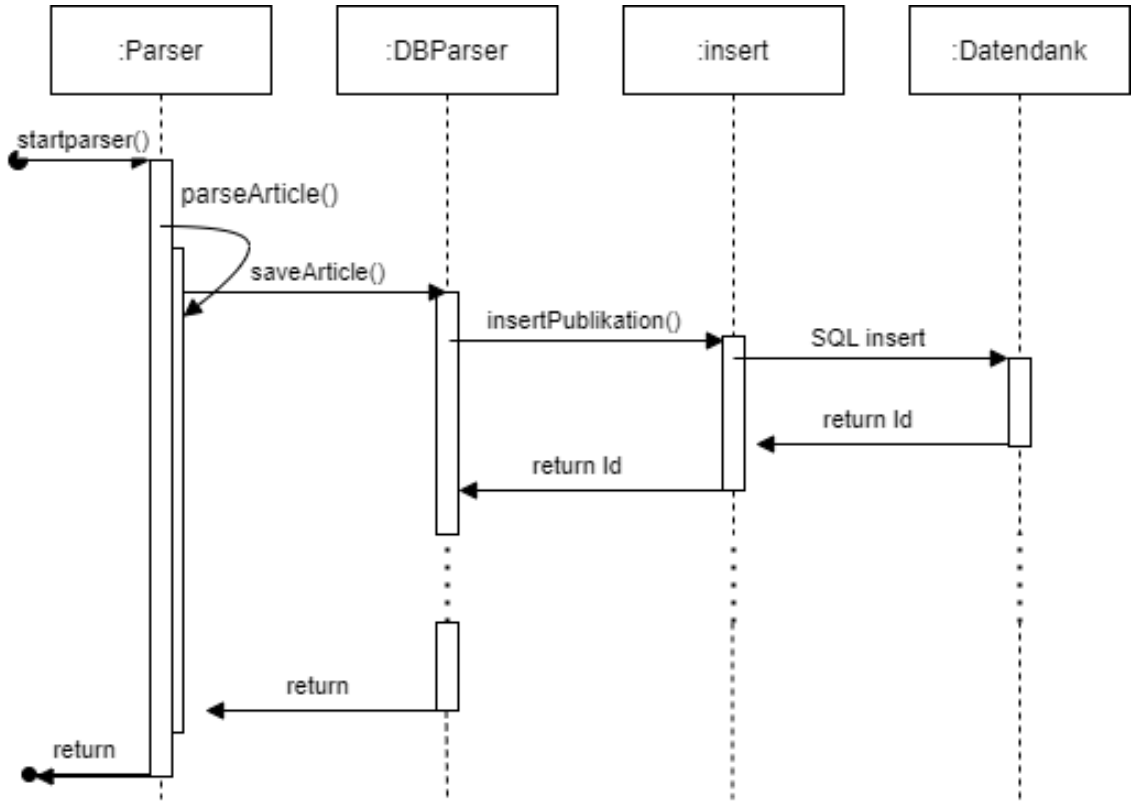

Abbildung 3: Sequenzdiagramm-Auschnitt

in eine Postgres Datenbank gespeichert. Postgres ist ein freies objektrelationales Datenbankmanagementsystem (ORDBMS) und und basiert auf SQL. SQL bedeutet Structured Query Language, also strukturierte Abfragesprache, und stellt einen Standard für relationale Datenbanken dar. Es gibt viele SQL-Datenbanken, wie zum Beispiel Amazon DynamoDB, IBM DB2, MongoDB, PostgresSQL u.Ä.

PostgresSQL wird verwendet, da es sich zum einen um eine freie Oben-Source Datenbank handelt und zum anderen keine Lizenzen benötigt. Dazu hält sie weites gehend die SQL-Standards und Normen ein. Diese werden benötigt, um die Aufgabe ordentlich zu erfüllen. In den Methoden ist nämlich darauf zu achten, dass nur triviale SQL Befehle benutzt werden und keine speziellen eines bestimmten Datenbanktyps. Deshalb wird sich innerhalb dieser Arbeit auf Insert-Befehle und Select-Abfragen fokussiert, um auf keinen bestimmten Datenbanktypen angewiesen zu sein.

Schlussendlich wird der Primärschlüssel der Publikation zurückgegeben, um die Beziehungen zwischen den Entitäten und der Datenbank einzutragen. Wenn nun die saveArticle-Methode zu Ende läuft und sich erfolgreich beendet, sucht die startparser-Methode nach dem nächsten Tag. So wird jedes Element nacheinander in die Datenbank eingespeichert, bis das Endtag von dblp eingelesen wird.

3.2 Datenbank

Zur Erstellung der Datenbank werden zunächst die Daten analysiert. Die *DBLP* gibt schon mit acht Elementen (article, inproceedings, proceedings, book, incollection, phdthesis, mastersthesis, www) einen groben Überblick. Diese Elemente sind allerdings noch nicht in dem relationalen Datenbankformat. Deshalb werden diese, wie vorhin erläutert, nochmal in kleinere Entitäten zerteilt. Für die Einteilung wird der *Microsoft Graph* aus Abbildung 1 als Inspiration genommen, da bei dieser Zerkleinerung die Entitäten in ihrem fachlichen Kontext belassen werden müssen:

‚article' - Es setzt sich aus einer Fachzeitschrift, einer Publikation und mehreren elektronischen Versionen zusammen.

‚proceedings' - Dies ist in diesem Zusammenhang eine Konferenz, die gehalten wurde und besitzt Autoren, die diese Sammlung von Arbeiten angepasst haben. Diese Konferenz kann auch ISBN's enthalten.

‚inproceedings' - Passend zu Konferenzen und ist eine Publikation, die in einer Konferenz veröffentlicht wurde.

‚book' - Book ist ein Buch, mit ein oder mehreren Autoren. Hierzu gehören auch mehrere ISBN's.

‚incollection' - Dies sind Publikationen, die in einem Buch veröffentlicht wurden.

‚masterthesis' - Die Masterthesis ist eine Publikation, die an einer Universität geschrieben wurde.

‚phdthesis' - Im Gegensatz zur Masterthesis sind die phdthesis immer als Buch veröffentlicht.

‚www' - Dies sind Homepages von Autoren, die abgegeben werden können.

Jede Publikation in den Erklärungen besitzt zusätzlich Autoren, die hier nicht aufgelistet sind. Dazu kommt noch, dass jeder Punkt elektronische Versionen enthält. Da in der *DBLP* Bücher sind, die mehrere ISBN's haben, muss eine extra Entität für die ISBN erstellt werden. Das Gleiche zählt für die elektronische Version, denn diese existiert für die meisten Entitäten mehrfach. Daraus ergeben sich dann diese Entitäten Autor, Publikation, Homepage, Konferenz, Fachzeitschrift, Buch, elektronische Version, ISBN und deren Beziehungen. Zu diesen Entitäten gehören ebenso die Attribute, die schon mit dem Parser eingelesen werden. Da nicht für jede Entität immer die gleiche Anzahl an Eigenschaften übergeben wird, werden die meisten Attribute optional. Mit den Entitäten und deren Attribute wird dann ein ER-Modell modelliert und an die entsprechenden Beziehungen angepasst.

3.2.1 ER-Modell

Das ER-Modell veranschaulicht die Beziehungen zwischen den Entitäten. Die Entitäten sind hier die Rechtecke. Die Attribute sind Kreise und direkt mit den Entitäten verbunden, zu denen sie gehören. Beziehungen werden durch Rauten dargestellt und verbinden immer zwei Entitäten oder nur eine Entität mit sich selbst (siehe Beziehung hat-Zitat). Ein

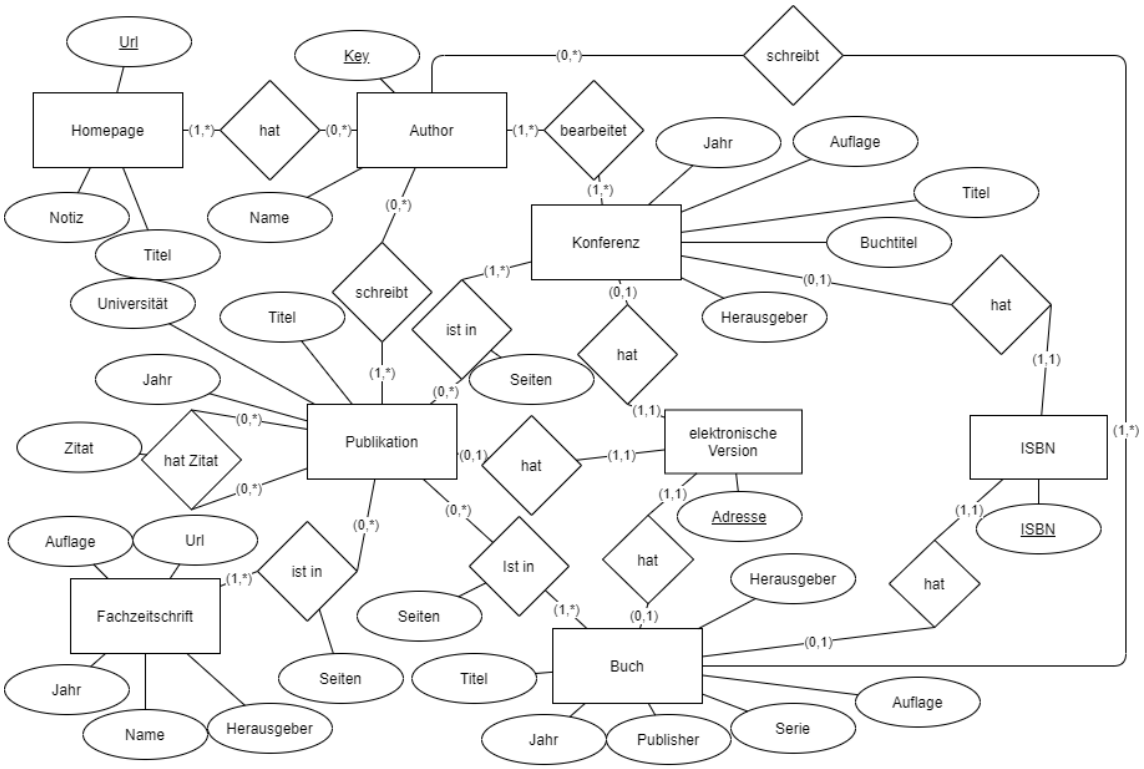

Abbildung 4: ER-Modell

Zitat verbindet eine Publikation, mit der Publikation, die zitiert wurde. Es wird deutlich, dass Beziehungen auch Attribute haben können, denn es gibt Fälle, wo dieses Attribut zu keinem von beiden passt. Das Attribut passt nur, wenn beide in Beziehung zueinander stehen. Die Relation zwischen Publikation und Buch besitzt sie das Attribut Seiten. Dies macht nur im einzelnen Zusammenhang Sinn, da Seiten allein in einer Publikation ohne Buch keinen Anhaltspunkt besitzen. Andersherum kann ein Buch mehrere Publikationen beinhalten und deshalb nicht verschiedene Seitenangaben für verschiedene Publikationen speichern. Somit macht die Angabe von Seiten nur in der Beziehung der beiden Entitäten Sinn. Mit Attributen werden Entitäten beschrieben und es gibt Attribute, die für eine eindeutige Identifikation der Entität sorgen. Diese Attribute werden unterstrichen und nachher zu Schlüsselkandidaten. Die eindeutige Eigenschaft bei Autor ist der Key. Dieser Key sieht wie in Abbildung 2. aus und ist der komplette Name eines Autors. Falls dieser Name doppelt vorkommt, haben die beiden Autoren eine zusätzliche Nummer hinter ihrem Namen. Dadurch wir dieser wieder ein Unikat.

Dieses Modell muss nun umgewandelt werden. Da in der Datenbank nur Tabellen angelegt werden können, muss dieses ER Modell in die Form von Tabellen gebracht werden. Diese Form ist dann das relationale Schema. Dieses Schema beschreibt genau das Gleiche, wie das ER-Modell, nur hier werden Fremdschlüssel und damit Beziehungen angegeben.

Autor(Key, Name)
Homepage(Url, Titel, Notiz)
Publikation(Id, Titel, Jahr, Universität)
Buch(Id, Titel, Jahr, Serie, Auflage, Herausgeber)
Fachzeitschrift(Id, Url, Name, Nummer, Auflage, Jahr, Herausgeber)
Konferenz(Id, Titel, Buchtitel, Serie, Auflage, Jahr, Herausgeber)
elektronischeVersion(Adresse)
ISBN(ISBN)
Publikation_ist_in_Konferenz(PublikationId, KonferenzId,Seiten)
Publikation_ist_in_Fachzeitschrift(PublikationId, FachzeitschriftId,Seiten)
Publikation_ist_in_Buch(PublikationId, BuchId,Seiten)
Autor_bearbeitet_Konferenz(AutorKey, KonferenzId)
Autor_schreibt_Publikation(AutorKey, PublikationId)
Autor_schreibt_Buch(AutorKey, BuchId)
Autor_hat_Homepage(AutorKey, HomepageUrl)
Publikation_hat_Zitat(HatZitatKey, IstZitiertKey, Zitat)
Konferenz_hat_elektronischeVersion(KonferenzId, Adresse)
Buch_hat_elektronischeVersion(BuchId, Adresse)
Publikation_hat_elektronischeVersion(PublikationId, Adresse)
Konferenz_hat_ISBN(KonferenzId, ISBN)
Buch_hat_ISBN(BuchId, ISBN)

Abbildung 5: Relationales Schema

3.2.2 Relationales Schema

Abbildung 5 zeigt das ER-Modell, welches genau in das relationale Schema übertragen wurde. Der erste Name ist der Name der Tabelle und alle Namen in der Klammer sind deren Attribute. Es wird deutlich, dass auch Beziehungen zwischen den Entitäten eine eigene Tabelle bekommen. Diese sorgen dafür, dass Entitäten mit einander verbunden werden. Die unterstrichenen Attribute stellen die Primärschlüssel da. Primärschlüssel sind Attribute, womit die Daten dieser Entität genau unterscheiden werden können, wie bspw. die Studierendennummer, die sich bei jedem* Studenten* unterscheidet. Gibt es keine eindeutige Eigenschaft, so wird ein künstlicher Schlüssel erstellt. In diesem Fall heißen alle künstlichen Schlüssel Id. Diese Id wird mit jeder Erstellung eines neuen Eintrags erhöht, damit diese eindeutig bleibt. Die Attribute die mit Punkten unterstrichen wurden, sind Fremdschlüssel. Diese Schlüssel sind Primärschlüssel aus anderen Tabellen. Um sich diese besser vorstellen zu können, wird ein Beispiel angeführt:

Es gibt 2 Entitäten, Schüler und Klasse. Der Primärschlüssel für die Klasse wäre die Klassen Bezeichnung, wie 2a oder 1b. Der Schüler hat eine Id als Primärschlüssel, da der Name nicht ganz eindeutig ist. Um nun die Relation so darzustellen, dass der Schüler in eine Klasse geht, wird den Klassen Schlüssel an die Schüler als Fremdschlüssel angefügt. Nun steht in der Schülertabelle bspw.:

Name: Max Mustermann und Klasse: 3a.

Somit dient der Fremdschlüssel dafür dass Entitäten verbunden werden können. Für das

erste wird in dem Schema jede Beziehung als Tabelle mit zwei Fremdschlüsseln erstellt. Die Schlüssel kommen aus den Entitäten, die sie verbinden. Damit stellen sie eine eindeutige Verbindung zwischen Entitäten her.

Da hier noch überflüssige Tabellen enthalten sind, werden die Tabellen verschmolzen.

3.2.3 Verschmolzenes relationale Schema

Autor(Key, Name)
Homepage(Url, Titel, Notiz)
Publikation(Id, Titel, Jahr, Universität)
Buch(Id, Titel, Jahr, Serie, Auflage, Herausgeber)
Fachzeitschrift(Id, Url, Name, Nummer, Auflage, Jahr, Herausgeber)
Konferenz(Id, Titel, Buchtitel, Serie, Auflage, Jahr, Herausgeber)
elektronischeVersion(Adresse)
ISBN(ISBN, BuchId, KonferenzId)
Publikation_hat_Zitat(HatZitatKey, IstZitiertKey, Zitat)
Autor_hat_Homepage(AutorKey, HomepageUrl)
Autor_bearbeitet_Konferenz(AutorKey, KonferenzId)
Autor_schreibt_Publikation(AutorKey, PublikationId)
Autor_schreibt_Buch(AutorKey, BuchId)
Publikation_ist_in_Konferenz(PublikationId, KonferenzId, Seiten)
Publikation_ist_in_Fachzeitschrift(PublikationId, FachzeitschriftId, Seiten)
Publikation_ist_in_Buch(PublikationId, BuchId,Seiten)
Publikation_hat_elektronischeVersion(PublikationId, Adresse)
Buch_hat_elektronischeVersion(BuchId, Adresse)
Konferenz_hat_elektronischeVersion(KonferenzId, Adresse)

Abbildung 6: Verschmolzenes Relationale Schema

Abbildung 6 zeigt nun das verschmolzene relationale Schema. Einige Tabellen sehen jetzt anders aus. Zum Beispiel verschwindet die Tabelle Buch-hat-ISBN ganz, denn der Primärschlüssel vom Buch steht nun direkt in der ISBN, da es nur ein Buch für eine ISBN geben kann. Dasselbe passiert mit Konferenz und ISBN. Dies ist eine N zu 1 Beziehung gewesen, da Fälle auftreten, wo Bücher mehrere ISBN's haben. Die meisten anderen Beziehungen sind N zu M Relationen, wie bspw. Autor und Publikation. Jeder Autor kann mehrere Publikationen schreiben und andersherum, denn Publikationen können von mehreren Autoren geschrieben werden. Diese Relationen können nicht verschmolzen werden. Der letzte Fall wäre eine 1 zu 1 Relation, aber die kommt in diesem Modell nicht vor. Ein typisches Beispiel ist eine Person und ihr Ausweis, denn eine Person hat nur einen Ausweis und ein Ausweis darf nur einer Person gehören.

3.2.4 Normalisierung

Normalisierungen sind dafür da, Fehler und Redundanzen zu verhindern. Redundanzen sind doppelt oder mehrfach vorkommende Informationen. Diese können bei großen Datenbanken für viel unnötigen Platzverbrauch sorgen. Fehler führen dazu, dass die Datenbank abstürzt oder Befehle verweigert werden. Die Normalisierung werden in Stufen von Normalformen getätigt.

Je höher die Stufe der Normalform, desto weniger Anomalien können auftreten. Anomalien sind die Ergebnisse von genau diesen Redundanzen, die denn Umgang mit der Datenbank erschweren oder sogar ganz stoppen. Es können Daten gelöscht werden, die noch gespeichert sein sollten. Werte und Datensätze können verschieden sein, die eigentlich gleich seien sollten. Insgesamt gibt es drei Anomalien, die einfach durch die Normalisierungen verhindert werden können.

Die Einfüge-Anomalie ist eine Anomalie, die beim Einfügen von Daten in der Datenbank auftritt. Es kann passieren, dass ein Primärschlüssel aus zwei Schlüssel besteht. Da das Einfügen nun verlangt, dass beide Primärschlüssel vorhanden sind, führt dies zu Schwierigkeiten oder auch zu Fehlern, wenn nur ein Teil vom Primärschlüssel besessen wird.

Beispiel:

PersonId	Vorname	Nachname	Kennzeichen	Automarke
1	Max	Mustermann	A BC 123	VW
2	Heinz	Paul	C BA 321	BMW
?	?	?	C OM 1337	Ferrari

Abbildung 7: Einfüge-Anomalie Beispiel

Es wird davon ausgegangen, dass die Schlüssel PersonId und Kennzeichen zusammen den Primärschlüssel ergeben, da eine Person mehrere Autos haben kann. Soll jetzt ein neues Auto eingegeben werden, führt das zu Problemen, denn ein Auto kann in diesem Beispiel nicht ohne eine Person existieren. In Abbildung 7 ist der Ferrari ohne Person und hat damit keinen vollständigen Primärschlüssel. Dies würde auch anderes herum zu Problemen führen, da eine Person auch nicht ohne ein Auto existieren kann. Somit haben wir eine Einfüge-Anomalie.

Die Änderungs-Anomalie tritt beim Ändern von Daten in der Datenbank auf. Wenn viele redundante Daten in einer Tabelle sind und ein Name geändert werden soll, tritt diese Anomalie auf, wenn dieser Namen an vielen Stellen geändert werden müsste - Wobei in diesem Fall eine Änderung reichen sollte.

Beispiel:

Hier ist der Primärschlüssel die AlbumId. Die Probleme treten auf, wenn der Namen von Queen geändert werden würde. Normalerweise sollte eine Änderung reichen, um ein Attributwert zu ändern. In diesem Fall müsste jedes Album durchgegangen werden und da die Namen anpassen. Dieser extra Aufwand ist die Änderungs-Anomalie.

Die Lösch-Anomalie entsteht durch das Löschen von Daten aus der Datenbank.Wenn

AlbumId	Albumtitel	Interpret	Gründungsjahr	Erscheinungsjahr
1	Jazz	Queen	1970	1978
2	Innuendo	Queen	1970	1991
3	Arrival	ABBA	1972	1976

Abbildung 8: Lösch- und Änderungs-Anomalie Beispiel

zwei unabhängige Daten gelöscht werden sollen und durch denn Aufbau beide Informationen gelöscht sind, entsteht eine Anomalie. Das Löschen sollte nur die Daten löschen, die auch beabsichtigt zu löschen sind.

Beispiel aus Abbildung 8:

Der Primärschlüssel ist immer noch AlbumId. Nun soll nichts mehr geändert, sondern gelöscht werden. Wenn jetzt das Album Arrival von ABBA gelöscht wird, würde damit auch direkt die Band ABBA gelöscht - obwohl nur beabsichtigt wurde, das Album zu löschen.

Diese Anomalien werden alle durch die drei Normalformen beseitigt. Verschiedene Stufen der Normalformen sind:

Die erste Normalform besagt, dass jedes Attribut atomar sein soll. Damit ist gemeint, dass Daten zerkleinert werden sollen, die auch im einzelnen Aspekt eine Gewichtung haben, wie zum Beispiel Adressen. Adressen können als ein Ganzes zusammen geschrieben werden, aber Atomar wären sie erst, wenn sie in Straße, PLZ, Hausnummer und Stadt einteilt würden. Diese Einteilung verschafft nachher Vorteile, die für Datenbanken sinnvoll sind. Es kann einfacher nach Leuten gesucht werden, die alle in einer Stadt wohnen. Ohne diese atomaren Attribute müssten die Adressen für jede einzelne Suche zerlegen werden. Danach würde es erst möglich sein, die Städte zu vergleichen. Diese Regel wird leider in dieser Datenbank gebrochen, da die Namen von Autoren nicht in Vor- und Nachnamen geteilt sind.

In der DBLP sind die Autoren als ein ganzer Name gespeichert. Wären sie im Voraus getrennt gewesen, würden hier die Namen in Vor- und Nachname geteilt sein. Namen zu trennen ist eine Aufgabe für sich. Namen wie Jürgen von der Lippe oder Angela Dorothea Merkel haben schon verschiedene Trennungsmöglichkeiten. Hier ist zu sehen, dass es nicht einfach möglich ist, ab einer bestimmten Anzahl von schon eingelesenen Namen den Nachnamen zu bestimmen. Würde man den letzten Namen als Nachname festlegen, hat Jürgen von der Lippe nur einen Teil seines Nachname. Diese zu verallgemeinern, ist eine größere Aufgabe. Dazu kommen Namen aus Ostasien, die wiederum Vor-und Nachname andersherum aufschreiben, da der Familien Name zuerst genannt wird. Diese Aufteilung würde das Ausmaß dieser Arbeit überschreiten, sodass angenommen wird, dass Namen atomar sind.

Für die zweite Normalform muss sie erst mal in der 1. Normalform sein und dazu darf kein Attribut, welches auch kein Primärschlüssel ist, eine Teilmenge in der Tabelle besitzen, die von dem Attribut abhängig ist. Demnach müssen Attribute vollständig von einem Primärschlüssel abhängig sein und nicht nur halb. Hierdurch wird auch die Einfüge-Anomalie gelöst, da es nun keine zwei einzelne Entitäten mehr gibt, die in eine Tabelle

zusammen gefügt sind. Um dies zu verdeutlichen, wird erneut ein Beispiel angeführt:

Person(Id, Name, Nachname, Kennzeichen, Automarke, Autotyp)

In dieser Tabelle wäre das Kennzeichen ein Primärschlüssel von einer Automarke und eines Autotyp und damit hat ein Nichtprimärattribut eine Teilmenge von Attributen, die von ihm abhängig ist. Zwar gehört das Auto dieser Person, aber es wäre nur vollständig abhängig von der Kombination der beiden Attribute Id und Kennzeichen. Damit ist diese Tabelle nicht in der zweiten Normalform, denn dafür müssen die Tabellen so aussehen:

Person(Id, Name, Nachname, Kennzeichen)
Auto(Kennzeichen, Automarke, Autotyp)

Für die dritte Normalform muss zunächst die Tabelle in der zweiten Normalform sein und dann darf kein Nichtprimärschlüssel transitiv von einem Schlüsselkandidaten abhängen. Schlüsselkandidaten sind Attribute, die als Primärschlüssel Infrage kommen. Durch diese Normalform werden direkt zwei Anomalien beseitigt, die Änderung-Anomalie und die Lösch-Anomalie.

Um dies zu verdeutlichen, ein weiteres Beispiel:

Album(Id, Albumtitel, Interpret, Gründungsjahr, Erscheinungsjahr)

Hier wäre das Gründungsjahr transitiv über Interpret abhängig, denn Gründungsjahr wäre erst über den Interpreten mit dem Album verlinkt und würde somit die Regel brechen. Demnach muss der Interpret eine eigene Entität werden. Damit sehen die Tabellen dann so aus:

Album(Id, Albumtitel, Erscheinungsjahr, Interpret)
Künstler(Interpret, Gründungsjahr)

Die Tabelle ist in der dritten Normalform, wenn man davon ausgeht, dass der Interpret eindeutig ist, ansonsten wird noch ein künstlicher Schlüssel eingefügt.

In der dritten Normalform sind nun alle Anomalien beseitigt. Das sind die Grundeigenschaften, die eine Datenbank erfüllen muss. Diese Datenbank erfüllt alle Kriterien der dritten Normalform, wenn davon ausgegangen wird, dass Namen ungetrennt schon atomar sind, falls nicht wären sie in der nullten Normalform.

3.3 Datendefinition in der Datenbank

Nun wurde der gesamte theoretische Teil dieser Datenbank beschrieben. Für die Praxis wird SQL verwendet, bei dem es sich um eine Datensprache handelt. Sie ist dafür da, Datenbanken zu erstellen und zu verwalten. Diese Sprache teilt sich in drei Hauptteile von Befehlsgruppen ein, DDL für die Struktur der Datenbank, DML für das Daten manipulieren und DQL für das auslesen der Daten. Diese drei Teile werden benötigt und noch im weiteren Verlauf erklärt.

Hierzu wird DDL - *ausgeschrieben Data Definition Language* - verwendet. Diese Sprache wird benutzt, um die Datenstruktur, die theoretisch aufgebaut wurde, in der Datenbank anzulegen. Also alle Tabellen, Beziehungen, Datentypen u.Ä.

```
CREATE TABLE AUTORSCHREIBTBUCH(
KEY  VARCHAR(100)  NOT NULL,
ID  INTEGER  NOT NULL,
PRIMARY KEY(KEY,ID),
FOREIGN KEY(KEY)  REFERENCES AUTOR(KEY)
ON DELETE CASCADE  ON UPDATE CASCADE,
FOREIGN KEY(ID)  REFERENCES BUCH(ID)
ON DELETE CASCADE  ON UPDATE CASCADE);
```

Abbildung 9: DDL Ausschnitt

Abbildung 9 zeigt einen typischen SQL Auschnitt, in der die Tabelle Autor-schreibt-Buch erstellt wird. Dabei ist zu beachten, dass SQL nicht case sensitive ist, also achtet die Sprache nicht auf Groß- und Kleinschreibung. Wie hier zu sehen ist, wird erst der Create-Befehl benutzt, sodass eine Tabelle namens AUTORSCHREIBTBUCH erstellt wird und in der dahinterliegenden Klammer die Tabellenstruktur kommt. Hier werden erst die beiden Attribute Key und ID erstellt. Hinter den jeweiligen Namen kommen die Datentypen in Varchar und Integer. Varchar steht für eine variable Zeichenkette mit einem Maximum an Zeichen. Variabel ist sie, da sie nicht immer die maximale Anzahl als Speicher einnimmt. Integer steht für eine Ganzzahl. Fasst alle Attribute sind in Varchar gespeichert, da die DBLP nicht genormt ist. Viele Werte, die eigentlich ein Integer wären, werden durch Ausnahmen wieder zu Varchar, bspw. gibt es bei Büchern das Attribut Auflage(Volume). Durchweg werden hier Ganzzahlen verwendet, die Auflagen beschreiben, aber manche Auflagen sind mit langen komplizierten Zeichenketten beschrieben, was Integer wieder hinfällig macht. Deshalb sind nur die künstlichen Schlüssel vom Typ Integer. Diese Tabelle ist die Relation zwischen Autor und Buch. Deshalb sind beide Attribute Fremdschlüssel(Foreign Key). Erst wird der Fremdschlüssel als ein solcher markiert und danach wird dieser auf die Tabelle mit dem Attribut, zu dem es gehört, referenziert. Dabei spielt es keine Rolle, ob der Fremdschlüssel denselben Namen hat wie der Primärschlüssel, den er repräsentiert. So könnte auch der Fremdschlüssel umbenannt werden und im Nachhinein trotzdem auf den richtigen Primärschlüssel verweisen. Es könnte zum Beispiel ID auf BuchId geändert werden, wozu nur die zwei ID's umbenannt werden, aber dabei nicht die Referenz geändert muss.

Unter dem Foreign-Key-Befehl stehen noch optionale Eigenschaften zu Fremdschlüsseln.

Hiermit wird beschrieben, was bei einer Änderung oder Löschung von dem Primärschlüssel in der referenzierten Tabelle passiert. Es gibt fünf Möglichkeiten, die eingestellt werden können:

NO ACTION - Hier wird nichts mit der Tabelle getan. Da dann aber ein Fremdschlüssel auf einen Datensatz verweist, wird ein Fehler geworfen. Dies ist auch die Standardeinstellung, wenn keine Option gewählt wurde.

RESTRICT - Es wird verboten, den verwiesenen Primärschlüssel zu löschen.

CASSCADE - Der Tabelleneintrag wird gelöscht/geändert, falls der verwiesene Primärschlüssel gelöscht wird oder sich ändert.

SET NULL - Der Fremdschlüssel wird auf Null gesetzt, wenn der verwiesene Primärschlüssel gelöscht wird oder sich ändert.

SET DEFAULT - Wie bei SET NULL, nur hier wird der Fremdschlüssel auf den voreingestellten Standardwert gesetzt.

In diesem Fall werden beide Male CASSCADE benutzt, da dies eine Relationstabelle ist und diese die Änderungen und Löschungen mit übernehmen soll. Da diese Tabelle nur die Beziehung darstellt und keine wirklichen Daten besitzt, geschehen hiermit keine Löschungen oder Änderungen, die Fehler erzeugen könnten.

3.4 Daten einfügen

Nun wird die Datenbank, mit den Daten die schon mit dem DBParser angepasst wurden, befüllt. Hierfür wird Data Manipulation Language (DML) verwendet, also die Datenbearbeitungssprache, welche wieder einen Teil von SQL darstellt. Mit diesem Teil können Daten eingefügt, geändert oder auch gelöscht werden. In diesem Teil der Arbeit wird die Hauptaufgabe das Einfügen sein.

```
INSERT INTO AUTOR (KEY, NAME)
VALUES (???,???);
```

Abbildung 10: DML Ausschnitt

In Abbildung 10 wird ein Datensatz in die Autor-Tabelle eingetragen. Mit dem INSERT INTO Befehl wird bestimmt, dass Daten eingespeist werden sollen. Nach dem INTO kommt die Tabelle in welche gespeichert wird. In der Klammer sind dann die Attribute angegeben, die bei diesem Datensatz befüllt werden sollen. Hinter VALUES in der Klammer werden nun die Daten in der Reihenfolge eingefügt, wie sie in der ersten Klammer schon angegeben wurden, ein. Die Fragezeichen stellen in diesem Beispiel die Daten da. Da der Key ein Primärschlüssel ist, tritt ein Fehler auf, wenn zwei Mal der gleiche Wert dafür einfügt wird. Dadurch wird klar, dass der Datensatz schon vorhanden ist.

4 Erweiterung

In diesem Abschnitt geht es hauptsächlich um die Erweiterung der Datenbank. Erst wird die richtige Methode ausgewählt, mit der die Daten geholt werden. Dann werden die Informationen über die Zitate abgerufen und nachher in die Datenbank eingefügt.

Für die Erweiterung wird der *Microsoft Academic Knowledge Graph*(siehe Färber, 2019) benutzt, da dieser im Gegensatz zur *DBLP* zusätzliche Zitate hat und diese, um die entsprechenden Zitate, erweitert werden können.

Es existieren wieder zwei Möglichkeiten, an die Daten zu kommen, eine API und ein Dump File. Das Dump File wurde seit 2018 nicht mehr aktualisiert,trotzdem hat die Summe aller Daten des Dump Files 1,2 Terabyte. Da mit diesem Teil nur die bestehende Datenbank erweitert werden soll, lohnt es sich nicht, eine Datenbank runter zuladen, die viel größer ist als diese. Da nur nach Zitaten gesucht wird, lohnt sich die Vorgehensweise mit den Dump Files nicht.

Deswegen wird hier das *Project Academic Knowledge* verwendet. In diesem Projekt gibt es die *Academic Search API*. (siehe Microsoft, o.j.) Also eine Schnittstelle, mit der in dem Graph gesucht und Daten ausgelesen werden können. Es werden vier verschiedene REST API's zur Verfügung gestellt: CalcHistogram, Interpret, Similarity und Evaluate.

CalcHistogram - Hier werden alle Suchergebnisse zusammen gerechnet und ein Histogramm darüber erstellt. Dabei kann eine Häufigkeitsverteilung über alle Attributswerte von den gesuchten Publikationen erhalten werden.(vlg. APIs, o.j.)

Interpret - Diese Schnittstelle funktioniert wie eine automatische Vervollständigung und gibt Lösungsvorschläge an. Dies kann am besten nach jeder Zeicheneingabe abgefragt werden.(vlg. APIs, o.j.)

Similarity - Dies gibt die Möglichkeit, zwei verschiedene Publikationen auf Gleichheit zu prüfen. Die Rückgabe ist eine Gleitkommazahl die von +1.0, sehr ähnlich bis zu -1.0 sehr unähnlich reicht.(vlg. APIs, o.j.)

Evaluate - Mit Evaluate können normale Suchanfragen gestellt werden und es wird eine Menge von akademischen Entitäten zurückgegeben, die dazu passen.(vlg. APIs, o.j.)

In dem Fall wird Evaluate verwendet, da weder ein Histogramm, noch einen Vergleich oder eine automatische Vervollständigung gebraucht wird. Bei dieser Anfrage können bestimmte Suchkriterien selber eingestellt werden. Dafür muss ein vorgegebenes Suchattribut mit einem Suchwert verglichen werden. Hier kann zum Beispiel als Attribute Name des Autors ausgewählt werden, wo dann der Suchwert den Namen des zu suchenden Autors enthält. Für diese Attribute gibt es eine ganze Dokumentation.

Ein paar Beispiele werden später benannt. Wird nichts weiteres angegeben, werden alle Informationen über die gesuchte Entität ausgegeben. Um diese Ausgabe zu verringern, werden bei der Suche weitere Optionen angegeben, mit der die Ausgabe angepasst wird. Hier können Optionen gewählt werden wie count, um die Anzahl der Ergebnisse anzupassen, offset, um nicht die ersten Ergebnisse ausgegeben zu bekommen, orderby, um nach einem Attribute zu sortieren oder attributes, um nur bestimmte Attribute auszugeben.

```
{
 "expr": "Composite(AA.AuN=='jaime teevan')",
 "entities": [{
  "logprob": -16.973,
  "prob": 4.25323872E-08,
  "CitCon": {
   "1114905064": ["A December 2008 survey by the Pew
       Internet Project [20] found 35% of adult
      internet users in the U.", "For example, the
      number of adults with social network profiles
      quadrupled between 2005 and 2008 [20], and
      users over 35 are the fastestgrowing Facebook
      demographic [11]."],
   "1500842519": ["To better understand the nature
      of the example questions provided by
      respondents, two of the authors used an
      affinity diagramming technique [6] to
      iteratively develop a classification scheme
      for question type and question topic."],
   ...
  }
 }
}
```

Abbildung 11: Response Beispiel

Die Response aus Abbildung 11 ist im JSON-Format. Diese dient genau wie XML für die Speicherung von menschlich lesbaren Daten.(vlg Wikipedia, o.j.(b)) Der gezeigte Ausdruck ist ein Beispiel, in dem nach einem Autor namens 'jaime teevan' gesucht wird. Die Ausgabe der Attribute wurde nur auf 'CitCon' beschränkt. Dies ist das Attribute für Zitate. Dennoch werden 'logprob' und 'prob' hinzugefügt. Da die Responses sehr kurz sein werden, müssen keine Parser selbst geschrieben werden. Somit kann ein fertiger Parser benutzen werden, der die Daten heraus filtert. Nun muss die Anfrage gesendet werden.

Mit dieser relativ simplen API kommt aber auch eine Grenze. Es muss ein kostenloses Abonnement abgeschlossen werden, um Zugriff zu erhalten. Dafür ist ein an den Account gebundener Schlüssel nötig, mit dem die API benutzt werden kann. Im Monat dürfen mit diesem Schlüssel nur 10.000 Transaktionen getätigt werden. Durch diese Einschränkung werden hier nur selektive Beispiele präsentiert. Deshalb wird hier eine Funktion genutzt, die die ersten 200 Publikationen ausgibt. Mit diesen 200 Daten wird das Erweitern getestet und begutachtet.

Zunächst werden aus der Datenbank 200 Titel heraus genommen. Diese 200 Titel werden nun einzeln angepasst, um sie mit einer Request loszuschicken, denn die API kann keine Sonderzeichen oder Großschreibung verwenden. Nachdem die ganze Punktierung entfernt und kleingeschrieben wurde, wird die Anfrage mit der Bedingung geschickt, dass ‚Ti' gleich des angepasstem Titel sein soll. ‚Ti' steht für normalized title. Dieser Titel ist auch klein geschrieben und ohne Sonderzeichen. Damit werden schnell passende Publikationen gefunden. Mit der Filtereinstellung ‚CitCon' kann nun die Id und der Abschnitt,

der zitiert wurde, erhalten werden.

Nun wird eine Antwort erhalten wie in Abbildung 11. Daraufhin werden die wichtigen Informationen mit Hilfe eines Standard JSON-Parsers herausgefiltert. Falls der Titel kein Zitat besitzt, wird der Titel übersprungen und der nächste Titel angefragt. Wenn doch ein Zitat zu diesem Titel gefunden wurde, sind die relevanten Daten in der Antwort. Diese sind die Paper Id und das Zitat an sich.

Die Paper Id ist in dieser Datenbank nicht zu gebrauchen, da diese nur zu den Publikationen in dem Microsoft Graphen gehören. Deswegen muss diese Id in etwas umwandeln werden, mit dem gearbeitet werden kann. In diesem Fall ist das wieder ein Titel. Für die Umwandlung wird eine weitere Anfrage an die Schnittstelle gestellt. Diese sucht nach Publikationen mit der selben ID, sodass der Titel zurückgegeben wird. Da die API nur Probleme bei der Eingabe von Sonderzeichen hat, aber nicht bei der Ausgabe,wird der Titel mit Sonderzeichen und Großschreibung zurückgegeben. Dazu wird der Filterwert ‚DN' verwendet, der den originalen Titel liefert.

Somit sind alle Daten vorhanden, die gebraucht werden, um ein Zitat in die Datenbank einzuspeisen. Zunächst gibt es den Titel, wofür die Zitate gesucht wurden. Dann wurde ein das Zitat mit der Paper Id von der zitierten Publikation erhalten. Schlussendlich wurde daraus auch der Titel der Publikation erhalten. Nun muss nur noch der Titel in unserer Datenbank gefunden werden um die PublikationsId zu erhalten. Denn in unserer Relation, mit der die Zitierung dargestellt wird, werden die beiden Primärschlüssel der Publikationen benötigt. Es kann passieren das die Id von der Zitierten Publikation nicht in unserer Datenbank gefunden wird, da im Academic Graph mehr Werke gespeichert sind als in unserer und dadurch sich nicht alle Arbeiten überschneiden. Wurden die Daten gefunden, werden sie einfach mit einem DML-Befehl in die Datenbank eingefügt.

So wird die Datenbank Schritt für Schritt erweitert.

5 Ergebnis

In diesem Abschnitt der Arbeit geht es nicht mehr um Planung und Aufbau der Datenbank sonder darum das die Planung geklappt hat. Erst fassen wir die Schritte zusammen, die wir bisher für die Extraktion erledigt haben. Dann zeigen wir was diese alles gebracht hat und wie die Daten in der Datenbank aussieht. Danach machen wir das selbe mit der Erweiterung.

Zum Anfang haben wir eine uns mit Python eine Postgres Datenbank aufgebaut. Dies haben wir erst mit der Planung und dann mit der Ausführung bewältigt. Zu nächst haben wir uns die Daten angeguckt. Dann haben wir daraus ein ER-Modell erstellt und dies in ein relationales Schema gebracht. Dieses Schema haben wir dann verschmolzen um die letzte theoretische Übersicht der Datenbank aufzuzeigen. Darauf hin haben wir uns Anomalien angeguckt und Normalformen erklärt um diese zu verhindern. Danach haben wir uns die eigentliche Erstellung der Datenbank angeguckt und wie man mit SQL eine Tabelle einfügt. Nun haben wir die Daten nur noch eingefügt und abgespeichert.

Dieses Einfügen klappt mit einer Geschwindigkeit von 40-60 Transaktionen die Sekunde, dass heißt nur für grob 5 Millionen Publikationen würden wir 100.000 Sekunden brauchen das sind 1667 Minuten oder auch ca. 28 Stunden. Davon abgesehen das noch 2 Millionen Autoren da zu kommen. Dazu kommt noch das Transaktionen nicht nicht Publikationen pro Sekunde sind, denn in Publikationen sind alleine zwischen 2-4 Autoren beteiligt die jedes mal Abgefragt werden müssen ob sie Existieren und wenn nicht angelegt werden. Hier müssen natürlich auch die Relationen angelegt werden die die beiden Tabellen verbindet. Daraus werden dann pro Publikation dann mindestens 6 Transaktionen mehr. Dazu kommen noch die ein bis zwei elektronischen Versionen pro Publikation mit den Relationen. Somit landen wir grob bei einer Dauer von 308 Stunden das sind fasst 13 Tage. Da durch viele erneute Anpassungen der Datenbank dieser Prozess wiederholt werden musste, werden hier nur Ausschnitte der bisherigen Ergebnisse gezeigt.

Um die Daten auszulesen wird Data Querie Language(DQL) verwendet, eine Datenabfragesprache die wieder ein Teil von SQL da stellt. Hiermit können Anfragen an die Datenbank gestellt werden die uns dann dem entsprechende Antworten liefert. Mit diesem Teil von SQL kann in der Datenbank nichts geändert werden weder Struktur noch Daten.

```
SELECT Name , COUNT(Key) AS Anzahl FROM AUTOR
GROUP BY Name
ORDER BY COUNT(Key) DESC
LIMIT 3;
```

Abbildung 12: DQL Beispiel

Da viele Autoren in der Datenbank sind, gibt es einige Leute die den selben Namen haben. Mit diesem DQL Befehl werden die Top 3 Namen ausgegeben die sich am öftesten wiederholen. Dafür wird der Select-Befehl benutzt. Dazu wird erst angegeben welche Attribute ausgegeben werden soll und dann aus welcher Tabelle diese Ausgabe kommen soll. In unserem Fall ist es der Name und die gezählten Autoren aus der Tabelle Autor. Der Befehl AS ändert nur den Spalten Namen von COUNT(Key) auf Anzahl. Mit GROUP

BY gruppieren wir die Zeilen an Hand des Namens. Nun sortieren wir die Tabelle nach der Anzahl mit ORDER BY. Das DESC steht für descending also für eine absteigende Reihenfolge, damit wir das höchste Ergebnis als erstes bekommen. Durch das LIMIT 3 werden nur die ersten 3 Einträge ausgegeben. Das Ergebnis erhalten wir darauf hin in Tabellenform. Jede Zeile ist ein eigener Eintrag und enthält einen Datensatz.

Name	Anzahl
Wie Wang	19
Wei Li	13
LeiWang	12

Abbildung 13: DQL Ergebnis

Aus diesem Beispiel kann man entnehmen das der häufigste Name der Autoren Wie Wang ist. Hier bei ist zu beachten das zu der Zeit der Ausgabe 84,640 Autoren in der Datenbank waren. Davon hießen Maximal 19 Leute Gleich. Diese Zahl kann noch variieren da 2,6 Millionen Autoren insgesamt in der DBLP sind.

Nun folgen die Resultate der Erweiterung. Wo erst die API besprochen wurde und dann ein typischer Durchlauf um ein Zitate Einzufügen.

In der Arbeit wird gesagt das die Transaktion mit der API auf 10.000 beschränkt ist, dennoch benutzen wir nur 200 Titel gleichzeitig zum Testen. Dies liegt daran das pro angefragten Titel alle gefunden Zitate nochmal angefragt werden müssen. Da es keine Begrenzung gibt wie viele Zitate vorhanden sein können, kann die Anzahl an Anfragen, die für die Zitate gebraucht werden, nicht eingeschätzt werden. Deshalb werden nur 200 Titel genutzt.

Hat Zitat Id	Ist Zitiert Id	Zitat
164	220	'2In order to prevent the system from…

Abbildung 14: Zitat Ausschnitt

In der Abbildung 14 sieht man die Relation Publikation-hat-Zitat. Der erste Wert repräsentiert die Publikations Id von der Arbeit die das Zitat verwendet. Der Zweite stellt die Id von dem Werk da das zitiert wurde. Die letzte Spalte zeigt das Zitat an sich. Dieses wurde gekürzt, da dies sonst zu viel platz ein nehmen würde. Dieser Ausschnitt wurde direkt aus der Datenbank geholt und beweist das Zitate eingefügt werden

6 Fazit & Ausblick

In diesem Teil werden erst die Erkenntnisse gezeigt, die nach der Arbeit durch Probleme, Herausforderungen und Ergebnisse erschlossen wurden. Danach wird ein Ausblick und Ideen auf zukünftige Projekte in der Richtung gegeben.

Mit dieser Arbeit sollte gezeigt werden, ob es möglich ist, mehrere Datenquellen im Knowledge Graph Format in einer Datenbank zu vereinen.

Durch ein strukturiert aufgebautes Datenbankdesign, was die Grundlage darstellt, und einen Parser, der die Datenbank befüllt, konnte die erste Datenbank extrahiert werden. Die Performance des Parsers ist leider nicht optimal, weswegen zu Beginn mit mehr Zeit geplant werden muss. Zudem sollten die Datenmengen nicht unterschätzen werden, denn oftmals verbirgt sich mehr Aufwand als vorher gedacht, wie bspw., dass für fünf Millionen Publikationen nicht genau dieselbe Anzahl an Transaktionen zur Datenbank braucht werden. Hier zeigt sich auch das Zeitmanagement, denn 13 Tage Durchlaufdauer müssen zukünftig besser in die Planung einfließen. Dazu kommt noch ein großes Problem, welches durch die sehr wenig beschriebene Datenstruktur auftritt, die die Daten beim Suchen nicht immer alle findet. Für ‚masterthesis' und ‚phdthesis' wurden erst sehr spät entsprechende Beispieldatensätze gefunden, um deren Attribute zu bestimmen. Deshalb wurden diese auch erst relativ spät in die Datenbank aufgenommen.

Anschließend wurden die zusätzlichen Zitate hinzugefügt und damit schlussendlich die beiden Datenquellen vereint. Hier stellt sich nur die Frage, ob die Publikationen in beiden Graphen dieselben sind, denn es könnte sein, dass zwei Arbeiten den gleichen Titel besitzen, im gleichen Jahr geschrieben und von den selben Autoren geschrieben wurden. Ein Titel ist nicht eindeutig, deshalb könnte es passieren, dass verschiedene Werke als gleich angesehen werden, es aber nicht sind.

Zusammenfassend wurde ein Parser erstellt, der die Daten aus einem Knowledge Graphen holt. Aus diesen Daten wurde dann eine Datenbank, die mit der Hilfe von Datenmodellierungstechniken entworfen und die Daten gespeichert wurden. Daraufhin wurde mit einer API die Daten aus einer weiteren Datenquelle geholt. Diese wurden dann an die Datenbank angepasst und hinzugefügt. Anschließend wurden noch Testausgaben aus der Datenbank vorgestellt.

Da es nicht sicher ist ob, sich Publikationen eindeutig über den Titel identifizieren, ist es wahrscheinlich sehr vom Vorteil, wenn eine Identifikationsnummer für Arbeiten erstellt wird. Mit dieser Nummer könnten Werke leichter in verschiedenen Datenquellen gefunden werden. Diese Idee gibt es schon für Autoren, da diese auch keine spezielle Identifikation haben. In diesem Fall würde für Autoren die Nummer heißen: ORCID ID. Leider ist diese noch nicht komplett durchgesetzt, aber es würde die Identifikation um ein vielfaches vereinfachen.

Literatur

Microsoft Research APIs (o.j.). Academic Search API. URL: https://msr-apis.portal.azure-api.net/docs/services/academic-search-api/ (besucht am 19. 10. 2020).

Michael Färber (2019). „The Microsoft Academic Knowledge Graph: A Linked Data Source with 8 Billion Triples of Scholarly Data“. In: Proceedings of the 18th International Semantic Web Conference. ISWC'19, S. 113–129.

Michael Ley (2009). „DBLP - Some Lessons Learned“. In: Proc. VLDB Endow. 2.2, S. 1493–1500.

Microsoft (o.j.). Project Academic Knowledge (PAK) documentation. URL: https://docs.microsoft.com/en-us/academic-services/project-academic-knowledge/ (besucht am 19. 10. 2020).

ontotext (o.j.). What is a Knowledge Graph? URL: https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/ (besucht am 16. 10. 2020).

Python (2020). Python FAQ. URL: https://docs.python.org/3/faq/general.html#what-is-python (besucht am 16. 10. 2020).

Wikipedia (o.j.[a]). Graph(Graphentheorie). URL: https://de.wikipedia.org/wiki/Graph_(Graphentheorie) (besucht am 17. 10. 2020).

Wikipedia (o.j.[b]). JSON. URL: https://en.wikipedia.org/wiki/JSON (besucht am 17. 10. 2020).

Abbildungsverzeichnis

Tabellenverzeichnis