

Graphentechnologien in den Digitalen Geisteswissenschaften

Andreas Kuczera

Inhaltsverzeichnis

1 Einleitung	4
1.1 Mit der Vermittlung der Konzepte von Graphen und Graphdatenbanken werden Kompetenzen in den Bereichen Modellierung (was hängt wie zusammen), Quellenkritik (welche Qualität haben die Informationen in Knoten und Kanten) und der Verknüpfung von wissenschaftlicher Fragestellung und ihrer digitalen Modellierung geschärft. Daher vermittelt dieses Studienbuch dem Leser wichtiges Rüstzeug für die zunehmende Digitalisierung immer größerer, auch die Geisteswissenschaften umfassender, Bereiche der Gesellschaft.	5
2 Inhalt	5
3 Was ist eine Graphdatenbank	6
3.1 Knoten und Kanten können also noch zusätzliche Eigenschaften besitzen, in denen weitere Informationen gespeichert sind. Diese Eigenschaften sind spezifisch für die jeweiligen Knotentypen. So sieht man in der Abbildung, dass die beiden Knoten vom Typ Person jeweils noch die Eigenschaft Namen haben, deren Wert dann die Namen der Person angibt, während der Knoten vom Typ Buch die Eigenschaft Titel trägt, in dem der Titel des Buches abgespeichert wird.	6
4 Inhalt	7
5 Das Projekt Regesta Imperii oder “Wie suchen Onlinenutzer Regesten”	7
5.1 Das Projekt Regesta Imperii	7
5.2 Die Digitalisierung der Regesta Imperii	8
5.3 Wie suchen Online-Nutzer Regesten ?	8
5.4 Historische Netzwerkanalyse in den Registern	10

6 Inhalt	13
7 Wie kommen die Regesten in den Graphen ?	13
7.1 Import mit dem LOAD CSV-Befehl	16
7.2 Google-Docs für den CSV-Download	16
7.3 Die neo4j-Eingabezeile	19
8 Erstellen der Ausstellungsorte	26
8.0.1 Herrscherhandeln in den Regesta Imperii	28
8.0.2 Zitationsnetzwerke in den Regesta Imperii	28
9 Import der Registerdaten in die Graphdatenbank	29
9.1 Vorbereitung der Registerdaten	29
9.2 Import der Registerdaten in die Graphdatenbank	31
9.3 Die Hierarchie des Registers der Regesten Kaiser Friedrichs III. .	32
10 Auswertungsperspektiven	33
10.1 Personennetzwerke in den Registern	33
10.1.1 Graf Robert II. von Flandern in seinem Netzwerk	33
10.2 Herrscherhandeln ausgezählt	36
10.3 Herrscherhandeln pro Ausstellungsort ausgezählt	36
10.4 Welche Literatur wird am meisten zitiert	37
10.5 Herrscherhandeln und Anwesenheit	37
10.6 Regesten 200 km rund um Augsburg	37
10.7 [^f663]: Die nun folgenden Abfragen sind zum Teil einer Präsentation entnommen, die für die Summerschool der Digitalen Akademie im Rahmen des Mainzed entwickelt wurden. Die Präsentation findet sich unter der URL https://digitale-methodik.adwmainz.net/mod5/5c/slides/graphentechnologien/RI.html	37
11 Inhalt	37
12 Import von strukturierten XML-Daten in neo4j	38
12.1 Das XML-Beispiel	38
12.2 Knotentypen	39
12.3 Kantentypen	39
12.4 Der Import mit apoc.load.xmlSimple	42
13 Inhalt	42
14 Das DTA im Graphen	42
15 Die Downloadformate des DTA	43
16 Vorbereitungen	43
17 Import	43

17.1 Tokenimport	43
17.2 Satzstrukturen	44
17.3 Lemmainport	44
17.4 Neo4j-Browser	45
17.5 Beispielabfrage	45
18 Zusammenfassung	45
18.1 Im vorliegenden Kapitel wurden die Schritte für den Import der DTA-TCF-Fassung von Goethes Faust in die Graphdatenbank neo4j vorgestellt. Die qualitativ hochwertigen Text-Quellen des Deutschen Textarchivs bieten in Verbindung mit Graphdatenbanken sehr interessante neue Möglichkeiten zur Auswertung der Texte. Durch Austausch des Links zur TCF-Fassung können auch andere Texte des DTA eingespielt werden.	45
19 Inhalt	45
20 Textmodelle im Graphen	46
20.1 Text als Graph	46
20.1.1 XML-Dateien als Ketten von Zeichen	46
20.1.2 Modellierungsüberlegungen	46
20.2 Technische Vorbemerkungen	47
20.2.1 Die Graphdatenbank neo4j	47
20.2.2 Der neo4j-XML-Importer	48
21 Das DTA-Basisformat im Graphen	52
21.1 Strukturen des Dokuments	52
21.1.1 Graphenmodellierung von Zeilen	52
21.1.2 Zeilenwechsel mit Worttrennungen	53
21.1.3 Seitenzahlen und Faksimilezählung	56
21.1.4 Absätze	60
21.1.5 Kapiteleinteilung	61
21.1.6 Zusammenfassung	62
21.2 Editorische Eingriffe	64
21.2.1 Hinzufügungen und Tilgungen	64
21.2.2 <choice>-Element	68
21.2.3 <sic> und <corr>-Elemente	68
22 Anhang	68
22.1 cypher-Befehle für den Import der Mitschrift von Patzig	68
22.2 Liste aller im Patzig-Manuskript vorkommenden Elemente sortiert nach Häufigkeit	68
22.3 Weitere Texte	71
22.3.1 Dokument vorbereiten	71
22.4 [^148e]: Vgl. http://www.deutsches-textarchiv.de/book/view/patzig_msgermfol841842_1828/?hl=zun	
23 Inhalt	72

24 Mehrere Werte in einem CSV-Feld importieren	72
25 MERGE schlägt fehl da eine Property NULL ist	73
26 Knoten hat bestimmte Kante nicht	74
27 Häufigkeit von Wortketten	74
28 Der WITH-Befehl	74
29 Die Apoc-Bibliothek	75
29.1 Installation in neo4j	75
29.2 Installation unter neo4j-Desktop	75
29.3 Liste aller Funktionen	75
29.4 Dokumentation aller Funktionen	76
29.5 [^03a5]: Hierzu vgl. https://de.wikipedia.org/wiki/Deklarative_Programmierung zuletzt abgerufen am 12.6.2018.	76
30 Inhalt	76
30.1 Graphentechnologien in den Digitalen Geisteswissenschaften von Andreas Kuczera ist lizenziert unter einer Creative Commons Attribution-ShareAlike 4.0 International Lizenz.	76

1 Einleitung

Graphentechnologien sind hervorragend für die Modellierung, Speicherung und Analyse hochvernetzter Daten geeignet. Obgleich als Konzept schon länger etabliert erlebten sie mit dem Aufkommen des Internets und der Social-Media-Welle einen Aufschwung. Gegenüber relationalen Datenbankmodellen, bei denen die Daten in miteinander verknüpften Tabellen gespeichert werden, sind die Informationen in Graphdatenbanken in Knoten und Kanten modelliert und auf Specherebene auch genau so abgelegt. Mit diesem, einer Mind-Map sehr ähnlichen Modell lassen sich Forschungsdaten und Forschungsfragestellungen in einer Weise modellieren, die dem menschlichen Denken oft sehr nahe kommt.

Gerade in den digitalen Geisteswissenschaften gelingt es mit dem Graphenmodell bei der Modellierung und Strukturierung von Forschungsdaten und Forschungsfragestellung die Kluft zwischen Informatiker und Geisteswissenschaftler zu schließen, da der Graph eine für beide Seiten verständliche Plattform bietet. Für den Informatiker ist er hinreichend genau und berechenbar und für den Geisteswissenschaftler wegen seiner Schema- und Hierarchiefreiheit ausreichend flexibel. Gerade diese Eigenschaften, mit denen sich die beiden zentralen Zweige der Digitalen Geisteswissenschaften vereinen lassen, machen Graphen zu einem Schlüsselkonzept der Geisteswissenschaften des 21 Jahrhunderts.

Das vorliegende Buch ist als Studienbuch konzipiert und richtet sich an Studierende, Lehrende und Wissenschaftler gleichermaßen.

Zu Beginn wird anhand einfacher Beispiele in die Grundlagen der Graphentechnologie eingeführt und die Verwendung von Graphdatenbanken erklärt.

Im nächsten Abschnitt werden Beispiele für die Modellierung und den Import bereits vorhandener Forschungsdaten aus den Projekten Nomen-et-Gens und den Regesta Imperii in eine Graphdatenbank vorgestellt. Anschließend wird auf ein aktuelles Forschungsvorhaben aus dem Bereich Architektur/Geschichtswissenschaften eingegangen, in dessen Verlauf Graphen für die Modellierung der Forschungsdaten und deren anschließende Analyse verwendet wurden.

Im vierten Abschnitt wird der Import von XML-Daten in eine Graphdatenbank anhand von Beispieldaten aus der Epidat-Datenbank des Steinheim-Instituts in Essen erklärt. In einem zweiten Schritt werden die in verschiedenen XML-Varianten vorliegenden Textdaten des Deutschen Textarchivs im Graph modelliert.

Schließlich folgen im Anhang Konzepte zum Graph-Refactoring, zum projektspezifischen Datenimport und zur Struktur von Cypher-Queries.

1.1 Mit der Vermittlung der Konzepte von Graphen und Graphdatenbanken werden Kompetenzen in den Bereichen Modellierung (was hängt wie zusammen), Quellenkritik (welche Qualität haben die Informationen in Knoten und Kanten) und der Verknüpfung von wissenschaftlicher Fragestellung und ihrer digitalen Modellierung geschärft. Daher vermittelt dieses Studienbuch dem Leser wichtiges Rüstzeug für die zunehmende Digitalisierung immer größerer, auch die Geisteswissenschaften umfassender, Bereiche der Gesellschaft.

title: Was ist eine Graphdatenbank layout: default order: 5 contents: true —

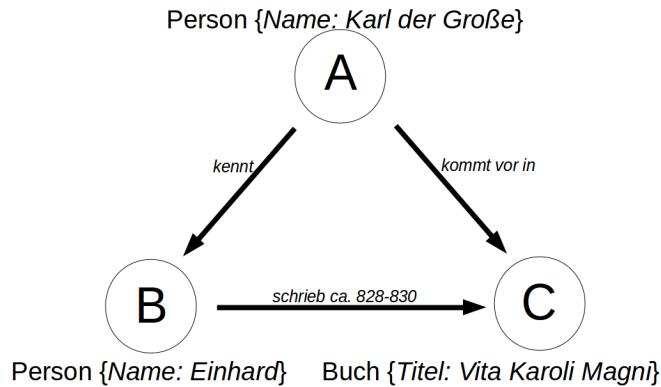
2 Inhalt

{:no_toc}

- Will be replaced with the ToC, excluding the “Contents” header {:toc}

3 Was ist eine Graphdatenbank

In den seit den 70er Jahren etablierten, relationalen Datenbanken werden die Daten in Tabellen abgespeichert, die untereinander über Schlüssel oder Schlüsseltabellen verknüpft sind. In Graphdatenbanken werden die Daten dagegen in Knoten und Kanten gespeichert.



ten und Kanten gespeichert.

Der zeigt oben einen Knoten (engl. Nodes) vom Typ Person mit der Eigenschaft Name. Diese hat den Wert "Karl der Große". Links unten ist ein weiterer Knoten vom Typ Person mit dem Namen "Einhard". Rechts unten ist ein Knoten vom Typ Buch und dem Titel "Vita Karoli Magni" abgebildet. Die Kanten (engl. Edges) geben an, dass Karl der Große Einhart kannte, Einhard ca. 828-830 das Buch "Vita Karoli Magni" schrieb und Karl der Große in dem Buch vorkommt.

3.1 Knoten und Kanten können also noch zusätzliche Eigenschaften besitzen, in denen weitere Informationen gespeichert sind. Diese Eigenschaften sind spezifisch für die jeweiligen Knotentypen. So sieht man in der Abbildung, dass die beiden Knoten vom Typ Person jeweils noch die Eigenschaft Namen haben, deren Wert dann die Namen der Person angibt, während der Knoten vom Typ Buch die Eigenschaft Titel trägt, in dem der Titel des Buches abgespeichert wird.

title: Das Projekt Regesta Imperii oder "Wie suchen Onlinenutzer Regesten"
layout: default order: 10 contents: true —

4 Inhalt

{::no_toc}

- Will be replaced with the ToC, excluding the “Contents” header {:toc}

5 Das Projekt Regesta Imperii oder “Wie suchen Onlinenutzer Regesten”

5.1 Das Projekt Regesta Imperii

Das Projekt Regesta Imperii wurde von Johann-Friedrich Böhmer im Jahr 1829 begonnen. Ursprünglich als Vorarbeit zu den Monumenta Germaniae Historica gedacht wurde es mit einem erweitereten Regestenkonzept bald zu einem unverzichtbaren Grundlagenwerk. In den Regesta Imperii werden Inhaltsangaben von Urkunden erstellt, die rechtlich relevante Personen, Inhalte, Orte und Sachverhalte in deutscher Sprache zusammenfassen. Zeitlich umfassen sie den Rahmen von den Karolingern (7. Jahrhundert) bis Kaiser Maximilian (gestorben 1519).

Beispielbild Urkunden

Ursprünglich von der DFG gefördert sind die Regesta Imperii heute Teil des Bund-Ländergeförderten Akademienprogramms und werden von der Akademie der Wissenschaften und der Literatur, Mainz, der Berlin-Brandenburgischen Akademie der Wissenschaften und der Akademie der Wissenschaften, Wien betreut.

Die Regesta Imperii arbeiten vor allem herrscherzentriert, d.h. in den Regesten muss der Herrscher eine zentrale Rolle spielen. Bei Urkundenregesten hat er selbst die Urkunde ausgestellt, bei historiographischen Regesten werden den Herrscher betreffende historische Hintergründe zusammengefasst.

Beispielbild Regest

In der Kopfzeile des Regests werden der Herrscher sowie Abteilung, Band und Regestennummer genannt. Die darunterliegende Datierungszeile nennt das Ausstellungsdatum der Urkunden und den Handlungs- bzw. Ausstellungsort. Es folgt der Regestentext mit den mit der Zusammenfassung der Urkunde, Hinweise zur Originaldatierung, die Kanzleivermerke und schließlich Angaben zur Überlieferungssituation (Gibt es eine Originalurkunden, wo liegt sie, gibt es ggf. Abschriften etc.).

5.2 Die Digitalisierung der Regesta Imperii

Im Rahmen eines von der DFG geförderten Projekts wurden die Regesta Imperii gemeinsame von der Akademie der Wissenschaften, Mainz und der Bayrischen Staatsbibliothek München von 2001 bis 2006 komplett digitalisiert. Alle seit 2006 erschienenen Regesten wurden sofort im Volltext online gestellt. Glücklicherweise hatte die Mainzer Akademie die Rechte selbst inne, so dass der Veröffentlichung als Volltext im Internet keine rechtlichen Hürden im Wege standen. Rückblickend lässt sich feststellen, dass der Absatz der gedruckten Bände nicht gelitten sondern teilweise sogar etwas zugelegt hat.

5.3 Wie suchen Online-Nutzer Regesten ?

Ende 2013 stellten Torsten Schrade und ich auf der Digital-Diplomatics-Konferenz in Paris eine Untersuchung vor, in der wir das Suchverhalten der Nutzer der Online-Regestensuche untersucht haben.¹ Ein interessantes Ergebnis war die Häufigkeitsverteilung der Treffermengen pro Suchanfrage.

Im Tortendiagramm ist die Treffermenge in Zehnerschritten angegeben. die hellgraue Gruppe oben rechts hat keine Treffen, die dunkelgraue Gruppe einen bis zehn Treffer, die gelbe Gruppe 11 bis 20 usw. Die lila Gruppe hat mehr als hundert Treffer. Was uns im Projekt überrascht hat, war die große Gruppe mit über 100 Treffern. Hinzu kam, dass über 68% der Nutzer nur ein Suchwort in die Suchmaske eingegeben haben, wobei das beliebteste Suchwort *Heinrich* Ende 2013 über 18.000 Treffer erzielte. Auf der Ergebnisseite hieß es dann lapidar: “Sie suchten nach *Heinrich*. Ihre Suche erzielte 18884 Treffer [...] Sie sehen die Treffer 1 bis 20.”

Zusammenfassend kamen wir zu dem Ergebnis, dass die Gruppe mit 1 bis 10 Treffern zufrieden mit ihrem Ergebnis war. 10 Regesten lassen sich gut ausdrucken und können anschließend gelesen, ausgewertet und in die eigene historische Arbeit integriert werden. Die Gruppe mit keinem Treffer hatte möglicherweise die Suche zu sehr eingeschränkt oder einen Tippfehler beim Suchbegriff und wäre lieber in der Gruppe mit einem bis 10 Treffern. Auch 20 Treffer lassen sich auf analoge Weise noch gut verarbeiten aber insgesamt gingen wir davon aus, dass die Nutzer aus den Gruppen von 11 bis 100 Treffern auch lieber ein kleineres Ergebnis bevorzugt hätten.

Sehr gut lässt sich am Tortendiagramm ablesen, dass über die Hälfte unserer Nutzer vor der Suche eine genaue Vorstellung vom Ergebnis haben. Sie sind CIN-Nutzer (concrete information need). Die Gruppe mit über 100 Treffern können der Gruppe der POIN-Nutzer (problem-oriented information need) zugeordnet

¹Vgl. Kuczera, Andreas; Schrade, Torsten: From Charter Data to Charter Presentation: Thinking about Web Usability in the Regesta Imperii Online. Vortrag auf der Tagung ›Digital Diplomatics 2013 – What is Diplomatics in the Digital Environment?‹ Folien: <https://prezi.com/vvacmdndthqg/from-charta-data-to-charta-presentation/>.

Distribution of Result Set Sizes

Result set sizes returned for search queries between November 2012 and October 2013 (101220 queries).

- 0 Hits
- 0 - 10 Hits
- 10 - 20 Hits
- 20 - 30 Hits
- 40 - 50 Hits
- 50 - 60 Hits
- 60 - 70 Hits
- 70 - 80 Hits
- 80 - 90 Hits
- 90 - 100 Hits
- > 100 Hits

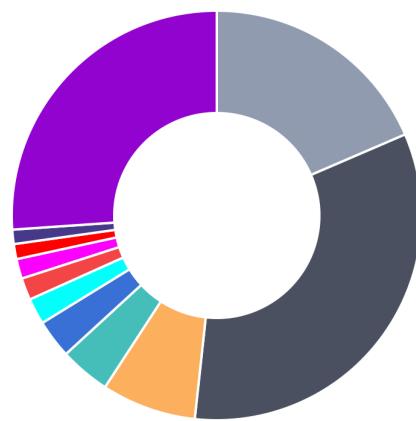


Abbildung 1: Treffermengen pro Suchanfragen im Jahr 2013.

werden, die problemorientierte Anfragen haben. Für diese Nutzergruppe ist die aktuelle Trefferanzeige der Regestensuche nur unzureichend, das sie für ihre großen Treffermengen weitere Einschränkungsmöglichkeiten brauchen.²

5.4 Historische Netzwerkanalyse in den Registern

Im Bereich der historischen Netzwerkanalyse gab es in den letzten Jahren sehr interessante Arbeiten.³ von Seiten der Regesta Imperii bieten sich hier vor allem die Register der Regesta Imperii als sehr interessante Quelle an. Geht man davon aus, dass alle Personen, die gemeinsam in einem Regest genannt sind, etwas miteinander zu tun haben, könnte man auf Grundlage der Registerdaten ein Personennetzwerk erstellen. Über die Qualität der Beziehungen lässt sich nichts sagen und dies schränkt die Aussage der Daten ein. Andererseits stehen wiederum sehr viele Verknüpfungen zur Verfügung.

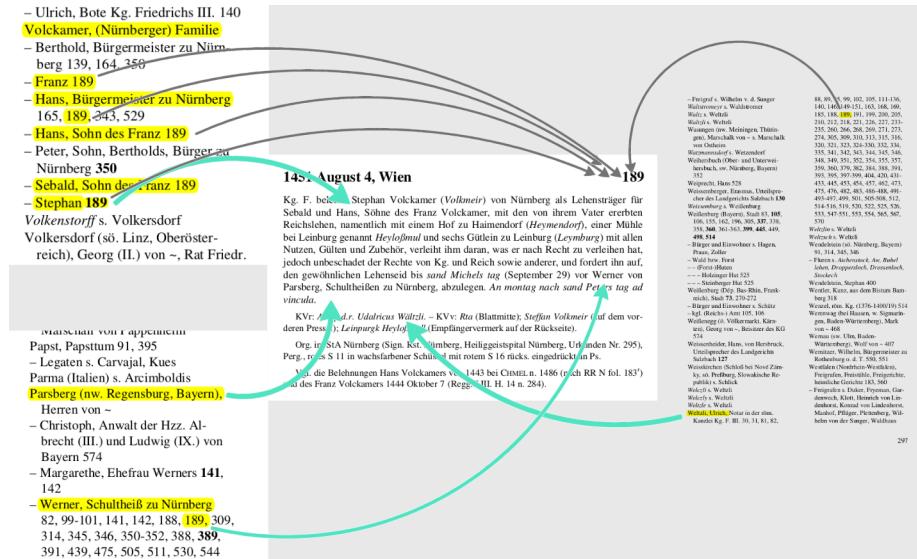


Abbildung 2: Registereinträge im Regest als Grundlage für ein Personennetzwerk.

Allein die Einträge in den Registen der Regesten Kaiser Friedrichs III. sind über 143.000 mal in Regesten genannt. Daraus ergeben sich dann über 460.000

²Näheres dazu in Kuczera, Andreas: Digitale Perspektiven mediävistischer Quellenrecherche, in: Mittelalter. Interdisziplinäre Forschung und Rezeptionsgeschichte, 18.04.2014. URL: mittelalter.hypotheses.org/3492.

³Vgl. beispielsweise Gramsch, Robert: Das Reich als Netzwerk der Fürsten - Politische Strukturen unter dem Doppelkönigtum Friedrichs II. und Heinrichs (VII.) 1225-1235. Ostfildern, 2013. Einen guten Überblick bietet das Handbuch Historische Netzwerkforschung - Grundlagen und Anwendungen. Herausgegeben von Marten Düring, Ulrich Eumann, Martin Stark und Linda von Keyserlingk. Berlin 2016.

1zu1-Beziehungen.⁴

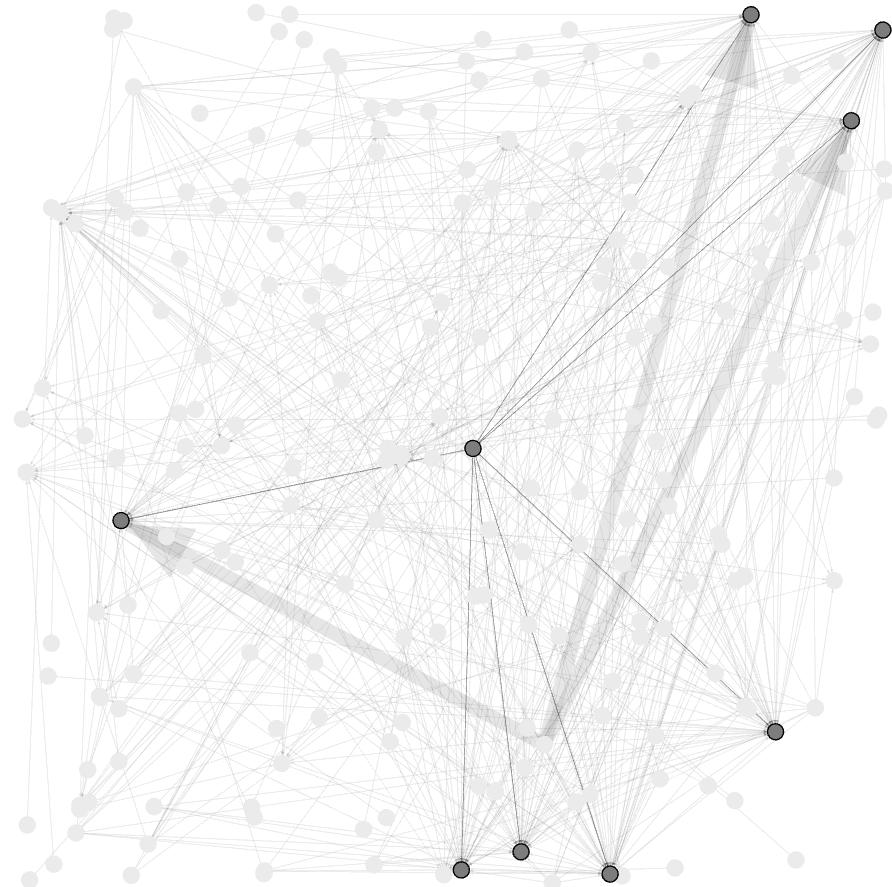


Abbildung 3: Ausschnitt der 1zu1-Beziehungen in Gephi.

In der folgenden Abbildung sind die den Registern des Regestenbandes von Joseph Chmel gewonnenen 1zu1-Beziehungen mit Gephi visualisiert.⁵

⁴Der cypher-Befehl zur Erstellung der 1zu1-Beziehungen lautet: `MATCH (n1:Registereintrag)-[:APPEARS_IN]->(r:Regest)<-[::APPEARS_IN]-(n2:Registereintrag) MERGE (n1)-[:KNOWS]->(n2);` Dabei werden die gerichteten KNOWS-Kanten jeweils in beide Richtungen erstellt. Mit folgendem Befehl lassen sich die KNOWS-Kanten zählen: `MATCH p=()-[r:KNOWS]->() RETURN count(p);` Für die Bestimmung der 1zu1-Beziehungen muss der Wert noch durch 2 geteilt werden.

⁵Regesta chronologico-diplomatica Friderici III. Romanorum imperatoris (regis IV.) : Auszug aus den im K.K. Geheimen Haus-, Hof- und Staats-Archive zu Wien sich befindenden Registraturbüchern vom Jahre 1440 - 1493 ; nebst Auszügen aus Original-Urkunden, Manuscripten und Büchern / von Joseph Chmel, Wien 1838 und 1840.

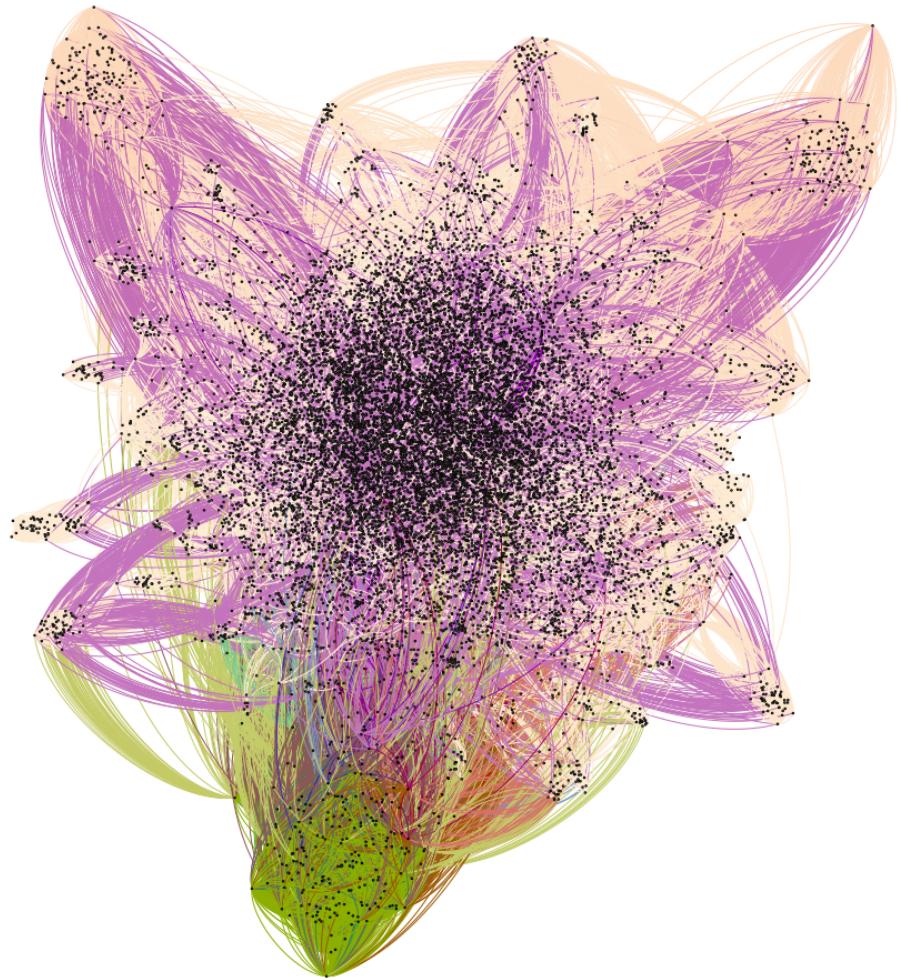


Abbildung 4: Personennetzwerk aus den Registern der Regesten Chmels.

Bei der Analyse ergaben sich aber verschiedene Probleme. Zum einen werden in den Registern auch Kanzleibeamte genannt, die mit der eigentlichen Regestenhandlung garnichts zu tun hatten sondern lediglich später ihr Kürzel auf der Urkunde hinterließen. Dies macht archivgeschichtlich interessant sein, für die Regestenhandlung ist es aber irrelevant. Ein zweites Problem ist der Aufbau des Registers, in dem Orte und Personen in einem Register zusammengefasst werden. Zum einen handelt es sich hierdurch nicht mehr um ein reines Personennetzwerk sondern um ein gemischtes Personen- und Ortsnetzwerden und zum anderen überragen die über sehr lange Zeit bestehenden Orte die in ihrer Lebensdauer begrenzten natürlichen Personen in den Netzwerkstrukturen.

Aus Historikersicht war der Ansatz also weniger zielführend jedoch ergaben sich aus Modellierungssicht interessante Einblicke. Um die Netzwerke näher analysieren zu können, untersuchten wir kürze Zeitschnitte der Regesten. Hierfür musste das in Java geschriebene Programm zur Erstellung der Netzwerkdaten jedesmal umgeschrieben werden. Mein Kollege Ulli Meybohm, der das Programm damals betreute wies mich nach dem wiederholgen Umschreiben des Programms darauf hin, dass ich für meine Daten besser eine Graphdatenbank verwenden solle, beispielsweise neo4j. Erste Versuche des Imports der Registerdaten in neo4j erwiesen sich aber als sehr komplex, obwohl das Datenmodell *Person kennt Person* eigentlich relativ einfach ist.

Schließlich ergaben Nachfragen bei neo4j, dass bei Problemen mit dem Datenmodell oft einfach ein Typ von Knoten vergessen worden sein könnte. Und tatsächlich hatten wir die Regestenknoten nicht bedacht. Mit den Regestenknoten im Modell war der Import schließlich mit weniger rechnerischem Aufwand möglich.

6 Inhalt

{::no_toc}

- Will be replaced with the ToC, excluding the “Contents” header {:toc}

7 Wie kommen die Regesten in den Graphen ?

Die Regesta Imperii Online basieren momentan auf dem Content-Management-System Typo3 welches auf eine mysql-Datenbank aufbaut. In der Datenbank werden die Regesteninformationen in verschiedenen Tabellen vorgehalten. Die Webseite bietet zum einen die Möglichkeit, die Regesten über eine REST-Schnittstelle im CEI-XML-Format oder als CSV-Dateien herunterzuladen. Für den Import in die Graphdatenbank bietet sich das CSV-Format an.

In der CSV-Datei finden sich die oben erläuterten einzelnen Elemente der Regesten in jeweils eigenen Spalten. Die Spaltenüberschrift gibt Auskunft zum Inhalt

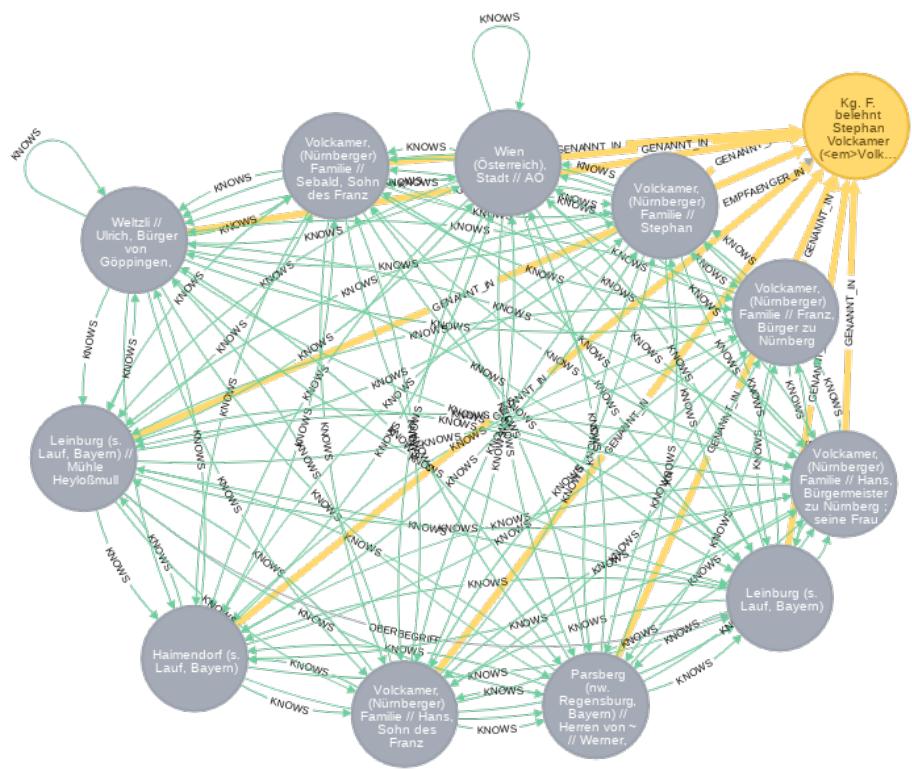


Abbildung 5: Regest und Registereinträge mit GENANNT_IN-Kanten und den KNOWS-Kanten.



Abbildung 6: Graphmodell ohne KNOWS-Kanten. Diese können bei Bedarf einfach errechnet werden.

The screenshot shows a CSV file titled 'Regesten' in a spreadsheet application. The file contains 27 rows of data, each representing a regestrum entry. The columns are labeled A through L. Column A is 'uid', B is 'start_date', C is 'end_date', D is 'identifier', E is 'regnu', F is 'persistent_id', G is 'title', H is 'date_string', I is 'locality_strin', J is 'summary', and K is 'archival_history'. The data entries are mostly dates and names, such as '1050-11-11 Heinrich IV.', '1050 Novem (Goslar?)', and 'Heinrich wirc Herim. Aug. 1050'. The last column, 'archival_history', contains URLs starting with 'http://opac.re'.

uid	A	B	C	D	E	F	G	H	I	J	K	L
1	2	3	4	5	6	7	8	9	10	11	12	13
20053	1050-11-11	1050-11-11	RI III,2,3 n. 1		1	1050-11-11_ Heinrich IV.	1050 Novem (Goslar?)		Heinrich wirc Herim. Aug. 1050			
20054	1050-12-25	1050-12-25	RI III,2,3 n. 2		2	1050-12-25_ Heinrich IV.	1050 Dezem Pöhlde		Heinrichs Va Herim. Aug. 1051			
20055	1051-02-02	1051-02-02	RI III,2,3 n. 3		3	1051-02-02_ Heinrich IV.	1051 Februar Augsburg		Heinrich folg Herim. Aug. 1051			
20056	1051-03-04	1051-03-04	RI III,2,3 n. 4		4	1051-03-04_ Heinrich IV.	1051 März (Speyer		Heinrich folg Herim. Aug. 1051			
20057	1051-03-31	1051-03-31	RI III,2,3 n. 5		5	1051-03-31_ Heinrich IV.	1051 März 3 Köln		Heinrich emj Herim. Aug. 1051			
20058	1051-11-12	1051-11-12	RI III,2,3 n. 6		6	1051-11-12_ Heinrich IV.	1051 Novem Regensburg	Während des Ungarnfeldzuges				
20059	1051-12-25	1051-12-25	RI III,2,3 n. 7		7	1051-12-25_ Heinrich IV.	1051 Dezem Goslar	Weihnachtsf Herim. Aug. 1052				
20060	1052-03-05	1052-03-05	RI III,2,3 n. 8		8	1052-03-05_ Heinrich IV.	1052 März 5 Kaiserswert	Heinrich inte <link http://opac.re				
20061	1052-03-27	1052-03-27	RI III,2,3 n. 9		9	1052-03-27_ Heinrich IV.	1052 März 2 Goslar	Heinrichs wii <link http://opac.re				
20062	1052-06-07	1052-06-07	RI III,2,3 n. 10		10	1052-06-07_ Heinrich IV.	1052 Juni (7 Zürich	Heinrich folg Herim. Aug. 1052				
20063	1052-09-01	1052-09-30	RI III,2,3 n. 11		11	1052-09-00_ Heinrich IV.	1052 (Septem –	Heinrichs Elt Herim. Aug. 1052				
20064	1052-12-25	1052-12-25	RI III,2,3 n. 12		12	1052-12-25_ Heinrich IV.	1052 Dezem Worms	Weihnachtsf Herim. Aug. 1053				
20065	1053-11-03	1053-11-03	RI III,2,3 n. 13		13	1053-11-03_ Heinrich IV.	1053 (vor de Tribur	Heinrich wirc Herim. Aug. 1053				
20066	1053-12-25	1053-12-25	RI III,2,3 n. 14		14	1053-12-25_ Heinrich IV.	1053 Dezem Altötting	Heinrich erh: Herim. Aug. 1053				
20067	1054-04-03	1054-04-03	RI III,2,3 n. 15		15	1054-04-03_ Heinrich IV.	1054 April (1 Mainz	Heinrich folg Herim. Aug. 1054				
20068	1054-05-29	1054-05-29	RI III,2,3 n. 16		16	1054-05-29_ Heinrich IV.	1054 Mai 29 (Goslar?)	Heinrich inte <link http://opac.re				
20069	1054-05-31	1054-05-31	RI III,2,3 n. 17		17	1054-05-31_ Heinrich IV.	1054 Mai 31 Goslar	Heinrich inte <link http://opac.re				
20070	1054-07-17	1054-07-17	RI III,2,3 n. 18		18	1054-07-17_ Heinrich IV.	1054 Juli 17 Aachen	Heinrich wirc Lampert 1054 (<lir				
20071	1054-07-17	1054-07-17	RI III,2,3 n. 19		19	1054-07-17_ Heinrich IV.	1054 (nach c (Aachen?)	Vermutlich anlässlich Heinrichs i				
20072	1054-11-17	1054-11-17	RI III,2,3 n. 20		20	1054-11-17_ Heinrich IV.	1054 Novem Mainz	Heinrich inte <link http://opac.re				
20073	1054-11-17	1054-11-17	RI III,2,3 n. 21		21	1054-11-17_ Heinrich IV.	1054 Novem Mainz	Heinrichs wii <link http://opac.re				
20074	1054-12-25	1054-12-25	RI III,2,3 n. 22		22	1054-12-25_ Heinrich IV.	1054 Dezem Goslar	Weihnachtsf Ann. Altah. 1055 (
20075	1055-01-16	1055-01-16	RI III,2,3 n. 23		23	1055-01-16_ Heinrich IV.	1055 Januar Quedlinburg	Heinrichs wii <link http://opac.re				
20076	1055-03-03	1055-03-03	RI III,2,3 n. 24		24	1055-03-03_ Heinrich IV.	1055 März 3 Regensburg	Heinrich inte <link http://opac.re				
20077	1055-03-06	1055-03-06	RI III,2,3 n. 25		25	1055-03-06_ Heinrich IV.	1055 März 6 Regensburg	Heinrichs wii <link http://opac.re				
20078	1055-03-12	1055-03-12	RI III,2,3 n. 26		26	1055-03-12_ Heinrich IV.	1055 März 6 „Utingen“ (?)	Heinrichs wii <link http://opac.re				

Abbildung 7: Regesten als CSV-Datei

der jeweiligen Spalte.

7.1 Import mit dem LOAD CSV-Befehl

Mit dem Befehl `LOAD CSV` können die CSV-Dateien mit den Regesten in die Graphdatenbank importiert werden.⁶ Hierfür muss die Datenbank aber Zugriff auf die CSV-Daten haben. Dies ist einerseits über den im Datenbankverzeichnis vorhandene Ordner `import` oder über eine URL auf die CSV-Datei möglich. Da sich die einzelnen Zugriffswägen auf den `import`-Ordner von Betriebssystem zu Betriebssystem unterscheiden wird hier beispielhaft der Import mit einer URL vorgestellt. Hierfür wird ein Webserver benötigt, auf den man die CSV-Datei hochlädt und sich anschließend die Webadresse für den Download der Datei notiert.

7.2 Google-Docs für den CSV-Download

Da viele aber keinen Zugriff auf einen eigenen Webserver haben wird hier auch der Download der CSV-Dateien über Google-Docs erklärt. Zunächst benötigt

⁶Verwendet wird die Graphdatenbank neo4j. Die Community-Edition ist kostenlos erhältlich unter <https://www.neo4j.com>.

man hierfür einen Google-Account. Anschließend öffnet man Google-Drive und erstellt dort eine leere Google-Tabellen-Datei in der man dann die CSV-Datei hochladen und öffnen kann.

uid	start_date	end_date	identifier	regnum	persistent_id	title	date_st	date_end
1	20053	1050-11-11	1050-11-11	RI III,2,3 n. 1	1	1050-11-11	Heinrich IV.	1050 N
2	20054	1050-12-25	1050-12-25	RI III,2,3 n. 2	2	1050-12-25	Heinrich IV.	1050 D
4	20055	1051-0					Heinrich IV.	1051 F
5	20056	1051-0					Heinrich IV.	1051 M
6	20057	1051-0					Heinrich IV.	1051 M
7	20058	1051-1					Heinrich IV.	1051 N
8	20059	1051-1					Heinrich IV.	1051 D
9	20060	1052-0					Heinrich IV.	1052 M
10	20061	1052-0					Heinrich IV.	1052 M
11	20062	1052-0					Heinrich IV.	1052 Jt
12	20063	1052-0					Heinrich IV.	1052 (S
13	20064	1052-1					Heinrich IV.	1052 D
14	20065	1053-1					Heinrich IV.	1053 (v
15	20066	1053-1					Heinrich IV.	1053 D
16	20067	1054-0					Heinrich IV.	1054 A
17	20068	1054-0					Heinrich IV.	1054 M
18	20069	1054-0					Heinrich IV.	1054 M
19	20070	1054-0					Heinrich IV.	1054 Jt
20	20071	1054-07-17	1054-07-17	RI III,2,3 n. 19	19	1054-07-17	Heinrich IV.	1054 (r
21	20072	1054-11-17	1054-11-17	RI III,2,3 n. 20	20	1054-11-17	Heinrich IV.	1054 N
22	20073	1054-11-17	1054-11-17	RI III,2,3 n. 21	21	1054-11-17	Heinrich IV.	1054 N
23	20074	1054-12-25	1054-12-25	RI III,2,3 n. 22	22	1054-12-25	Heinrich IV.	1054 D
24	20075	1055-01-16	1055-01-16	RI III,2,3 n. 23	23	1055-01-16	Heinrich IV.	1055 Jt
25	20076	1055-03-03	1055-03-03	RI III,2,3 n. 24	24	1055-03-03	Heinrich IV.	1055 M
26	20077	1055-03-03	1055-03-03	RI III,2,3 n. 25	25	1055-03-03	Heinrich IV.	1055 M

Abbildung 8: Freigabe der Datei zum Ansehen für Dritte!

Wichtig ist nun, die Datei zur Ansicht freizugeben (Klick auf **Freigeben** oben rechts im Fenster dann Link zum Freigeben abrufen und anschließend bestätigen). Jetzt ist die CSV-Datei in Google-Docs gespeichert und kann von Dritten angesehen werden. Für den Import in die Graphdatenbank benötigen wir aber einen Download im CSV-Format. Diesen findet man unter **Datei/Herunterladen als/Kommagetrennte Werte.csv aktuelles Tabellenblatt**.

Damit lädt man das aktuelle Tabellenblatt als CSV runter. Nach dem Download muss man nun im Browser unter Downloads den Download-Link der Datei suchen und kopieren.

Regesten

Datei Bearbeiten Ansehen Einfügen Format Daten Tools Add-ons Hilfe Letzte Änderung am 2

Freigeben... .00 123 Arial 10 B I S A

fx | u Neu ►

1 Öffnen... Strg+O C D E

2 Importieren... end_date identifier regnum per

3 Kopie erstellen... 050-11-11 RI III,2,3 n. 1 1 10

4 Herunterladen als 050-12-25 RI III,2,3 n. 2 2 10

5 Als E-Mail-Anhang senden... Microsoft Excel (.xlsx) 3 10

6 Versionsverlauf OpenDocument-Format (.ods) 4 10

7 Umbenennen... PDF-Dokument (.pdf) 5 10

8 Verschieben nach... Webseite (HTML, komprimiert) 6 10

9 In Papierkorb verschieben Kommagetrennte Werte (.csv, aktuelles Tabellenblatt) 7 10

10 Im Web veröffentlichen... Tabulatorgetrennte Werte (.tsv, aktuelles Tabellenblatt) 8 10

11 E-Mail an Mitbearbeiter senden... 052-03-27 RI III,2,3 n. 9 9 10

12 Dokumentdetails... 052-06-07 RI III,2,3 n. 10 10 10

13 Tabelleneinstellungen... 052-09-30 RI III,2,3 n. 11 11 10

14 Drucken Strg+P 052-12-25 RI III,2,3 n. 12 12 10

15 053-11-03 RI III,2,3 n. 13 13 10

16 053-12-25 RI III,2,3 n. 14 14 10

17 054-04-03 RI III,2,3 n. 15 15 10

18 20068 1054-05-29 1054-05-29 RI III,2,3 n. 16 16 10

19 20069 1054-05-31 1054-05-31 RI III,2,3 n. 17 17 10

20 20070 1054-07-17 1054-07-17 RI III,2,3 n. 18 18 10

21 20071 1054-07-17 1054-07-17 RI III,2,3 n. 19 19 10

22 20072 1054-11-17 1054-11-17 RI III,2,3 n. 20 20 10

23 20073 1054-11-17 1054-11-17 RI III,2,3 n. 21 21 10

24 20074 1054-12-25 1054-12-25 RI III,2,3 n. 22 22 10

25 20075 1055-01-16 1055-01-16 RI III,2,3 n. 23 23 10

26 20076 1055-03-03 1055-03-03 RI III,2,3 n. 24 24 10

27 20077 1055-03-03 1055-03-03 RI III,2,3 n. 25 25 10

+ tx_hisodat_domain_model_dateranges

Abbildung 9: Herunterladen als CSV-DAtei

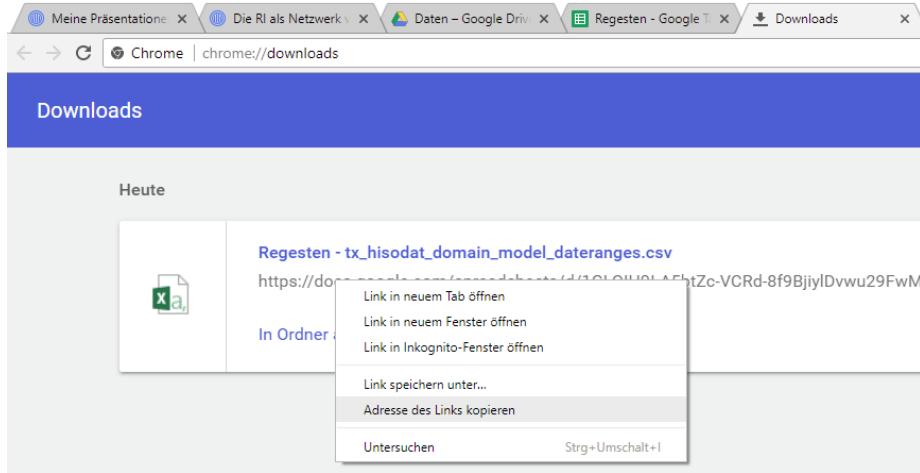


Abbildung 10: Download-Link der CSV-Datei

7.3 Die neo4j-Eingabezeile

Nachdem die Download-URL der CSV-Datei nun ermittelt ist, kann der LOAD CSV-Befehl angepasst und ausgeführt werden.

```
// ReggH4-Regesten und Datumsangaben aus Google-Docs importieren
LOAD CSV WITH HEADERS FROM "https://docs.google.com/spreadsheets/d/1GLQIH9LA5btZc-VCRd-8f9B"
CREATE (r:Regesta {regid:line.persistentIdentifier, text:line.summary, archivalHistory:line}
MERGE (d:Date {startDate:line.start_date, endDate:line.end_date})
MERGE (r)-[:DATE]->(d)
;
```

Der LOAD CSV-Befehl kann noch um die Angabe WITH HEADERS FROM ergänzt werden. Dann werden die Angaben in der ersten Zeile für die Identifizierung der Spalten verwendet. In den Anführungszeichen nach dem Befehl LOAD CSV WITH HEADERS FROM wird der Download-Link der CSV-Datei angegeben. Der LOAD CSV-Befehl lädt dann bei seiner Aufführung die CSV-Datei von der angegebenen URL und gibt sie zeilenweise an die folgenden cypher-Befehle weiter.

	A	B	C	D	E	F	G	H
1	summary	uid	start_date	end_date	identifier	sorting	persistent_identifier	title
2	Kg. F. quittiert der Stadt Lübeck die Summe von 100 Gulden, die sie ihm schuldete.	97164	1440-06-05	1440-06-05	[RI XIII] H. 1 n. 1	013_001_000_008973_00	1440-06-05_1_0_13_1_0	Friedrich III.
3	Kg. F. bestätigt auf Bitte des Konsistoriums die Summe von 100 Gulden, die die Stadt Lübeck ihm schuldete.	97173	1442-04-23	1442-04-23	[RI XIII] H. 1 n. 10	013_001_000_008982_00	1442-04-23_1_0_13_1_0	Friedrich III.
4	K. F. bekundet, daß die Stadt Lübeck die Summe von 100 Gulden, die sie ihm schuldete.	97263	1473-07-06	1473-07-06	[RI XIII] H. 1 n. 100	013_001_000_009072_00	1473-07-06_1_0_13_1_0	Friedrich III.

Abbildung 11: Blick auf die ersten Zeilen der Google-spreadsheets-Tabelle.

Mit dem CREATE-Befehl wird in der Graphdatenbank ein Knoten vom Typ Re-

gestae erzeugt.⁷ Diesem Knoten werden noch verschiedene Eigenschaften wie der Regestenidentifier, das Regest, die Überlieferung, die Datumsangabe und die Regestennummer mitgegeben. Wie dem Beispiel zu entnehmen ist, sind die Eigenschaften in CamelCase-Notation angegeben (z.B. archivalHistory).⁸ Dies ist die hier übliche Notation, da Leerzeichen nicht verwendet werden dürfen und der Unterstrich je nach Betriebssystems Probleme verursachen kann. In der folgenden Tabelle sind die einzelnen Eigenschaften nochmal zusammengefasst und beispielhaft für das erste Regest aufgelistet.

Eigenschaft	Spaltenüberschrift	Bedeutung	Wert
regid	persistent_identifier	Ident des Regests	1050-11-11_1_0_3_2_3
text	summary	Regestentext	Heinrich wird als vierter
archivalHistory	archival_history	Überlieferung	Herim. Aug. 1050 ...
date	date_string	Datumsangabe des Regests	1050 November 11
regnum	regnum	Regestennummer innerhalb des Bandes	1

Der CREATE-Befehl bekommt über das Array line beim ersten Durchlauf die Zellen der ersten Zeile der CSV-Datei. Über line.regid kann dann auf den Wert der Spalte regid zugegriffen und dieser dann für die Erstellung der Eigenschaft regid verwendet werden. Auf die gleiche Weise werden dann auch die weiteren Eigenschaften des Regestenknotens erstellt. In der folgenden Abbildung sind die Eigenschaften des Regestenknotens dargestellt.

Regest <id>: 619802 registerid: #19-316 ident: [Ri XIII] H. 19 n. 316
Text:
K. F. gewährt Bürgermeistern, Rat und Bürgern der Stadt Nürnberg in Anbetracht der Tatsache, daß die auf unfruchtbarem Boden und nicht an einem schiffbaren Gewässer liegende Stadt in besonderer Weise auf den Handelsverkehr auf unser und des reichs strassen angewiesen ist, die besondere Gnade, vom künftigen heiligen Weihnacht tag (1452 Dezember 25) an ein Jahr lang Umgang mit Ächtern und Aberächtern pflegen zu dürfen, und gebietet allen Fürsten, Gff. etc. und Reichsuntertanen die Beachtung dieser Gnade bei seiner und des Reichs Ungnade.
regid: 1452-12-20_1_0_13_19_0_316_316
Überlieferung:
Org. im STA Nürnberg (Sign. Rst. Nürnberg, Kaiserprivilegien Nr. 431), Perg., rotes S 18 rückw. aufgedrückt. - Kop.: Abschriften ebd. (Sign. Rst. Nürnberg, Amts- und Standbücher Nr. 44: Großes Grünbuch fol. 20^v-^r 21^r-^v u. ebd. Nr. 49: N. Schwarzbuch I fol. 324^v-^r /kurztitlesuche_r.php?kurztitle=muellner_annalen>MÜLLNER</link>, Annalen 2 S. 493. Lit.: clink http://opac.regesta-imperii.de/lang_de /kurztitlesuche_r.php?kurztitle=baader_handel>BAADER</link>, Handel S. 97; clink http://opac.regesta-imperii.de/lang_de /kurztitlesuche_r.php?kurztitle=ruf_acht>RUF</link>, Acht S. 46 Das Stück ist textlich identisch mit dem Privileg von 1451 Juli 28 (n. 185), nimmt indes keinen Bezug auf dieses. Bei der Strafandrohung wird im Gegensatz zu dem früheren Stück nur auf die einfache herrscherliche Ungnade verwiesen. Die Befristung zeigt, daß der Gesandte Erhard Gyner keinen Erfolg gehabt hatte, seinem Auftrag von 1452 Oktober 26 entsprechend den Zeitraum der Frist länger fassen zu lassen (StA Nürnberg, Sign. Rst. Nürnberg, Briefbücher Nr. 23 fol. 56^v). Vgl. auch n. 359.

Abbildung 12: Die Eigenschaften des Regests, registerid, ident, Text regid und Überlieferung.

In einer Zeile der CSV-Datei finden sich alle Angaben eines Regests. Die in der oben abgebildeten Tabelle angegebenen Werte werden als Eigenschaften des Regestenknotens erstellt.

⁷ Die Angaben in der Graphdatenbank sind Englisch, daher *Regesta*.

⁸ Gemeint ist hier der lowerCamelCase bei dem der erste Buchstabe kleingeschrieben und dann jedes angesetzte Wort mit einem Großbuchstaben direkt angehängt (wie bei archivalHistory). Vgl. auch <https://de.wikipedia.org/wiki/Binnenmajuskel#Programmiersprachen>.

In der nächsten Abbildung wird das Modell des Regests im Graphen abgebildet.

Friedrich III. - [RI XIII] H. 19 n. 316

URI

Merken

1452 Dezember 20, Wiener Neustadt

K. F. gewährt Bürgermeistern, Rat und Bürgern der Stadt Nürnberg in Anbetracht der Tatsache, daß die auf unfruchtbarem Boden und nicht an einem schiffbaren Gewässer liegende Stadt in besonderer Weise auf den Handelsverkehr auf *unser und des reichs strassen* angewiesen ist, die besondere Gnade, vom künftigen *heiligen Weihnacht tag* (1452 Dezember 25) an ein Jahr lang Umgang mit Ächtern und Aberächttern pflegen zu dürfen, und gebietet allen Fürsten, Gff. etc. und Reichsuntertanen die Beachtung dieser Gnade bei seiner und des Reichs Ungnade.

Originaldatierung: *An sannd Thamas abend des heiligen zwelfboten.*

Kanzleivermerke: KVR: A.m.d.i.i.c. Ulricus Weltzli. - KVv: Tho(ma)s 52 (unterer linker Blattrand); von echt(ern) etc. (Empfängervermerk auf der Rückseite).

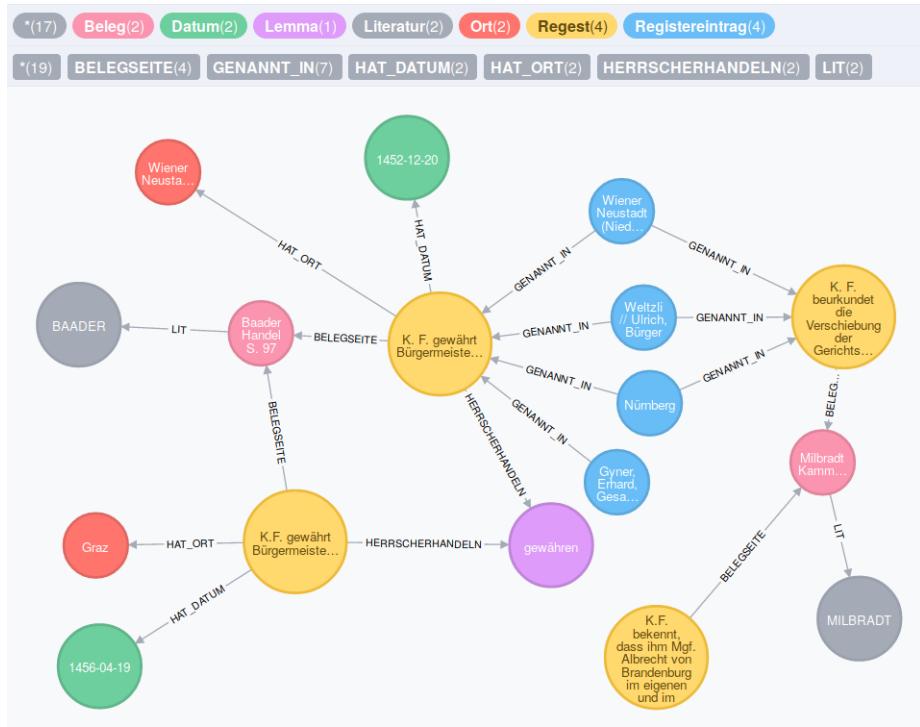
Überlieferung/Literatur

Org. im StA Nürnberg (Sign. Rst. Nürnberg, Kaiserprivilegien Nr. 431), Perg., rotes S 18 rücks. aufgedrückt.
- Kop.: Abschriften ebd. (Sign. Rst. Nürnberg, Amts- und Standbücher Nr. 44: Großes Grünbuch fol. 20^v-
21^r u. ebd. Nr. 49: N. Schwarzbuch I fol. 324^v -335^v), Perg. bzw. Pap. (15. Jh.).

Erwähnt von MÜLLNER, Annalen 2 S. 493.

Lit.: BAADER, Handel S. 97; RUF, Acht S. 46

Das Stück ist textlich identisch mit dem Privileg von 1451 Juli 28 (n. 185), nimmt indes keinen Bezug auf dieses. Bei der Strafandrohung wird im Gegensatz zu dem früheren Stück nur auf die einfache herrscherliche Ungnade verwiesen. Die Befristung zeigt, daß der Gesandte Erhard Gyner keinen Erfolg gehabt hatte, seinem Aufrag von 1452 Oktober 26 entsprechend den Zeitraum der Frist länger fassen zu lassen (StA Nürnberg, Sign. Rst. Nürnberg, Briefbücher Nr. 23 fol. 56^v). Vgl. auch n. 359.



Die gelben Knoten sind die Regesten. Aus den Angaben des Regests werden mit dem o.a. Befehl noch ein Datumsknoten und ein Ortsknoten erstellt. Mit dem ersten CREATE-Befehl werden die Regesten erstellt. Mit den folgenden MERGE-Befehlen werden anschließend ergänzende Knoten für die Datumsangaben und die Ausstellungsorte erstellt. Nun ist es aber so, dass Ausstellungsort und Ausstellungsdatum mehrfach vorkommen können. Daher wird der hier nicht der CREATE-Befehl sondern der MERGE-Befehl verwendet. Dieser funktioniert wie der CREATE-Befehl, prüft aber vorher ob in der Datenbank ein solcher Knoten schon existiert. Falls es ihn noch nicht gibt wird er erzeugt, wenn es ihn schon gibt, wird er der entsprechenden Variable zugeordnet. Anschließend wird dann die Kante zwischen Regestenknoten und Ausstellungsortsknoten und Regestenknoten und Datumsknoten erstellt. In der folgenden Tabelle werden die einzelnen Befehle dargestellt und kommentiert.

Befehl	Variablen	Bemerkungen
LOAD CSV WITH HEADERS FROM “https://docs.google.com/” ... AS line	line	Import der CSV- Dateien. Es wird jeweils eine Zeile an die Variable line weitergegeben
CREATE (r:Regest {regid: li- ne.summ ne.persistent Text:line.summary, Überliefe- rung: li- ne.archival_history, ident:line.identifier})	line.persis li- ne.summ ne.persistent eidentifier Für die weiteren Befehlt steht der neu erstellt Reges- tenkno- ten unter der Variable r zur Verfügung.	Erstellt des Re- gisten- knotens. Text: Überliefe- rung: li- ne.archival_history, ident:line.identifier})

Befehl	Variablen	Bemerkungen
MERGE (d:Datum {startdate: line.start_date, enddate:line.end_date}) li-	line.start_Erford und geprüft, line.end_dat ein ne.start_date, enddate:line.end_date} mit der Datums- angabe schon existiert, falls nicht, wird er erstellt. In jedem Fall steht anschlie- ßend der Datums- knoten unter der Variable d zur Verfügung.	Erfordert geprüft, line.end_dat ein ne.start_date, enddate:line.end_date} mit der Datums- angabe schon existiert, falls nicht, wird er erstellt. In jedem Fall steht anschlie- ßend der Datums- knoten unter der Variable d zur Verfügung.

Befehl	Variablen	Bemerkungen
MERGE (o:Ort {ort:line.name Orts- latitu- de:toFloat(line. latitude) longitude:toFloat(line. longitude })} >(d)	line.name ist der name, de:toFloat(latitude) longitude:toFloat(line.longitude)}} Anga- ben sind die Geoda- ten des Ortes	Es wird geprüft, ob ein Ortskno- nen existiert, falls nicht, wird er erstellt. In jedem Fall steht anschlie- ßend der Ortskno- ten unter der Variable o zur Verfügung. Zwischen Reges- tenkno- ten und Datums- knoten wird Datumsknoten HAT_DATUM- Kante erstellt.
MERGE (r)->(d)	(r) ist [:HAT_DATUM]Re- gesten- kno- ten, (d) ist der	

Befehl	Variablen	Bemerkungen
MERGE (r)-[:HAT_ORT]-(o);	(r) ist gesten- kno- ten, (o) ist der Ortsknoten und eine Ortsknote.	Zwischen Regestenknosten und Ortsknoten wird eine HAT_ORT-Kante erstellt.

8 Erstellen der Ausstellungsorte

In den Kopfzeilen der Regesten ist, soweit bekannt, der Ausstellungsort der Urkunde vermerkt. Im Rahmen der Arbeiten an den Regesta Imperii Online wurden diese Angaben zusammengestellt und soweit möglich die Orte identifiziert, so dass diese Angabe nun bei der Erstellung der Regestendatenbank im Graphen berücksichtigt werden können. Insgesamt befinden sich in den Regesta Imperii über 12.000 verschiedene Angaben für Ausstellungsorte, wobei sie sich teilweise aber auch auf den gleichen Ort beziehen können (Wie z.B. Aachen, Aquisgrani, Aquisgradi, Aquisgranum, coram Aquisgrano etc.). Allein mit den 1.000 häufigsten Ortsangaben konnten schon die Ausstellungsorte der Mehrzahl der Regesten georeferenziert werden.

Mit dem folgenden cypher-Query werden die Ausstellungsorte in die Graphdatenbank importiert:

```
// RI-Ausstellungsorte-geo erstellen
LOAD CSV WITH HEADERS FROM "https://docs.google.com/spreadsheets/d/13_f6Vja4Hf0pju9RVDubHiM"
WITH line
WHERE line.Lat IS NOT NULL
AND line.normalisiertDeutsch IS NOT NULL
MATCH (r:Regesta {origPlaceOfIssue:line.Original})
MERGE (p:Place {normalizedGerman:line.normalisiertDeutsch, longitude:line.Long, latitude:line.Lat})
WITH r, p, line
MERGE (r)-[rel:PLACE_OF_ISSUE]->(p)
SET p.wikidataId = line.wikidataId
SET p.name = line.name
SET p.gettyId = line.GettyId
SET p.geonamesId = line.GeonamesId
SET rel.original = line.Original
SET rel.alternativeName = line.Alternativname
SET rel.commentary = line.Kommentar
```

```

SET rel.allocation = line.Zuordnung
SET rel.state = line.Lage
SET rel.certainty = line.Sicherheit
SET rel.institutionInCity = line.InstInDerStadt
RETURN count(p)
;

```

Da Import-Query ist etwas komplexer ist, wird er im folgenden näher erläutert. Nach dem `LOAD CSV WITH HEADERS FROM`-Befehl wird zunächst überprüft, ob der jeweils eingelesene Eintrag in der Spalte `line.lat` und in der Spalte `line.normalisiertDeutsch` Einträge hat. Ist dies der Fall wird überprüft, ob es einen Regestenknoten gibt, der einen Ausstellungsorteintrag hat, der der Angabe in der Spalte `Original` entspricht. Diese Auswahl ist notwendig, da in der Tabelle die Ausstellungsorte der gesamten Regesta Imperii enthalten sind, wir beim Import aber nur die Ortsknoten erstellen, die für die Regesten Kaiser Heinrichs IV. relevant sind. Sind die genannten Bedingungen erfüllt, wird mit dem `MERGE`-Befehl der Place-Knoten erstellt und anschließend mit dem Regestenknoten verknüpft. Schließlich werden noch weitere Details der Ortsangabe im Place-Knoten und in den `PLACE_OF_ISSUE`-Kanten ergänzt.

Mit dem folgenden Query werden die Koordinatenangaben zu Höhen- und Breitengraden der Ausstellungsorte (Place-Knoten), die in den Propertys Lat und Long abgespeichert sind in der neuen Property LatLong zusammengefasst und in point-Werte umgewandelt. Seit Version 3 kann neo4j mit diesen Werten Abstandsberechnungen durchführen (Mehr dazu siehe unten bei den Auswertungen).

```

// Regesten und Ausstellungsorte mit Koordinaten der Ausstellungsorte versehen
MATCH (r:Regesta)-[:PLACE_OF_ISSUE]->(o:Place)
SET r.latLong = point({latitude: tofloat(o.latitude), longitude: tofloat(o.longitude)})
SET o.latLong = point({latitude: tofloat(o.latitude), longitude: tofloat(o.longitude)})
SET r.placeOfIssue = o.normalizedGerman
SET r.latitude = o.latitude
SET r.longitude = o.longitude
;

```

In den Regesta Imperii Online sind die Datumsangaben der Regesten isokonform im Format JJJJ-MM-TT (also Jahr-Monat-Tag) abgespeichert. neo4j behandelt diese Angaben aber als String. Um Datumsberechnungen durchführen zu können, müssen die Strings in Datumswerte umgerechnet werden. Der cypher-Query hierzu sieht wie folgt aus:

```

// Date in Isodatum umwandeln
MATCH (n:Regesta)
SET n.isoStartDate = date(n.startDate);
MATCH (n:Regesta)
SET n.isoEndDate = date(n.endDate);
MATCH (d:Date)

```

```

SET d.isoStartDate = date(d.startDate);
MATCH (d:Date)
SET d.isoEndDate = date(d.endDate);

```

Zunächst werden mit dem MATCH-Befehl alle Regestenknoten aufgerufen. Anschließend wird für jeden Regestenknoten aus der String-Property `startDate` die Datumsproperty `isoStartDate` berechnet und im Regestenknoten abgespeichert. Mit Hilfe der Property können dann Datumsangaben und Zeiträume abgefragt werden (Beispiel hierzu unten in der Auswertung).

8.0.1 Herrscherhandeln in den Regesta Imperii

Regesten sind in ihrer Struktur stark formalisiert. Meist wird mit dem ersten Verb im Regest das Herrscherhandeln beschrieben. Um dies auch digital auswerten zu können haben wir in einem kleinen Testprojekt mit Hilfe des Stuttgart-München Treetaggers aus jedem Regest das erste Verb extrahiert und normalisiert. Die Ergebnisse sind in folgender Tabelle einsehbar. Diese Tabelle wird mit dem folgenden cypher-Query in die Graphdatenbank eingelesen.

```

// ReggH4-Herrscherhandeln
LOAD CSV WITH HEADERS FROM "https://docs.google.com/spreadsheets/d/1nlbZmQYcT1E3Z58yPmcnulch
AS line FIELDTERMINATOR ','
MATCH (r:Regesta{ident:line.regid})
MERGE (1:Lemma{lemma:line.Lemma})
MERGE (r)-[:ACTION]->(1);

```

Dabei wird zunächst mit dem MATCH-Befehl das jeweilige Regest gesucht, anschließend mit dem MERGE-Befehl der Lemma-Knoten für das Herrscherhandeln angelegt (falls noch nicht vorhanden) und schließlich der Regesta-knoten mit dem Lemma-Knoten über eine ACTION-Kante verbunden. Auswertungsperspektiven finden Sie hier #####

8.0.2 Zitationsnetzwerke in den Regesta Imperii

In vielen Online-Regesten ist die zitierte Literatur mit dem Regesta-Imperii-Opac verlinkt. Da es sich um URLs handelt, sind diese Verweise eindeutig andererseits lassen sie sich mit regulären Ausdrücken aus den Regesten extrahieren. Mit folgendem Query werden aus den Überlieferungsteilen der Regesten die mit dem Opac verlinkten Literaturangaben extrahiert und jede Literaturangabe als Reference-Knoten angelegt.

```

// ReggH4-Literaturnetzwerk erstellen
MATCH (reg:Regesta)
WHERE reg.archivalHistory CONTAINS "link"
UNWIND apoc.text.regexGroups(reg.archivalHistory, "<link (\\\S+)>(\\\S+)</link>") as link
MERGE (ref:Reference {url:link[1]}) ON CREATE SET ref.title=link[2]

```

```
MERGE (reg)-[:REFERENCES]->(ref);
```

Da dies mit dem **MERGE**-Befehl geschieht, wird in der Graphdatenbank jeder Literaturtitel nur einmal angelegt. Anschließend werden die **Reference**-Knoten mit den Regesten über **REFERENCES**-Kanten verbunden. Zu den Auswertungsmöglichkeiten vgl. unten den Abschnitt zu den Auswertungsperspektiven.

9 Import der Registerdaten in die Graphdatenbank

9.1 Vorbereitung der Registerdaten

Register spielen für die Erschließung von gedrucktem Wissen eine zentrale Rolle, da dort in alphabetischer Ordnung die im Werk vorkommenden Entitäten (z.B. Personen und Orte) hierarchisch gegliedert aufgeschlüsselt werden. Für die digitale Erschließung der Regesta Imperii sind Register von zentraler Bedeutung, da mit ihnen die in den Regesten vorkommenden Personen und Orte bereits identifiziert vorliegen. Für den Import in die Graphdatenbank wird allerdings eine digitalisierte Fassung des Registers benötigt. Im Digitalisierungsprojekt Regesta Imperii Online wurden Anfang der 2000er Jahre auch die gedruckt vorliegenden Register digitalisiert. Sie dienen nun als Grundlage für die digitale Registererschließung der Regesta Imperii. Im hier gezeigten Beispiel werden die Regesten Kaiser Heinrichs IV. und das dazugehörige Register importiert. Da der letzte Regestenband der Regesten Kaiser Heinrichs IV. mit dem Gesamtregister erst vor kurzem gedruckt wurde, liegen hier aktuelle digitale Fassung von Registern und Regesten vor. Die für den Druck in Word erstellte Registerfassung wird hierfür zunächst in eine hierarchisch gegliederte XML-Fassung konvertiert, damit die Registerhierarchie auch maschinenlesbar abgelegt ist.

In der XML-Fassung sind die inhaltlichen Bereiche und die Abschnitte für die Regestennummern jeweils extra in die Tags **<Inhalt** und **Regestennummer** eingefasst. Innerhalb des Elements **Regestennummer** ist dann nochmal jede einzelne Regestennummer in **<r>**-Tags eingefasst. Die aus dem gedruckten Register übernommenen Verweise sind durch ein leeres **<vw/>**-Element gekennzeichnet.

Die in XML vorliegenden Registerdaten werden anschließend mit Hilfe von TuStep in einzelne CSV-Tabellen zerlegt.

In einer Tabelle werden alle Entitäten aufgelistet und jeweils mit einer ID versehen.

In der anderen Tabelle werden die Verknüpfungen zwischen Registereinträgen und den Regesten aufgelistet. Der Registereintrag Adalbero kommt also in mehreren Regesten vor. Da das Register der Regesten Heinrichs IV. nur zwei Hierarchiestufen enthält, in denen beispielsweise verschiedene Amtsphasen ein und

```

<Stufe0 id="H4P00005">
    <Inhalt>Abiram, biblische Gestalt, Sohn Eliabs, Bruder Dathans, Auflehrer gegen
        Moses</Inhalt>
    <Regestennummer>
        <r>762</r>
    </Regestennummer>
</Stufe0>
<Stufe0 id="H4P00006">
    <Inhalt>AC – Gottschalk v. Aachen</Inhalt>
</Stufe0>
<Stufe0 id="H4P00007">
    <Inhalt>Achalmer, Adelsgeschlecht aus Schwaben</Inhalt>
    <Regestennummer>
        <r>363</r>
    </Regestennummer>
<Stufe1>
    <Inhalt><vw/> Liutold, Gf. v. Achalm</Inhalt>
</Stufe1>
<Stufe1>
    <Inhalt><vw/> Werner (v. Achalm), Bf. II. v. Straßburg</Inhalt>
</Stufe1>
<Stufe1>
    <Inhalt><vw/> Williberga</Inhalt>
</Stufe1>
</Stufe0>

```

Abbildung 13: Ausschnitt aus dem XML-Register der Regesten Heinrichs IV.

6	H4P00005	Abiram, biblische Gestalt, Sohn Eliabs, Bruder Dathans, Auflehrer gegen Moses			
7	H4P00006	AC – Gottschalk v. Aachen			
8	H4P00007	Achalmer, Adelsgeschlecht aus Schwaben			
9	H4P00008	Aczo, Sohn Rudolf Rubias, Bruder Attos			
10	H4P00009	Adalbero, Mgf. d. karantanischen Mark			
11	H4P00010	Adalbero, Gf. v. Ebersberg, Gem. Richildes			
12	H4P00011	Adalbero, Edelfreier			
13	H4P00012	Adalbero A (AA), namentlich unbekannter Bamberger Diktator			

Abbildung 14: Ausschnitt der Entitätentabelle des Registers der Regesten Heinrichs IV.

ID	regnum	regnum2	name1	name2			
H4P00001	805	805		<i>A. </i>,			
H4P00002	1517	1517		<i>A.</i>,			
H4P00003	1519	1519		<i>A.</i>, <i>centurio</i>			
H4P00005	762	762		Abiram, biblische Gestalt, Sohn Eliabs, Bruder Dathans, Auflehrer gegen Moses			
H4P00007	363	363		Achalmer, Adelsgeschlecht aus Schwaben			
H4P00008	1056	1056		Aczo, Sohn Rudolf Rubias, Bruder Attos			
H4P00009	714	714		Adalbero, Mgf. d. karantanischen Mark			
H4P00010	78	78		Adalbero, Gf. v. Ebersberg, Gem. Richildes			
H4P00011	331	331		Adalbero, Edelfreier			
H4P00012	666	666		Adalbero A (AA), namentlich unbekannter Bamberger Diktator			
H4P00012	717	717		Adalbero A (AA), namentlich unbekannter Bamberger Diktator			
H4P00012	959	959		Adalbero A (AA), namentlich unbekannter Bamberger Diktator			
H4P00012	1400	1400		Adalbero A (AA), namentlich unbekannter Bamberger Diktator			
H4P00012	1403	1403		Adalbero A (AA), namentlich unbekannter Bamberger Diktator			
H4P00012	1420	1420		Adalbero A (AA), namentlich unbekannter Bamberger Diktator			
H4P00012	1481	1481		Adalbero A (AA), namentlich unbekannter Bamberger Diktator			
H4P00012	1485	1485		Adalbero A (AA), namentlich unbekannter Bamberger Diktator			

Abbildung 15: Ausschnitt der Verknüpfungstabelle des Registers der Regesten Heinrichs IV.

derselben Person unterschieden werden, wurden diese beim Import zusammengefasst.⁹ Damit gibt es pro Person jeweils nur einen Knoten.

9.2 Import der Registerdaten in die Graphdatenbank

Im Gegensatz zu den Regesten Kaiser Friedrichs III., bei denen Orte und Personen in einem Register zusammengefasst sind, haben die Regesten Kaiser Heinrich IV. getrennte Orts- und Personenregister. Die digitalisierten Registerdaten können hier eingesehen werden. In dem Tabellendokument befinden sich insgesamt drei Tabellen. In der Tabelle Personen sind die Einträge des Personenregisters aufgelistet und in der Tabelle Orte befindet sich die Liste aller Einträge des Ortsregisters. Schließlich enthält die Tabelle APPEARS_IN Information dazu, welche Personen oder Orte in welchen Regesten genannt sind. Der folgende cypher-Query importiert die Einträge der Personentabelle in die Graphdatenbank und erstellt für jeden Eintrag einen Knoten vom Typ :IndexPerson:

```
// Registereinträge Personen erstellen
LOAD CSV WITH HEADERS FROM "https://docs.google.com/spreadsheets/d/12T-RD1Ct4aAUNNNxipjMmHe9B...AS line
CREATE (:IndexPerson {registerId:line.ID, name1:line.name1});
```

Mit dem folgenden cypher-Query werden nach dem gleichen Muster aus der Tabelle Orte die Ortseinträge in die Graphdatenbank importiert.

```
OAD CSV WITH HEADERS FROM "https://docs.google.com/spreadsheets/d/12T-RD1Ct4aAUNNNxipjMmHe9B...AS line
CREATE (:IndexPlace {registerId:line.ID, name1:line.name1});
```

Die beiden Befehle greifen also auf verschiedene Tabellenblätter des gleichen Google-Tabellendokuments zu, laden es als CSV-Daten und übergeben die Daten zeilenweise an die weiteren Befehle (Hier an den MATCH- und den CREATE-Befehl). Im nächsten Schritt werden nun mit den Daten der APPEARS_IN-Tabelle die Verknüpfungen zwischen den Registereinträgen und den Regesten erstellt.

```
// PLACE_IN-Kanten für Orte erstellen
LOAD CSV WITH HEADERS FROM "https://docs.google.com/spreadsheets/d/12T-RD1Ct4aAUNNNxipjMmHe9B...AS line
MATCH (from:IndexPlace {registerId:line.ID})
MATCH (to:Regesta {regnum:line.regnum2})
CREATE (from)-[:PLACE_IN {regnum:line.regnum, name1:line.name1, name2:line.name2}]->(to);
```

Dabei werden zunächst mit den beiden MATCH-Befehlen jeweils das Regest und der Registereintrag aufgerufen und schließend mit dem CREATE-Befehl

⁹Vgl. die Vorbemerkung zum Register in Böhmer, J. F., Regesta Imperii III. Salisches Haus 1024-1125. Tl. 2: 1056-1125. 3. Abt.: Die Regesten des Kaiserreichs unter Heinrich IV. 1056 (1050) - 1106. 5. Lief.: Die Regesten Rudolfs von Rheinfelden, Hermanns von Salm und Konrads (III.). Verzeichnisse, Register, Addenda und Corrigenda, bearbeitet von Lubich, Gerhard unter Mitwirkung von Junker, Cathrin; Klocke, Lisa und Keller, Markus - Köln (u.a.) (2018), S. 291.

eine PLACE_IN-Kante zwischen den beiden Knoten angelegt, die als Attribute den Inhalt der Spalten `name1` und `name2` mitbekommt. Analog werden die Verknüpfungen zwischen Regestenknoten und Personenknoten angelegt:

```
// PERSON_IN-Kanten für Person erstellen
LOAD CSV WITH HEADERS FROM "https://docs.google.com/spreadsheets/d/12T-RD1Ct4aAUNNNxipjMmHe
AS line
MATCH (from:IndexPerson {registerId:line.ID}), (to:Regesta {regnum:line.regnum2})
CREATE (from)-[:PERSON_IN {regnum:line.regnum, name1:line.name1, name2:line.name2}]->(to);
```

9.3 Die Hierarchie des Registers der Regesten Kaiser Friedrichs III.

In anderen Registern der Regesta Imperii, wie beispielsweise den Regesten Kaiser Friedrichs III. sind teilweise fünf oder mehr Hierarchiestufen vorhanden, die jeweils auch Entitäten repräsentieren. In diesen Fällen müssen die Hierarchien auch in der Graphdatenbank abgebildet werden, was durch zusätzliche Verweise auf die ggf. vorhandenen übergeordneten Registereinträge möglich wird.

nodeID	xmlID	topnodeID	name1	name3
1	A00000001		Aa, Johann von	Aa, Johann von ~
2	A00000002	1	Sophie von ~, Tc	Aa, Johann von ~ // Sophie von ~, Tochter Johannis, Bürgerin zu Köln
3	A00000003		Aach	Aach (Fluß durch Aach, n. Singen, Baden-Württemberg)
4	A00000004		Aach	Aach (n. Singen, Baden-Württemberg), Stadt
5	A00000005		Aache s. Aacher	Aache s. Aachen
6	A00000006		Aachen	Aachen (Aache; Nordrhein-Westfalen), Stadt
7	A00000007	6	Einwohner und B	Aachen (Aache; Nordrhein-Westfalen), Stadt // Einwohner und Bürger ; s. Col
8	A00000008	6	Fischmarkt	Aachen (Aache; Nordrhein-Westfalen), Stadt // Fischmarkt (Parwisch)
9	A00000009	6	Gerichte	Aachen (Aache; Nordrhein-Westfalen), Stadt // Gerichte
10	A00000010	6	Kurgericht	Aachen (Aache; Nordrhein-Westfalen), Stadt // Gerichte // Kurgericht
11	A00000011	6	Schöffenstuhl, k	Aachen (Aache; Nordrhein-Westfalen), Stadt // Gerichte // Schöffenstuhl, kgl.
12	A00000012	6	Richter und Sch	Aachen (Aache; Nordrhein-Westfalen), Stadt // Gerichte // Schöffenstuhl, kgl.
13	A00000013	6	Schöffen	Aachen (Aache; Nordrhein-Westfalen), Stadt // Gerichte // Schöffenstuhl, kgl.
14	A00000014	6	„Grafschaften“	Aachen (Aache; Nordrhein-Westfalen), Stadt // „Grafschaften“
15	A00000015	6	„Grasgebot“	Aachen (Aache; Nordrhein-Westfalen), Stadt // „Grasgebot“
17	A00000017	6	Meierei	Aachen (Aache; Nordrhein-Westfalen), Stadt // Meierei
19	A00000019	6	Brothaus	Aachen (Aache; Nordrhein-Westfalen), Stadt // Meierei // Brothaus
21	A00000021	6	Gewandhaus	Aachen (Aache; Nordrhein-Westfalen), Stadt // Meierei // Gewandhaus
22	A00000022	6	Grashaus	Aachen (Aache; Nordrhein-Westfalen), Stadt // Meierei // Grashaus
23	A00000023	6	Plankenhaus	Aachen (Aache; Nordrhein-Westfalen), Stadt // Meierei // Plankenhaus
24	A00000024	6	Tuchhaus	Aachen (Aache; Nordrhein-Westfalen), Stadt // Meierei // Tuchhaus
25	A00000025	6	Haus zum Haner	Aachen (Aache; Nordrhein-Westfalen), Stadt // Meierei // Haus zum Haner (zu
26	A00000026	6	zo der Geiss	Aachen (Aache; Nordrhein-Westfalen), Stadt // Meierei // zo der Geiss
27	A00000027	6	Rentmeister	Aachen (Aache; Nordrhein-Westfalen), Stadt // Rentmeister
29	A00000029	6	„Stadtbücher“	Aachen (Aache; Nordrhein-Westfalen), Stadt // „Stadtbücher“
30	A00000030	6	Vogtei	Aachen (Aache; Nordrhein-Westfalen), Stadt // Vogtei

Abbildung 16: Ausschnitt der Entitätentabelle des Registers der Regesten Friedrichs III.

Im Tabellenausschnitt wird jedem Registereintrag in der ersten Spalte eine `nodeID` als eindeutige Kennung zugewiesen. Bei Registereinträgen, die kein Hauptlemma sind, enthält die dritte Spalte `topnodeID` den Verweis auf die eindeutige Kennung `nodeID` des übergeordneten Eintrages. Beim Import in die Graphdatenbank wird diese Hierarchie über `CHILD_OF`-Kanten abgebildet, die

vom untergeordneten Eintrag auf das übergeordnete Lemma verweisen. Damit ist die komplette Registerhierarchie im Graphen abgebildet. In der Spalte `name1` ist das Lemma angegeben, in der Spalte `name3` zusätzliche zum Lemma noch der gesamte Pfad vom Hauptlemma bis zum Registereintrag, jeweils mit Doppelslashes (//) getrennt. Bei tiefer gestaffelten Registern ist teilweise ohne Kenntnis der übergeordneten Einträge eine eindeutige Identifizierung eines Eintrages nicht möglich. So wird in Zeile 17 der o.a. Abbildung allein mit der Angabe aus der Spalte `name1` nicht klar ist, um welche **Meierei** es sich handelt. Mit dem kompletten Pfad des Registereintrages in der Spalte `name3` wird dagegen deutlich, dass die Aachener **Meierei** gemeint ist.

10 Auswertungsperspektiven

10.1 Personennetzwerke in den Registern

10.1.1 Graf Robert II. von Flandern in seinem Netzwerk

Nach dem Import können nun die Online-Regesten und die Informationen aus dem Registern der Regesten Kaiser Heinrichs IV. in einer Graphdatenbank aus einer Vernetzungsperspektive abgefragt werden.^[f663]

Ausgangspunkt ist der Registereintrag von Graf Robert II. von Flandern. Diesen Knoten finden wir mit folgendem Query.

```
// Robert II. von Flandern
MATCH (n:IndexPerson) WHERE n.registerId = 'H4P01822'
RETURN *;
```

Mit einem Doppelklick auf den `IndexPerson`-Knoten öffnen sich alle `Regesta`-Knoten, in denen Robert genannt ist. Klickt man nun wiederum alle Regesten-knoten doppelt, sieht man alle Personen und Orte, mit denen Robert gemeinsam in den Regesten genannt ist.

Dies kann auch in einem cypher-Query zusammengefasst werden.

```
// Robert II. von Flandern mit Netzwerk
MATCH (n:IndexPerson)-[:PERSON_IN]->
(r:Regesta)<-[:PERSON_IN]->
(m:IndexPerson)
WHERE n.registerId = 'H4P01822'
RETURN *;
```

In der folgenden Abb. wird das Ergebnis dargestellt.

Hier wird der MATCH-Befehl um einen Pfad über `PERSON_IN`-Kanten zu `Regesta`-Knoten ergänzt, von denen dann wiederum eine `PERSON_IN`-Kante zu den anderen, in den Regesten genannten `IndexPerson`-Knoten führt.



Abbildung 17: Robert mit den Personen, mit denen er gemeinsam in Regesten genannt wird.

Nimmt man noch eine weitere Ebene hinzu, wächst die Ergebnismenge stark an.

```
// Robert II. von Flandern mit Netzwerk und Herrscherhandeln (viel)
MATCH
(n1:IndexPerson)-[:PERSON_IN]->(r1:Regesta)<-[:PERSON_IN]-
(n2:IndexPerson)-[:PERSON_IN]->(r2:Regesta)<-[:PERSON_IN]-
(n3:IndexPerson)
WHERE n1.registerId = 'H4P01822'
RETURN *;
```

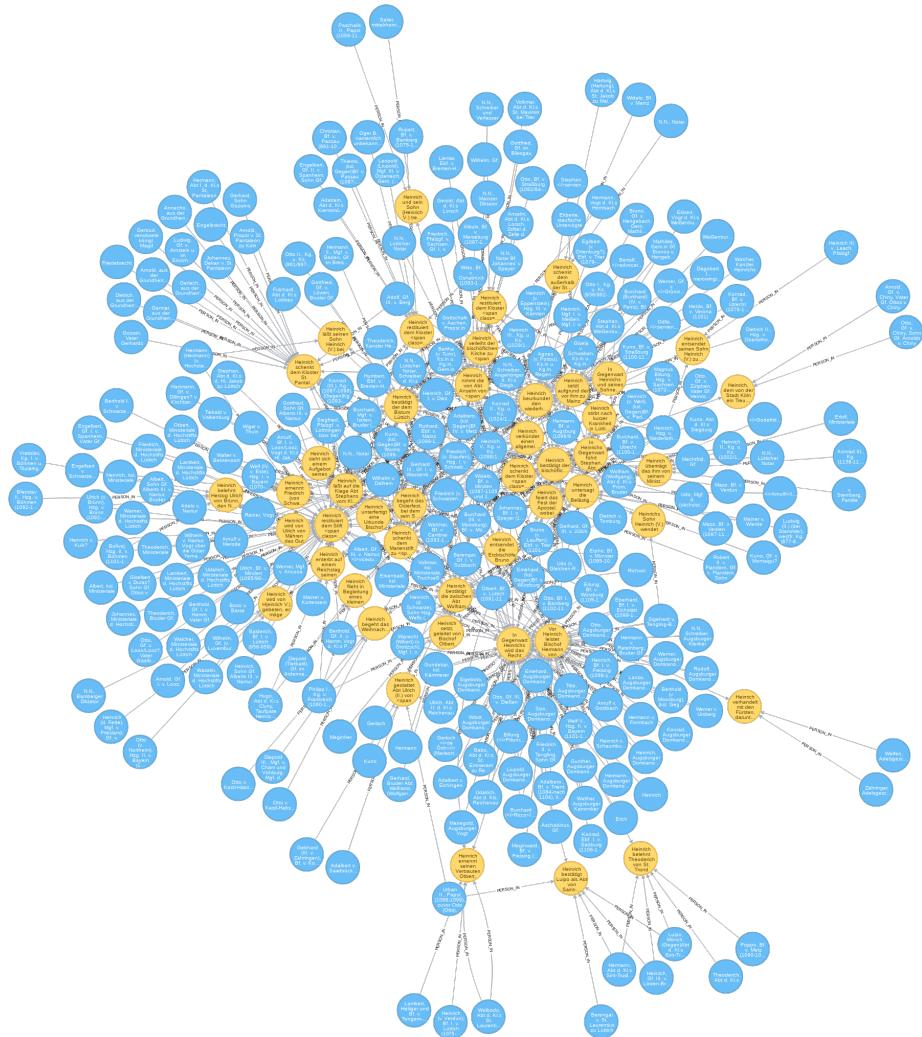


Abbildung 18: Robert mit Personen, die wiederum mit Personen gemeinsam in Regesten genannt sind.

```

// Robert II. von Flandern und Herzog Heinrich von Niederlothringen mit Netzwerk
MATCH
(n:IndexPerson)-[:PERSON_IN]->
(r:Regesta)<-[PERSON_IN]-(m:IndexPerson)
WHERE n.registerId = 'H4P01822'
AND m.registerId = 'H4P00926'
RETURN *;

// Rausrechnen der dazwischenliegenden Knoten
MATCH
(startPerson:IndexPerson)-[:PERSON_IN]->
(regest:Regesta)<-[PERSON_IN]-(endPerson:IndexPerson)
WHERE startPerson.registerId in ['H4P01822', 'H4P00926']
WITH startPerson, endPerson, count(regest) as anzahl,
collect(regest.ident) as idents
CALL apoc.create.vRelationship(startPerson, "KNOWS",
{anzahl:anzahl, regesten:idents}, endPerson) YIELD rel
RETURN startPerson, endPerson, rel

// Liste der Regesten als Ergebnis
MATCH
(startPerson:IndexPerson)-[:PERSON_IN]->
(regest1:Regesta)<-[PERSON_IN]-(middlePerson:IndexPerson)
-[:PERSON_IN]->(regest2:Regesta)
-<-[PERSON_IN]-(endPerson:IndexPerson)
WHERE startPerson.registerId in ['H4P00926']
AND endPerson.registerId in ['H4P01822']
RETURN DISTINCT startPerson.name1, regest1.ident, regest1.text,
middlePerson.name1, regest2.ident, regest2.text, endPerson.name1;

```

10.2 Herrscherhandeln ausgezählt

```

// Herrscherhandeln ausgezählt
MATCH (n:Lemma)<-[:ACTION]-(m:Regesta)
RETURN n.lemma, count(m) as ANZAHL ORDER BY ANZAHL desc LIMIT 25;

```

10.3 Herrscherhandeln pro Ausstellungsort ausgezählt

```

// Herrscherhandeln pro Ausstellungsort
MATCH (n:Lemma)<-[:ACTION]-(r:Regesta)-[:PLACE_OF_ISSUE]->(p:Place)
WHERE p.name <> '-' AND
p.name <> ''
RETURN p.name, n.lemma, count(r) as ANZAHL ORDER BY ANZAHL desc LIMIT 25;

```

10.4 Welche Literatur wird am meisten zitiert

```
MATCH (n:Reference)-[r:REFERENCES]-(m:Regesta)
RETURN n.title, count(r) as ANZAHL ORDER BY ANZAHL desc LIMIT 25;
```

10.5 Herrscherhandeln und Anwesenheit

```
MATCH (p:IndexPerson)-[:PERSON_IN]-(r:Regesta)-[:ACTION]-(l:Lemma)
RETURN p.name1, l.lemma, count(l) AS Anzahl ORDER BY p.name1, Anzahl DESC;
```

10.6 Regesten 200 km rund um Augsburg

```
// Entfernung von Orten berechnen lassen
MATCH (n:Place)
WHERE n.normalizedGerman = 'Augsburg'
WITH n.latLong as point
MATCH (r:Regesta)
WHERE distance(r.latLong, point) < 200000
AND r.placeOfIssue IS NOT NULL
AND r.placeOfIssue <> 'Augsburg'
RETURN r.ident, r.placeOfIssue,
distance(r.latLong, point) AS Entfernung
ORDER BY Entfernung;
```

- 10.7 [^f663]: Die nun folgenden Abfragen sind zum Teil einer Präsentation entnommen, die für die Summerschool der Digitalen Akademie im Rahmen des Mainz entwickelt wurden. Die Präsentation findet sich unter der URL <https://digitale-methodik.adwmainz.net/mod5/5c/slides/graphentechnologien/RI.html>.

title: Import von strukturierten XML-Daten in neo4j layout: default order: 25
contents: true —

11 Inhalt

{::no_toc}

- Will be replaced with the ToC, excluding the “Contents” header {:toc}

12 Import von strukturierten XML-Daten in neo4j

In diesem Kapitel wird der Import von strukturierten XML-Daten in die Graph-datenbank neo4j beschrieben. Strukturiert meint hierbei, dass es sich nicht um Text handelt, der beispielsweise in TEI-XML ausgezeichnet ist, sondern um Daten in einer datenbank-ähnlichen Struktur. Die Daten stammen aus einem Projekt meines Kollegen Thomas Kollatz, der sie mir freundlicherweise zur Verfügung gestellt hat. Ziel des Kapitels ist es, die Struktur der XML-Daten im Graphen nachzubilden und anschließend den Import durchzuführen.

12.1 Das XML-Beispiel

In der folgenden Abbildung wird ein Auszug aus den Daten gezeigt.

```
<collection>
  <work id="1">
    <title>Be'ur millot ha-higgajon</title>
    <autor>[1] Mose ben Maimon</autor>
    <autor>[2] Nieto, David ben Pinchas</autor>
    <kommentator>Mendelssohn, Moses</kommentator>
    <druckort>Berlin</druckort>
  </work>
  <work id="2">
    <title>Be'ur millot ha-higgajon</title>
    <autor>Mose ben Maimon</autor>
    <kommentator>Mendelssohn, Moses</kommentator>
    <druckort>Berlin</druckort>
  </work>
  <work id="3">
    <title>Be'ur millot ha-higgajon</title>
    <autor>Mose ben Maimon</autor>
    <kommentator>Mendelssohn, Moses</kommentator>
    <druckort>Berlin</druckort>
  </work>
```

Abbildung 19: Auszug aus dem XML-Beispiel (Quelle: Kuczera)

Das root-Element der XML-Datei ist `<collection>`. Innerhalb der `collection` finden sich Angaben zu verschiedenen Büchern. Zu jedem Buch werden folgende

Angaben gemacht:

- Titel des Buches
- Autor(en) des Buches
- Kommentator des Buches
- Druckort des Buches

12.2 Knotentypen

Für die Modellierung dieser Datenstruktur in der Graphdatenbank müssen zunächst die verschiedenen Knotentypen festgelegt werden. Als erstes scheint es sinnvoll einen Knoten vom Typ **Werk** anzulegen, wie es auch im XML über das **<work>**-Element modelliert ist. Die dem **<work>**-Element untergeordneten Elemente **<title>**, **<autor>**, **<kommentator>** und **<druckort>** können dann als Properties des **<work>**-Knotens angelegt werden. Damit wären alle Informationen des XMLs auch im Graphen gespeichert.

Die Graphmodellierung geht hier jedoch einen Schritt weiter. In einem ersten Schritt sind die in den Daten vorhandenen Entitäten zu identifizieren. Neben dem oben schon genannten Knoten vom Typ **Werk** ergeben sich noch Knoten vom Typ **Person** und **Ort**. In den **Ort**-Knoten werden die Druckorte abgelegt, die Angaben zu Autoren und Kommentatoren werden in den **Person**-Knoten gespeichert, da es sich ja um Personen handelt, die aber je nach Situation verschiedene Rollen einnehmen können.

12.3 Kantentypen

Nach den Knotentypen sind nun die Kantentypen festzulegen. Sie geben an, in welcher Beziehung die verschiedenen Knoten zueinander stehen. Sieht man sich die XML-Vorlage an, ergeben sich folgende Typen von Kanten:

- GEDRUCKT_IN
- AUTOR_VON
- KOMMENTIERT_VON

Mit der **GEDRUCKT_IN**-Kante werden ein Werk und ein Ort verbunden und damit angegeben, dass dieses Buch in jenem Ort gedruckt worden ist.

Die **AUTOR_VON**-Kante verbindet einen Personenknoten mit einem Werkknoten und ordnet damit den Autor dem von ihm geschriebenen Buch zu.

Mit der **KOMMENTIERT_VON**-Kante wird auch ein Personenknoten einem Werkknoten zugeordnet, diesmal nimmt die Person aber die Rolle des Kommentierenden ein.

Im der folgenden Abbildung werden alle Knoten und Kanten des Beispiels gemeinsam dargestellt.

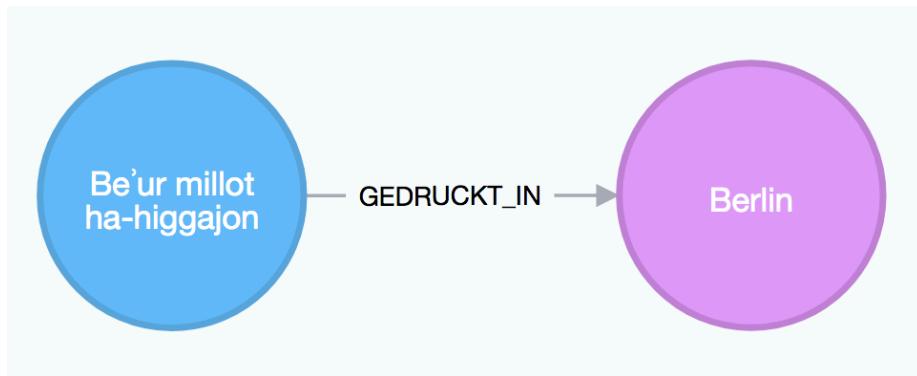


Abbildung 20: Verbindung zwischen einem Werk- und einem Ort-Knoten (Quelle: Kuczera).

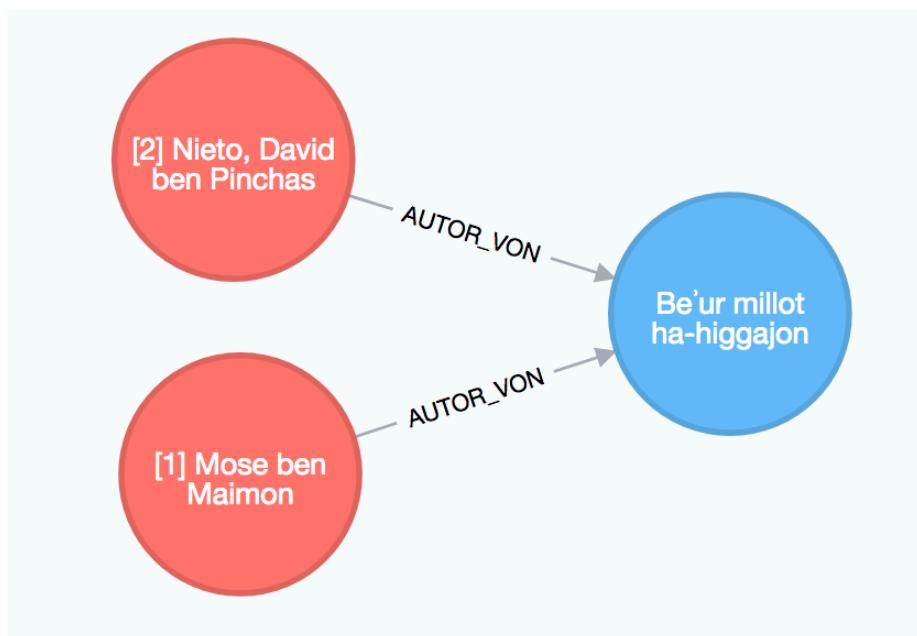


Abbildung 21: Verbindung zwischen einem Werk- und einem Ort-Knoten (Quelle: Kuczera).



Abbildung 22: Verbindung zwischen einem Werk- und einem Ort-Knoten (Quelle: Kuczera).

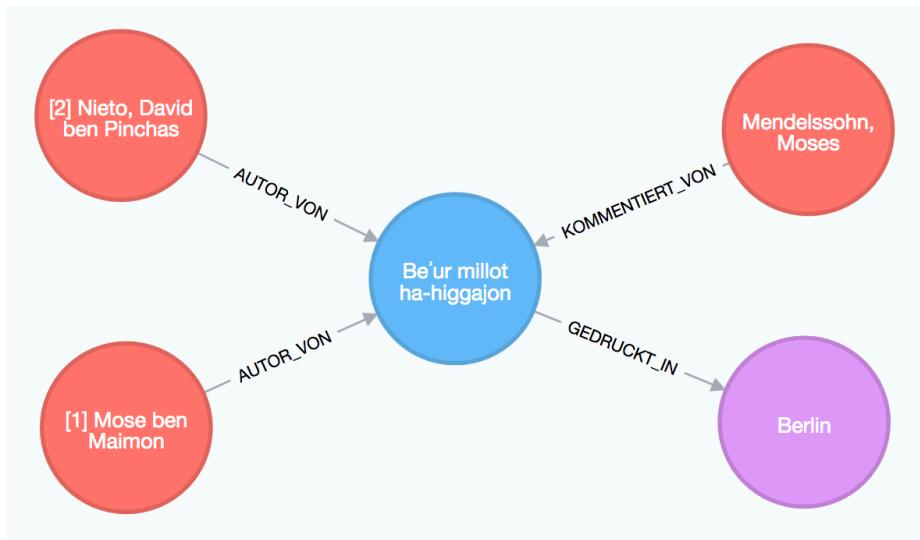


Abbildung 23: Verbindung zwischen einem Werk- und einem Ort-Knoten (Quelle: Kuczera).

Damit steht das Graphmodell fest und im nächsten Abschnitt geht es an den Import.

12.4 Der Import mit apoc.load.xmlSimple

Für den Import von XML-Daten steht in der apoc-Bibliothek der Befehl apoc.load.xmlSimple zur Verfügung. Im folgenden wird zunächst der gesamte Befehl für den Import gelistet.

```
CALL apoc.load.xmlSimple("https://docs.google.com/
document/u/0/export?format=txt&id=1Ujx9cdequEvuXx6
DxK7xfxbuS63Aq9n_xpLOAcjU_xc&token=AC4w5Vj0yTkdPLob1lo1ajkaG75fynnZ0Q%3A1490694667158") yield value
UNWIND value._work as wdata
    MERGE (w1:Werk{eid:wdata.id})
    set w1.name=wdata._title._text
    FOREACH (name in wdata._autor |
        MERGE (p1:Person {Name:name._text})
        MERGE (p1)-[:AUTOR_VON]->(w1) )
    FOREACH (name in wdata._kommentator |
        MERGE (p1:Person {Name:name._text})
        MERGE (p1)-[:KOMMENTIERT_VON]->(w1))
    FOREACH (druckort in [x in
        wdata._druckort._text where x is not null] |
        MERGE (o1:Ort{name:druckort})
        MERGE (w1)-[:GEDRUCKT_IN]->(o1));
```

13 Inhalt

{::no_toc}

- Will be replaced with the ToC, excluding the “Contents” header {:toc}

14 Das DTA im Graphen

Das Deutsche Textarchiv (DTA) stellt einen Disziplinen übergreifenden Grundbestand deutscher Werke aus dem Zeitraum von ca. 1600 bis 1900 im Volltext und als digitale Faksimiles frei zur Verfügung und bereitet ihn so auf, dass er über das Internet in vielfältiger Weise nutzbar ist. Das DTA-Korpus soll in größtmöglicher Breite widerspiegeln, was an bedeutenden Werken in deutscher Sprache veröffentlicht wurde. Die ausgewählten Texte stehen repräsentativ für die Entwicklung der deutschen Sprache seit der Frühen Neuzeit. Alle DTA-Texte werden unter einer offenen Lizenz veröffentlicht (CC BY-NC). Das DTA fördert

die Wiederverwendung seiner Texte in allen Bereichen der Digitalen Geisteswissenschaften.

15 Die Downloadformate des DTA

Das DTA bietet zu den bereitgestellten Texten verschiedene Formate zum Download an.

- TEI-P5
- TCF
- Plain-Text

Für den Import in eine Graphdatenbanken bietet sich das TCF-Format an, da es den Text tokenisiert, serialisiert, lemmatisiert und normalisiert bietet. In diesem Format lässt er sich mit cypher-Befehlen in die Graphdatenbank importieren. Im Beispiel wird Goethes Faust in der TCF-Fassung in die Graphdatenbank importiert.

Beispiel aus der XML-Datei

16 Vorbereitungen

Als Vorbereitung müssen einige Constraints eingerichtet werden.

```
create constraint on (t:Token) assert t.id is unique;
create constraint on (s:Sentence) assert s.id is unique;
create constraint on (l:Lemma) assert l.text is unique;
```

Mit den Befehlen wird sichergestellt, dass die im nächsten Schritt importierten Knoten eindeutige IDs haben.

17 Import

17.1 Tokenimport

Nun folgt der Import-Befehl mit der apoc-procedure *apoc.load.xmlSimple*.

```
call apoc.load.xmlSimple('http://deutschestextarchiv.de/book/download_fulltcf/16181') yield
unwind doc._TextCorpus._tokens._token as token
create (t:Token{id:token.ID, text:token._text})
with collect(t) as tokens
unwind apoc.coll.pairs(tokens)[0..-1] as value
with value[0] as a, value[1] as b
create (a)-[:NEXT_TOKEN]->(b);
```

In der ersten Zeile wird der apoc-Befehl `apoc.load.xmlSimple` aufgerufen, der als Argument die URL der TCF-Version von Goethes Faust im Deutschen Textarchiv erhält. Die weiteren cypher-Befehle parsen die XML-Datei und spielen die Token (also die einzelnen Wörter) als Wortknoten in die Graphdatenbank ein. Schließlich werden die NEXT_TOKEN-Kanten zwischen den eingespielten Wörtern erstellt.

17.2 Satzstrukturen

Der nächste Befehl lädt wieder die gleiche XML-Datei und importiert die Satzstrukturen.

```
call apoc.load.xmlSimple("http://deutschesetextarchiv.de/book/download_fulltcf/16181") yield
unwind doc._TextCorpus._sentences._sentence as sentence
match (t1:Token{id:head(split(sentence.tokenIDs, " "))})
match (t2:Token{id:last(split(sentence.tokenIDs, " "))})
create (s:Sentence{id:sentence.ID})
create (s)-[:SENTENCE_STARTS]->(t1)
create (s)-[:SENTENCE_ENDS]->(t2)
with collect(s) as sentences
unwind apoc.coll.pairs(sentences)[0..-1] as value
with value[0] as a, value[1] as b
create (a)-[:NEXT_SENTENCE]->(b);
```

17.3 Lemmimport

Im folgenden Befehl werden die Lemmata importiert und jedes Token mit dem zugehörigen Lemma verknüpft.

```
call apoc.load.xmlSimple('http://deutschesetextarchiv.de/book/download_fulltcf/16181') yield
unwind doc._TextCorpus._lemmas._lemma as lemma
match (t:Token{id:lemma.tokenIDs})
merge (l:Lemma{text:lemma._text})
create (t)-[:LEMMATISIERT]->(l);
```

Der letzte Befehl ergänzt bei jedem Token-Knoten noch die Lemma-Information als Proptery.

```
call apoc.load.xmlSimple('http://deutschesetextarchiv.de/book/download_fulltcf/16181') yield
unwind doc._TextCorpus._lemmas._lemma as lemma
match (t:Token{id:lemma.tokenIDs}) set t.Lemma = lemma._text;
```

Damit ist nun die Fassung von Goethes Faust aus dem Deutschen Textarchiv in die Graphdatenbank importiert worden und kann weiter untersucht werden (hier klicken, um den Code mit den cypher-Querys für den gesamten Artikel herunterzuladen).

17.4 Neo4j-Browser

Der neo4j-Graphbrowser bietet einen ersten Überblick zu den in der Datenbank vorhandenen Knoten- und Kantentypen. In Abbildung 3 ist der Wortknoten Tragödie ausgewählt, so dass in der Fußleiste der Graphvisualisierung die verschiedenen Eigenschaften des Knotens angezeigt werden. Der Knoten hat beispielsweise die “45414”, die Eigenschaft Lemma hat den Wert “Tragödie” während die Eigenschaft text das Originalwort enthält.

17.5 Beispielabfrage

Bei Cypher-Abfragen können alle Eigenschaften von Knoten und Kanten mit einbezogen werden.

```
match w=()-[:NEXT_TOKEN*5]->(a:Token{Lemma:'Gestalt'})-[:NEXT_TOKEN*5]->() return *;
```

Die folgende Abbildung zeigt das Ergebnis der Abfrage nach allen Vorkommen von Wörtern mit dem Lemma Gestalt im Faust mit den jeweils vorhergehenden und anschließenden drei Wörtern.

Abbildung mit Gestalt.

18 Zusammenfassung

18.1 Im vorliegenden Kapitel wurden die Schritte für den Import der DTA-TCF-Fassung von Goethes Faust in die Graphdatenbank neo4j vorgestellt. Die qualitativ hochwertigen Text-Quellen des Deutschen Textarchivs bieten in Verbindung mit Graphdatenbanken sehr interessante neue Möglichkeiten zur Auswertung der Texte. Durch Austausch des Links zur TCF-Fassung können auch andere Texte des DTA eingespielt werden.

title: XML-Text im Graphen layout: default order: 60 contents: true —

19 Inhalt

{::no_toc}

- Will be replaced with the ToC, excluding the “Contents” header {::toc}

20 Textmodelle im Graphen

20.1 Text als Graph

Die Diskussion über Modellierungsansätze von Text als Graph hält aktuell an.¹⁰

20.1.1 XML-Dateien als Ketten von Zeichen

Da die technische Grundlage von XML Textdateien sind, handelt es sich bei XML um eine eindimensionale Kette von Tokens¹¹.

20.1.2 Modellierungsüberlegungen

Prinzipiell können XML-Dateien ohne größere Probleme in einen Graphen importiert werden, da sie einen geerdeten, gerichteten azyklischen Graphen, der vielfache Elternbeziehungen verhindert, und damit ein Ordered Hierarchy of Content Objects (OHCO) darstellen. Es gibt vor allem im Bereich des Mixed-Content verschiedene Ansätze, XML-Strukturen im Graphen abzubilden¹². Überlegungen zur Auslagerung von Annotationen aus XML in eine Graphdatenbank brachte schon Desmond Schmidt in die Diskussion ein:

Embedded annotations can also be removed from TEI texts. The elements <note>, <interp>, and <interpGrp> describe content that, like metadata, is about the text, not the text itself. These are really annotations, and should ideally be represented via the established standards and practices of external annotation (Hunter and Gerber 2012). Annotations are stored in triple stores or graph databases like Neo4J,²⁰ which record the identifiers of each component of the annotation and its data¹³.

20.1.2.1 Granularität des Modells – Was ist ein Token ?

Für den Bereich der historisch-kritischen und philologischen Editionen ist es in der Regel ausreichend, beim Import von XML-kodierten Texten in den Graphen jeweils ein Wort in einen Knoten zu importieren, da meist die historische Aussage der Quelle im Vordergrund steht. In anderen Bereichen der digitalen Geisteswissenschaften kann die Entscheidung, welche Einheit für den Import in einen Knoten gewählt wird, durchaus anders ausfallen. So ist für Philologien die

¹⁰Vgl. zuletzt @DekkerHaentjensItmorejust2017.

¹¹@HuitfeldtMarkupTechnologyTextual2014, S. 161 sieht digitale Dokumente prinzipiell als lineare Sequenz von Zeichen

¹²Vgl. @DekkerHaentjensItmorejust2017.

¹³@SchmidtInteroperableDigitalScholarly2014, 4.1 Annotations.

Betrachtung auf Buchstabenebene interessant¹⁴. Im Graphmodell ist man im Hinblick auf die Granularität des Datenmodells wesentlich flexibler als z.B. bei XML oder Standoff-Markup. So ist es beispielsweise denkbar, an einen Wortknoten eine weitere Kette von Knoten anzulagern, welche pro Knoten jeweils einen Buchstaben des Wortes und die zugehörigen Annotationen enthalten. Es handelt sich um einen buchstabenbasierten Sub-Graphen, dessen Anfang und Ende mit dem Wortknoten verbunden ist. Damit können verschiedene Granularitätsstufen in einem Modell und in einer Datenbank abgebildet werden.

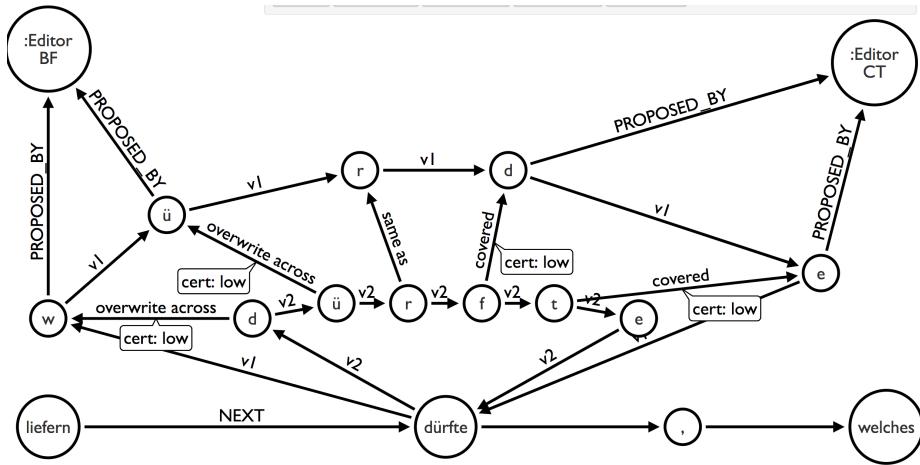


Abbildung 24: Granularität von Text im Graphen

20.2 Technische Vorbemerkungen

20.2.1 Die Graphdatenbank neo4j

Technische Grundlage der im Rahmen dieses Werkes vorgestellten Beispiel ist die Graphdatenbank neo4j¹⁵. Die Graphdatenbank gibt es unter Open-Source-Lizenz in einer Community-Edition und unter einer kommerziellen Lizenz in einer Enterprise-Edition. Für die hier vorliegenden Aufgabengebiete war die Leistungsfähigkeit der Community-Edition völlig ausreichend. Darüber hinaus hat sich neo4j in den letzten Jahren als eine der führenden Graphdatenbanken etabliert und bietet neben stabiler Funktionalität auch genormte Schnittstellen. Prinzipiell wurde darauf hingearbeitet, dass die hier vorgestellten Überlegungen zu Graphenmodellierung historisch-kritischer Editionen auch auf andere

¹⁴In FuD (<http://fud.uni-trier.de/>) werden Texte in Standoff-Markup auf Buchstabenebene ausgezeichnet, während beim DTA-Basisformat der Fokus auf der wortbasierten Auszeichnung liegt (vgl. <http://www.deutschesextarchiv.de/doku/basisformat/eeAllg.html>).

¹⁵Vgl. www.neo4j.com (abgerufen am 7.8.2017).

Property-Graphdatenbanken wie z.b. orientdb¹⁶ übertragbar sind.

20.2.2 Der neo4j-XML-Importer

Für den Import der Texte wurde der neo4j-XML-Importer von Stefan Armbruster verwendet¹⁷. Der Importer nimmt TEI-XML-Dateien entgegen und importiert sie in die Graphdatenbank.

20.2.2.1 Import

Befehl für den Import der Patzig-XML-Datei¹⁸:

```
CALL apoc.xml.import('http://www.deutschesetextarchiv.de/
    book/download_xml/patzig_msgermfol841842_1828',
    {createNextWorkRelationships: true})
    yield node return node;
```

Dabei werden die XML-Knoten in Graphknoten umgewandelt und verschiedene Arten von Kanten erstellt, die einerseits die Baum-Hierarchie des XMLs im Graphen abbilden und andererseits die im XML vorhandenen Textknoten miteinander verknüpfen¹⁹. Dabei werden die Wörter innerhalb der XML-Textknoten werden in Ketten von Wortknoten umgewandelt. Zur Abbildung des Wurzelement der importierten XML-Datei wird ein Knoten vom Typ `XmlDocument` angelegt. Dieser erhält die Propertys `_xmlEncoding` zur Darstellung des Encodings, `_xmlVersion` für die Xml-Version und `url` für die URL des importierten XML-Dokuments.

Mit einem weiteren cypher-Query erhalten alle der importierten Knoten die Eigenschaft `url` mit der URL des importierten XML-Dokuments. Damit lassen sich Knoten in einer Graphdatenbank mit mehreren importierten XML-Dokumenten auseinanderhalten.

```
MATCH (d:XmlDocument)-[:NEXT_WORD*]->(w:XmlWord)
SET w.url = d.url;
```

¹⁶Die Graphdatenbank orientdb (<http://orientdb.com/>) bietet ein für Historiker sehr interessantes Feature, da sie als Datentyp für die Propertys von Knoten auch Datumsangaben im ISO-Format zulässt. Vgl. <https://orientdb.com/docs/2.2/Managing-Dates.html> (abgerufen am 7.8.2017).

¹⁷Der generische XML-Import wird momentan von Stefan Armbruster entwickelt (<https://github.com/sarmbruster/> abgerufen am 23.11.2017). Es ist geplant, den Importer in die apoc-Bibliothek zu integrieren. (<https://github.com/neo4j-contrib/neo4j-apoc-procedures> abgerufen am 23.11.2017).

¹⁸Für die Vereinheitlichung des Druckbildes mussten an einigen Stellen Zeilenumbrüche in die Codebeispiele eingefügt werden, die deren direkte Ausführung behindern.

¹⁹In TEI-XML gibt es zwei verschiedene Arten von Elementen. Die eine Klasse dient der Klassifizierung von Text, die zweite Art bringt Varianten und zusätzlichen Text mit!!! Literaturhinweis ergänzen, Hans-Werner Bartz fragen.

Mit dem nächsten cypher-Query werden die Knoten des importierten XML-Dokuments durchnummertiert und der jeweilige Wert in der Property `DtaID` abgelegt.

```

MATCH p = (start:XmlDocument)-[:NEXT*]->(end:XmlTag)
WHERE NOT (end)-[:NEXT]->()
AND start.url = 'http://www.deutsches-textarchiv.de/book/download'
WITH nodes(p) as nodes, range(0, size(nodes(p))) AS indexes
UNWIND indexes AS index
SET (nodes[index]).DtaID = index;

```

20.2.2.2 Erläuterung der entstandenen Graphstrukturen

Nach Abschluss des Imports werden jetzt die importierten Datenstrukturen erläutert. In der folgenden Tabelle werden die verschiedenen Typen von Knoten erläutert, die während des Imports erstellt wurden.

Tabelle zum Importvorgang der XML-Elemente und den entsprechenden Knoten

XML-Knoten	Graphknoten	Bemerkungen
XML-Wurzelement	XmlDocument	Gibt es nur einmal. Es enthält Angaben zur Encodierung, zur XML-Version und die URL der importierten XML-Datei
XML-Element-Knoten	XmlTag-Knoten	Die Attribute des XML-Elements werden in entsprechende Propertys des XMLTag-Knotens in der Datenbank umgewandelt
XML-Text-Knoten	XmlWord	Jedes Wort des XML-Textknotens wird ein XmlWord-Knoten im Graphen

In der nächsten Tabelle werden die verschiedenen Kantentypen erläutert, mit einerseits die Serialität des XMLs (`NEXT`-Kanten), und die Hierarchie (`NEXT_SIBLING` und `IS_CHILD_OF`-Kanten), andererseits aber auch die Abfolge der Inhalte der XML-Textelemente (`NEXT_WORD`) dargestellt werden.

Tabelle zu den erstellen Kantentypen

Kante	Bemerkungen
:NEXT	Zeigt die Serialität der XML-Datei im Graphen
:NEXT_SIBLING	Zeigt auf den nächsten Graphknoten auf der gleichen XML-Hierarchie-Stufe
:NEXT_WORD	Zeigt auf das nächste Wort in einem XML-Textknoten
:IS_CHILD_OF	Zeigt auf den in der XML-Hierarchie übergeordneten Knoten

Kante	Bemerkungen
:FIRST_CHILD_OF	Zeigt vom ersten untergeordneten auf den übergeordneten Knoten.
:LAST_CHILD_OF	Zeigt vom letzten untergeordneten auf den übergeordneten Knoten.

Die folgende Abbildung zeigt einen kleinen Ausschnitt aus der TEI-XML-Datei der Patzig-Vorlesungsmitschrift.

```

rendition="#aq">po&x017F;thumi&x017F;chen</hi>
Werke auf-<lb/>
gedeckt u. <subst><del rendition="#s">&x017F;eine
Fehler</del><add
place="superlinear">die&x017F;e</add></subst> zum
Theil erka<supplied reason="damage"
resp="#BF">n&x0303;t,</supplied><lb/>
wen&x0303; er redete von häßlichen u.

```

Abbildung 25: XML-Beispiel aus der TEI-XML-Datei der Patzig-Vorlesungsmitschrift.

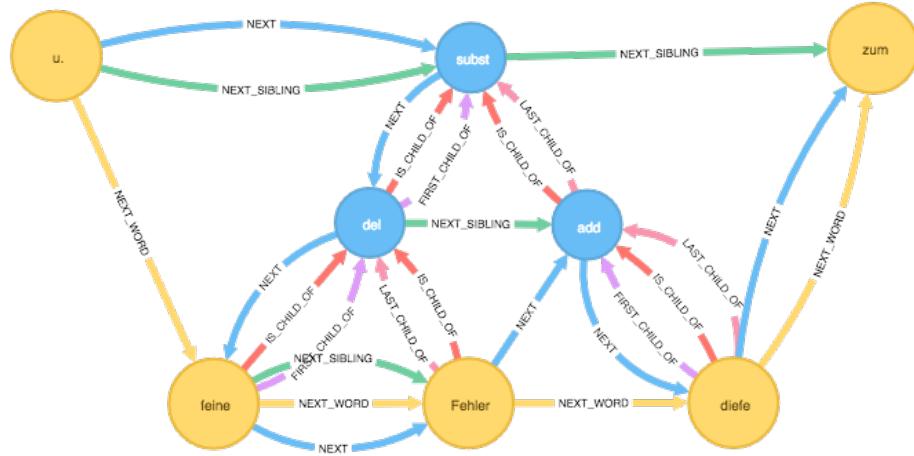


Abbildung 26: XML-Beispiel im Graphen.

Beim Import der XML-Datei in den Graphen die XML-Element-Knoten in XML-Tag-Knoten. In der Abbildung des XML-Ausschnittes sind jene Teile blau markiert, die sich auch in der Graphabbildung befinden. Aus Sicht der XML-Hierarchie befindet sich der XML-Textknoten mit dem Inhalt *gedeckt u.* auf der gleichen Ebene mit dem <subst>-Element. Dies wird beim

Mit dieser Modellierung lassen sich beispielsweise die von einem <add>-Element umfassten Wörter abfragen, in dem man ausgehend vom add-Knoten der FIRST_CHILD_OF-Kante rückwärts folgt, anschließend von diesem Knoten den NEXT_SIBLING-Kanten so lange folgt, bis wieder die LAST_CHILD_OF-Kante wieder zum add-Knoten zurückführt. Der entsprechende cypher-Query sieht wie folgt aus:

```

MATCH
(n:XmlTag {_name:'add'})
<-[:FIRST_CHILD_OF]-(w1:XmlWord)
-[:NEXT_WORD*..5]->(w2:XmlWord)
-[:LAST_CHILD_OF]->(n)
RETURN * LIMIT 25

```

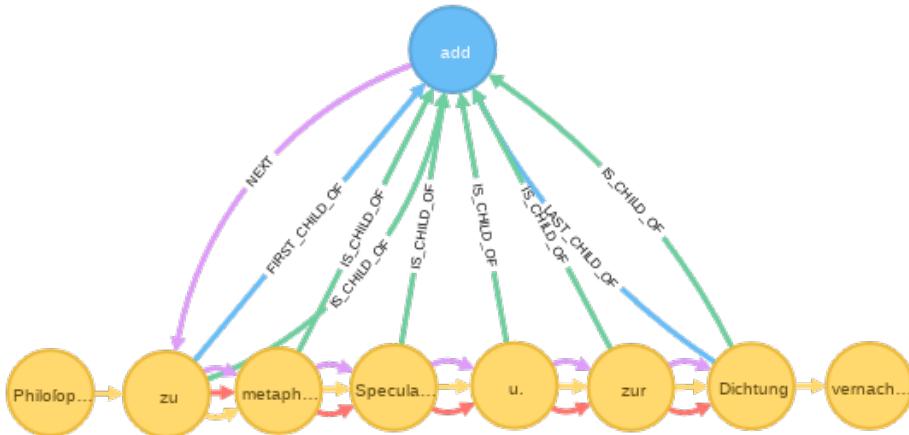


Abbildung 27: XML-Hierarchie eines <add>-Elements und der von ihm umfassten Wörter im Graphen.

In einem zweiten Schritt kann der so entstandene Graph mit Hilfe von cypher-Querys weiter bearbeitet werden. Die Graphdatenbank neo4j ist schemafrei und somit können nun über die importierten XML-Strukturen weitere Erschließungsstrukturen gelegt werden, ohne dass ein XML-Parser sich über das nicht mehr wohlgeformte XML beschwert. Zu beachten ist bei jedem Schritt, ob wieder der Schritt zurück nach XML getätigert werden soll. Sicherlich ist es kein größeres Problem, eine in eine Graphdatenbank importierte XML-Datei wieder als solche zu exportieren. Ist der Graph aber mit weiteren Informationen angereichert, so muss geklärt werden, ob, und wenn ja wie, diese zusätzlichen Informationen in wohlgeformtes XML transferiert werden können.

21 Das DTA-Basisformat im Graphen

Das DTA-Basisformat ist ein Subset der TEI und bietet für Textphänomene jeweils nur eine Möglichkeit der Auszeichnung. Damit wird die in der TEI vorhandene Flexibilität bei der Auszeichnung eingeschränkt, um damit einen höheren Grad an Interoperabilität zu erreichen. Das DTA-Basisformat folgt den P5-Richtlinien der TEI, trifft aber eine Tag-Auswahl der für die Auszeichnung historischer Texte notwendigen Elemente.

Im folgenden Abschnitt werden für ausgewählte Elemente des DTA-Basisformats mögliche Modellierungsformen im Graphen beschrieben. Zum äußereren Erscheinungsbild wird der Seitenfall sowie Spalten- und Zeilenumbrüche berücksichtigt. Bei den Textphänomenen werden Absätze, Schwer- und Unleserliches und inhaltlich werden die Kapiteleinteilung, inhaltliche Inline-Auszeichnungen und editorische Eingriffe behandelt. Für die Metadaten werden keine Modellierungsvorschläge formuliert, da diese sich sauber im XML-Baum darstellen lassen und keine Überlappungsprobleme etc. entstehen.

21.1 Strukturen des Dokuments

21.1.1 Graphenmodellierung von Zeilen

Nehmen wir als Beispiel Zeilenwechsel auf einer Seite des Patzig-Manuskripts (http://www.deutschesextarchiv.de/book/view/patzig_msgermfol841842_1828/?hl=Himalaja&p=39).²⁰

```
... Die<lb/>
Jnder hegen die große Verehrung vor<lb/>
dem Himalaja Gebirge. ...
```

Im Graphen sieht die Stelle wie folgt aus:

Das leere <lb>-Element steht für die Markierung eines Zeilenanfangs (*line begins*). Der Graph soll nun so umgebaut werden, dass die Zeile durch einen line-Knoten gekennzeichnet wird, von dem aus eine FIRST_CHILD_OF-Kante mit dem ersten Wort der Zeile und eine LAST_CHILD_OF-Kante mit dem letzten Wort der Zeile verbunden ist.

Mit dem folgenden cypher-query kommt man den auf der Abbildung sichtbaren Subgraphen:

```
MATCH (n0:XmlWord)-[:NEXT_WORD]->
(n1:XmlWord {DtaID:10272})-[:NEXT_WORD*..8]->(n2:XmlWord),
(n1)<-[:NEXT]-(t1:XmlTag {_name:'lb'}),
(n3:XmlWord {DtaID:10277})-[:NEXT]->(t2:XmlTag {_name:'lb'})
RETURN * LIMIT 20;
```

²⁰URL des Beispieltexes: http://www.deutschesextarchiv.de/book/view/patzig_msgermfol841842_1828/?hl=Himalaja&p=39 abgerufen am 02.01.2018.

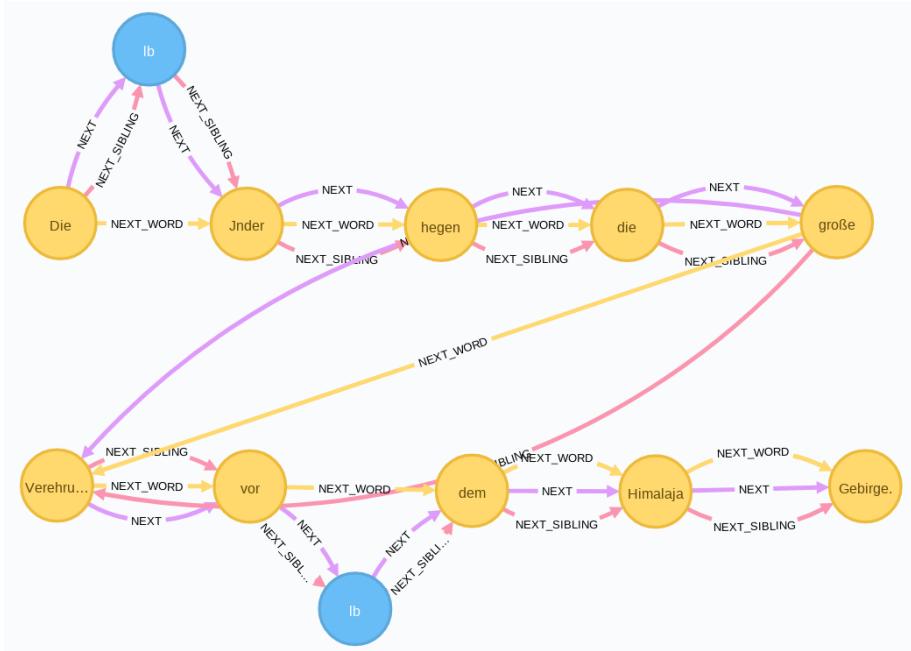


Abbildung 28: <1b/>-Element im Graphen

Im folgenden Schritt wird ein **line**-Knoten erzeugt, der die Zeile darstellen soll. Mit diesem werden dann das erste und das letzte Wort der Zeile verbunden.

```

MATCH (n0:XmlWord)-[:NEXT_WORD]->
(n1:XmlWord {DtaID:10272})-[:NEXT_WORD*..8]->(n2:XmlWord),
(n1)<-[:NEXT]-(t1:XmlTag {_name:'lb'}),
(n3:XmlWord {DtaID:10277})-[:NEXT]->(t2:XmlTag {_name:'lb'})
MERGE (n3)<-[:LAST_CHILD_OF]-(l:line {_name:'line'})-[:FIRST_CHILD_OF]->(n1)
DETACH DELETE t1, t2
RETURN * LIMIT 20;

```

Im Graphen sieht die Stelle wie folgt aus:

21.1.2 Zeilenwechsel mit Worttrennungen

Nun kommt es im Bereich der Zeilenwechsel sehr häufig zu Worttrennungen. Als Beispiel nehmen wir folgende Zeile, die sich auf der gleichen Seite wo das eben behandelte Beispiel befindet:

```

... Die Ken&#x0303;t-<1b/>
niß des Jahres durch nicht von einer Nation<1b/>
auf ...

```

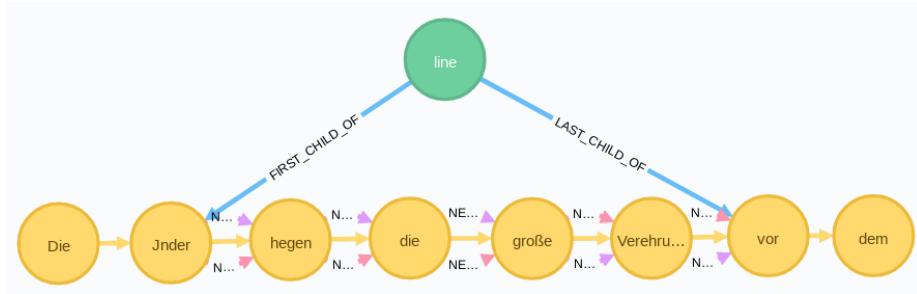


Abbildung 29: <1b/>-Element im Graphen

Im Graphen sieht die Stelle wie folgt aus:

Mit dem folgenden cypher-query kommt man den auf der Abbildung sichtbaren Supgraphen:

```

MATCH (n0:XmlWord {DtaID:10197})-[:NEXT_WORD]->
(n1:XmlWord)-[:NEXT_WORD*..9]->(n2:XmlWord),
(n1)-[:NEXT]->(t1:XmlTag {_name:'1b'}),
(n3:XmlWord {DtaID:10207})-[:NEXT]->(t2:XmlTag {_name:'1b'})
RETURN * LIMIT 20;

```

Das <1b/>-Element trennt das Wort *Kenntniß*²¹. Im nächsten Schritt werden nun die beiden getrennten Wortknoten *Kennt-* und *niß* im zweiten Wortknoten *niß* zusammengefasst. Der erste Wortknoten *Kennt-* inkl. seiner Kanten wird gelöscht und eine neue NEXT-Kante zwischen dem *niß*-Wortknoten und dem vorhergehenden *Die*-Wortknoten erstellt. Die Informationen, an welcher Stelle das Wort getrennt war, wird in den Eigenschaften des neuen *Kenntniß*-Wortknotens gespeichert. In der Eigenschaft **before** steht dann der Inhalt des ursprünglich ersten Wortknotens *Kennt-* und in der Eigenschaft **after** der Inhalt des ursprünglich zweiten Wortknotens *niß*.

Hier werden die notwendigen Cypher-Befehle angezeigt:

```

MATCH (n0:XmlWord {DtaID:10197})-[:NEXT_WORD]->
(n1:XmlWord {DtaID:10198})-[:NEXT_WORD]->
(n2:XmlWord {DtaID:10200})-[:NEXT_WORD*..8]->(n3:XmlWord {DtaID:10207}),
(n1)-[:NEXT]->(t1:XmlTag {_name:'1b'}),
(n4:XmlWord {DtaID:10207})-[:NEXT]->(t2:XmlTag {_name:'1b'})
SET n2.before = left(n1.text, size(n1.text)-1)
SET n2.after = n2.text
SET n2.text = left(n1.text, size(n1.text)-1)+n2.after
MERGE (n0)-[:NEXT_WORD]->(n2)
MERGE (n4)<-[ :LAST_CHILD_OF ]-(l:line {_name:'line'})-[:FIRST_CHILD_OF]->(n2)
DETACH DELETE t1, t2, n1

```

²¹Zur einfacheren Lesbarkeit wurden im Wort *Kenntniß* die Sonderzeichen normalisiert.

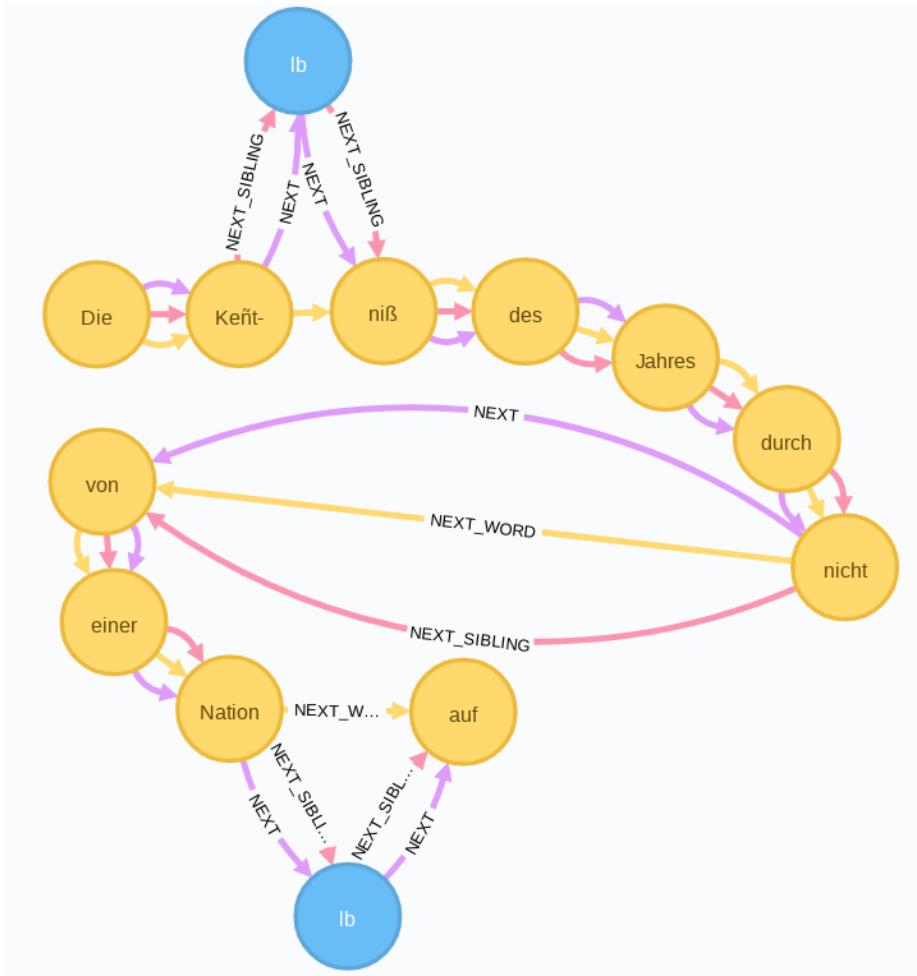


Abbildung 30: <1b/>-Element im Graphen

```
RETURN * LIMIT 20;
```

Im Graphen ergibt sich anschließend folgendes Bild:

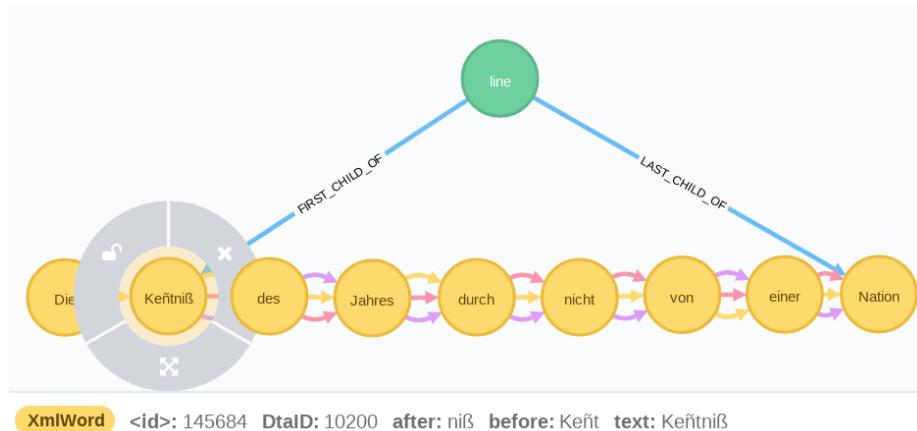


Abbildung 31: <1b>-Element im Graphen herausgenommen, Wortknoten zusammengefasst

Am unteren Bereich der Abbildung sind in der Legende die Propertys des Wortknotens *Kentniñ* hervorgehoben. Dort erkennt man die vorher vorhandenen Wortbestandteile und den neuen Wert der Property *text*.

21.1.3 Seitenzahlen und Faksimilezählung

Im DTA-Bf wird jeweils der Anfang einer Seite mit dem leeren Element <pb> markiert²². Das leere Element kann noch die Attribute **facs** für die Zählung der Faksimileseiten und **n** für die auf der Seite ggf. angegebene Seitenzahl enthalten.

```
<pb facs="#f[Bildnummer]" n="[Seitenzahl]" />
```

Ist eine Seitenzahl im Faksimile falsch wiedergegeben, so wird diese originalgetreu übernommen und die richtige Seitenzahl in eckigen hinzugefügt in das **n**-Attribut übernommen.

```
<pb facs="#f[Bildnummer]" n="[fehlerhafte Seitenzahl [korrigierte Seitenzahl]]" />
```

Das <pb>-Element auf den Seiten 5 und 6 aus Patzig (http://www.deutschestextarchiv.de/book/view/patzig_msgermfol841842_1828/?p=5)

```
... Abwe&#x017F;enheit vom heimi&#x017F;chen Boden ent-<1b/>
<note place="left"><figure type="stamp"/><1b/>
```

²²Vgl. die Dokumentation des DTA-Basisformats unter <http://www.deutschestextarchiv.de/doku/basisformat/seitenFacsnr.html> abgerufen am 25.11.2017.

²³Die Beispelseite findet sich unter http://www.deutschestextarchiv.de/book/view/patzig_msgermfol841842_1828/?p=5 abgerufen am 25.11.2017.

```

</note>fernt hielt, der &#x017F;ich viel mit einem Volke<lb/>
<fw place="bottom" type="catch">befreun-</fw><lb/>
<pb facs="#f0006" n="2."/>
befreundete, welches durch den
...
in einzelnen großen Zügen zu ent-<lb/>
werfen</hi>.</p><lb/>
<fw place="bottom" type="catch">Nachdem</fw><lb/>
<pb facs="#f0007" n="3."/>
<p><note place="left"><hi rendition="#u">Neue&#x017F;te A&#x017F;tronomi&#x017F;che Ent-<lb/>
deckungen.</hi><lb/> ...

```

Im Graphen findet man das <pb>-Element der Seite 6 mit folgendem Query[^a825]:

```

MATCH
(n1:XmlWord {DtaID:869})-[:NEXT]->
(lb1:XmlTag {_name:'lb'})-[:NEXT]->
(t2:XmlTag {_name:'fw', place:'bottom', type:'catch'})-[:NEXT_SIBLING]->
(lb2:XmlTag {_name:'lb'})-[:NEXT_SIBLING]->
(pb:XmlTag {_name:'pb'}),
(n1:XmlWord)-[nw1:NEXT_WORD]->
(n2:XmlWord)-[nw2:NEXT_WORD]->
(n3:XmlWord)-[nw3:NEXT_WORD]->
(n4:XmlWord),
(n2:XmlWord)-[:NEXT]->(t1:XmlTag {_name:'lb'})
RETURN * LIMIT 20;

```

Im Graphen ergibt sich folgendes Bild:

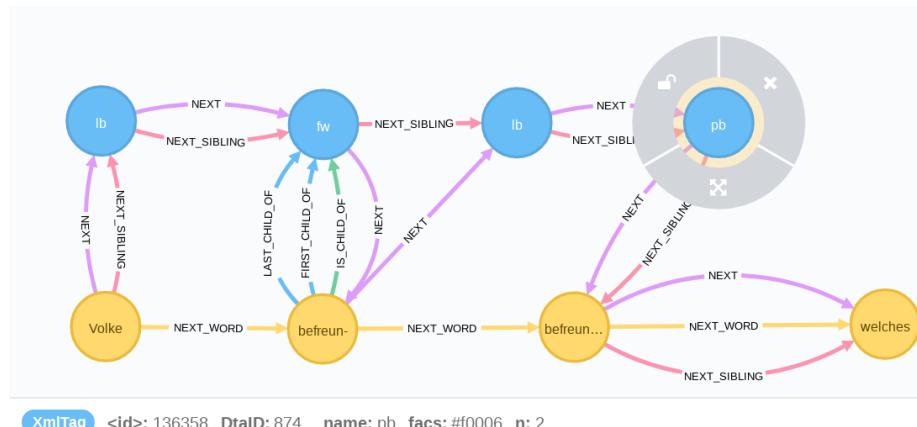


Abbildung 32: Der Pfad vom <pb/>-Element zum ersten Wort der Seite *befreundet*.

Markiert ist das <pb/>-Element der Seite 6. Im Fuß der Abbildung werden die Propertys des Elements angezeigt. Der Textfluss wird durch den Wortknoten **befreun-** unterbrochen, der eine Kustode darstellt. Diese soll aus dem Textfluss herausgelöst und direkt mit dem letzten Wortknoten **Volke** über die neu eingeführte **catch_words**-Kante verbunden werden. Der <fw>, und der <lb/>-Knoten werden gelöscht und der letzte Wortknoten der Seite über eine neue NEXT-Kante mit dem <pb/>-Knoten verknüpft.

Hier der Query für den Umbau:

```

MATCH
(n1:XmlWord {DtaID:869})-[:NEXT]-
(lb1:XmlTag {_name:'lb'})-[:NEXT]-
(t2:XmlTag {_name:'fw', place:'bottom', type:'catch'})-[:NEXT_SIBLING]-
(lb2:XmlTag {_name:'lb'})-[:NEXT_SIBLING]-
(pb:XmlTag {_name:'pb'}), 
(n1:XmlWord)-[nw1:NEXT_WORD]-
(n2:XmlWord)-[nw2:NEXT_WORD]-
(n3:XmlWord)-[nw3:NEXT_WORD]-
(n4:XmlWord),
(n2:XmlWord)-[:NEXT]->(t1:XmlTag {_name:'lb'})
DELETE nw1, nw2
DETACH DELETE t2
MERGE (n1)-[:NEXT_WORD]->(n3)
MERGE (n1)-[:CATCH_WORDS]->(n2)
MERGE (n1)-[:NEXT_WORD]->(n3)
MERGE (lb1)-[:NEXT]->(n2)
RETURN * LIMIT 20;

```

Im Graphen ergibt sich folgendes Bild:

Die Kustode ist nun nicht mehr über NEXT_WORD-Kanten mit dem Fließtext verknüpft, bleibt aber über die CATCH_WORDS-Kante mit dem letzten Wort der Seite verbunden. In einem zweiten Schritt müssen nun die beiden <pb/>-Elementknoten zu einem neu einzuführenden page-Knoten zusammengeführt werden. Hierfür lassen wir uns im nächsten cypher-Query alle <pb/>-Knoten mit einer DtaID kleiner als 875 anzeigen, da diese vor dem <pb/>-Knoten der Seite 6 mit der DtaID 874 liegen:

```

MATCH (n:XmlTag {_name:'pb'})
WHERE n.DtaID < 875
RETURN n;

```

Aus der Tabellenansicht ist zu entnehmen, dass Seite 5 von den <pb/>-Elementen mit der DtaID 562 und 874 eingefasst wird.

Der cypher-Query zum Einfügen des page-Knoten sieht wie folgt aus:

```

MATCH
(pb1:XmlTag {DtaID:562, _name:'pb'})-[n1:NEXT*..5]->(w1:XmlWord {DtaID:565}),

```

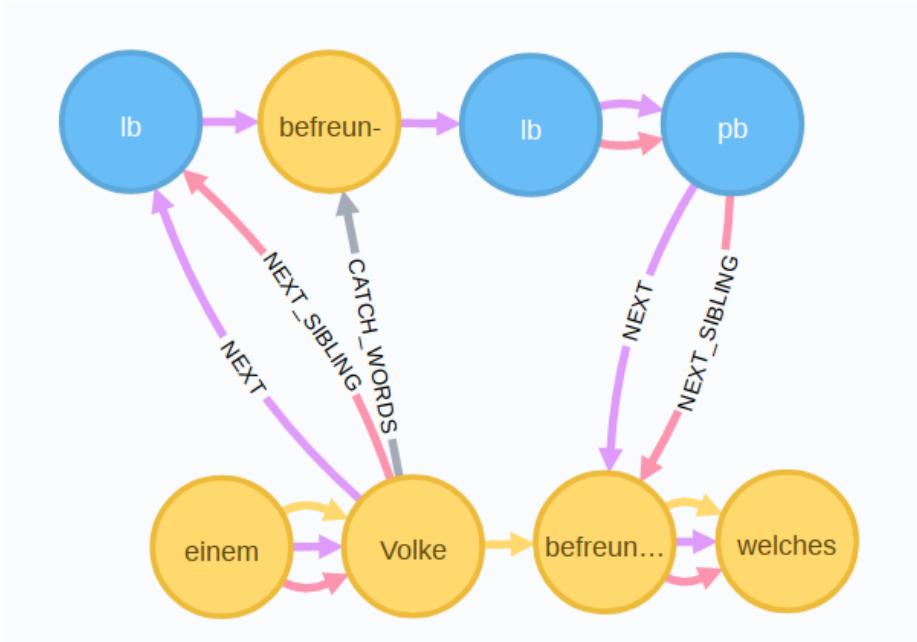


Abbildung 33: Die Kustode *befreun-* wird aus der NEXT_WORD-Textkette herausgenommen und über eine CATCH_WORDS-Kante mit dem Wortknoten *Volke* verknüpft.

"n"
{"_name": "pb", "fac": "#f0001", "DtaID": 444}
{"_name": "pb", "fac": "#f0002", "DtaID": 445}
{"_name": "pb", "fac": "#f0003", "DtaID": 455}
{"_name": "pb", "fac": "#f0004", "DtaID": 558}
{"_name": "pb", "fac": "#f0005", "DtaID": 562, "n": "1."}
{"_name": "pb", "fac": "#f0006", "DtaID": 874, "n": "2."}

Abbildung 34: Tabellenansicht aller <pb/>-Knoten mit einer DtaID kleiner als 875.

```
(pb2:XmlTag {DtaID:874, _name:'pb'})->[n2:NEXT*..5]-(w2:XmlWord {DtaID:872})
MERGE
(w1)-[:FIRST_CHILD_OF]-(page:page {facs:'#f0005', n:1})-[:LAST_CHILD_OF]->(w2)
RETURN pb1, w1, pb2, w2, page;
```

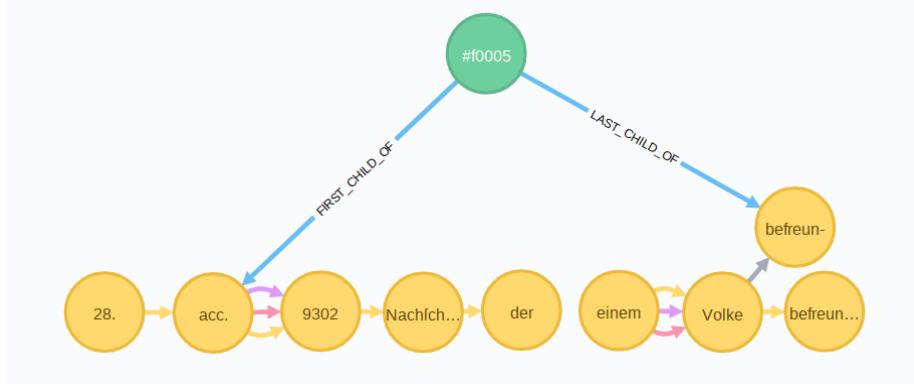


Abbildung 35: Die Seite wird modelliert mit dem `page`-Knoten #0005 der mit dem ersten Wort über eine `FIRST_CHILD_OF`- und mit dem letzten Wort der Seite über eine `LAST_CHILD_OF`-Kante verknüpft ist.²⁵

21.1.4 Absätze

Absätze werden im DTA-Basisformat mit dem `<p>`-Element eingefasst. Im Manuskript von Patzig finden sich insgesamt 238 mit dem `<p>`-Element eingefasste Textabschnitte²⁶.

```
<p>&#x017F;ie, <choice><orig><reg resp="#BF">Metereologie</reg></choice> u. Phy&#x017F;iologie<lb/>
der <choice><abbr>Pflanz&#xFFC;</abbr><expan resp="#BF">Pflanzen</expan></choice> haben mich den größten Theil<lb/>
meines Lebens be&#x017F;chäftigt u. &#x017F;o hoffe ich<lb/>
in die&#x017F;en Gegen&#x017F;tänden mich &#x017F;o deutlich<lb/>
zu mache, <choice><abbr>d&#x017F;</abbr><expan resp="#BF">doß</expan></choice> auch die mit mindern <choice><abbr>Vorken&#x0303;t;<lb/>
ni&#x017F;;&#x017F;8&#xFFC;</abbr><expan resp="#BF">Vorken&#x0303;t;<lb/>
ni&#x017F;;&#x017F;en</expan></choice> meinen Vorträgen folgen kön&#x0303;en:</p><lb/>
```

Abbildung 36: XML-Auszug aus Patzig mit einem Absatz als Beispiel.

Da das `<p>`-Element im Unterschied zu den leeren Elementen wie `pb` oder `1b` ein öffnendes und schließendes Tag hat, wird beim Import der TEI-Xml-Datei durch den Importer schon ein `p`-Knoten erstellt, der mit einer `FIRST_CHILD_OF`-Kante mit dem ersten Wort des Absatzes und mit einer `LAST_CHILD_OF`-Kante mit dem letzten Wort des Absatzes verknüpft ist.

²⁵Die Darstellung der Wortkette ist zwischen den Wortknoten `der` und `einem` zu Gunsten der Übersichtlichkeit gekürzt.

²⁶Die Anzahl der `<p>`-Elemente im Graph erhält man mit der Abfrage MATCH (n:XmlTag {`_name:'p'`}) RETURN count(n);

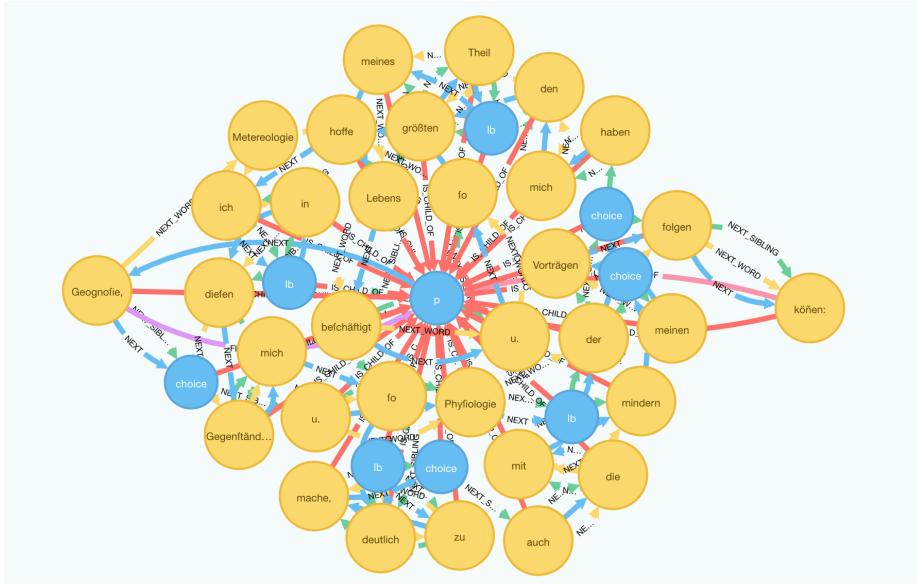


Abbildung 37: Ein Teil des gleichen Absatzes aus Patzig im Graphen.

Alle Wörter eines Absatzes sind darüber hinaus über `NEXT_SIBLING`-Kanten in der Textreihenfolge verknüpft.

21.1.5 Kapiteleinteilung

Im DTA-Basisformat wird bei der Transkription von Büchern die Kapiteleinteilung mit verschachtelten `div`-Element vorgenommen. Das im `div`-Element erlaubte `@n`-Attribut gibt die Strukturebene an. Über das `@type`-Attribut kann der Typ des Kapitels näher spezifiziert werden. Eine Liste der möglichen Werte für das Attribut findet sich unter <http://deutschestextarchiv.de/doku/basisformat/div.html>.

Für Manuskripte, wie der hier behandelten Vorlesungsmitsschrift von Patzig gibt es unter <http://deutschestextarchiv.de/doku/basisformat/msKapitel.html> noch zwei zusätzliche Werte für das `@type`-Attribut, nämlich *session* für Vorlesungsmitsschriften und *letter* für Briefe.

Mit folgendem cypher-Query erhalten wir die in Patzig verwendeten Werte für das `@type`-Attribut des `div`-Elements.

```
MATCH (n:XmlTag {_name:'div'})
RETURN n.type, count(n.type) AS Anzahl ORDER BY Anzahl DESC;
```

n.type	Anzahl
session	62
null	0

Es sind also insgesamt 62 Kapitel vom Typ *session* (Vorlesungsmitschrift) enthalten. Mit folgendem cypher-Query wird die Kapitelstruktur der ersten Kapitel und der darunter liegenden Ebenen bis zum jeweils ersten und letzten Wort des Kapitels angezeigt.

```
MATCH
p1 = shortestPath(
    (div:XmlTag {_name:'div'})-<[:FIRST_CHILD_OF*..20]-(w1:XmlWord)) ,
p2 = shortestPath(
    (div:XmlTag {_name:'div'})-<[:LAST_CHILD_OF*..20]-(w2:XmlWord))
RETURN p1,p2 LIMIT 20;
```

Mit dem folgenden cypher-Query wird das erste Wort des Kapitels über eine FIRST_CHILD_OF-Kante und das letzte Wort des Absatzes über eine LAST_CHILD_OF-Kante mit dem div-Knoten verbunden. Um die neu erstellen Kanten von den vom Importer erstellen zu unterscheiden erhalten diese die Proptery *type* mit dem Wert *graph*. Um die div-Knoten von den anderen XmlTag-Knoten unterscheiden zu können erhalten sie das zusätzliche Label *Session*.

```
MATCH
p1 = shortestPath(
    (div:XmlTag {_name:'div'})-<[:FIRST_CHILD_OF*..20]-(w1:XmlWord)
),
p2 = shortestPath(
    (div:XmlTag {_name:'div'})-<[:LAST_CHILD_OF*..20]-(w2:XmlWord)
)
MERGE (w1)-[:FIRST_CHILD_OF {type:'graph'}]->
    (div)-[:LAST_CHILD_OF {type:'graph'}]-(w2)
SET div:Session
RETURN * LIMIT 20;
```

21.1.6 Zusammenfassung

In diesem Kapitel wurden exemplarisch die XML-Strukturen für Zeilen (1b), Seiten (pb), Absätze (p) und Kapitel (div) in Graphstrukturen überführt, in denen jedes Element nur noch aus einem Knoten besteht. Mit diesem Knoten wird jeweils das erste und das letzte betroffene Wort mit einer FIRST_CHILD_OF- und einer LAST_CHILD_OF-Kante verknüpft. Damit entstehen offensichtlich übereinanderliegende Strukturen, was im Graphen aber kein Problem darstellt.

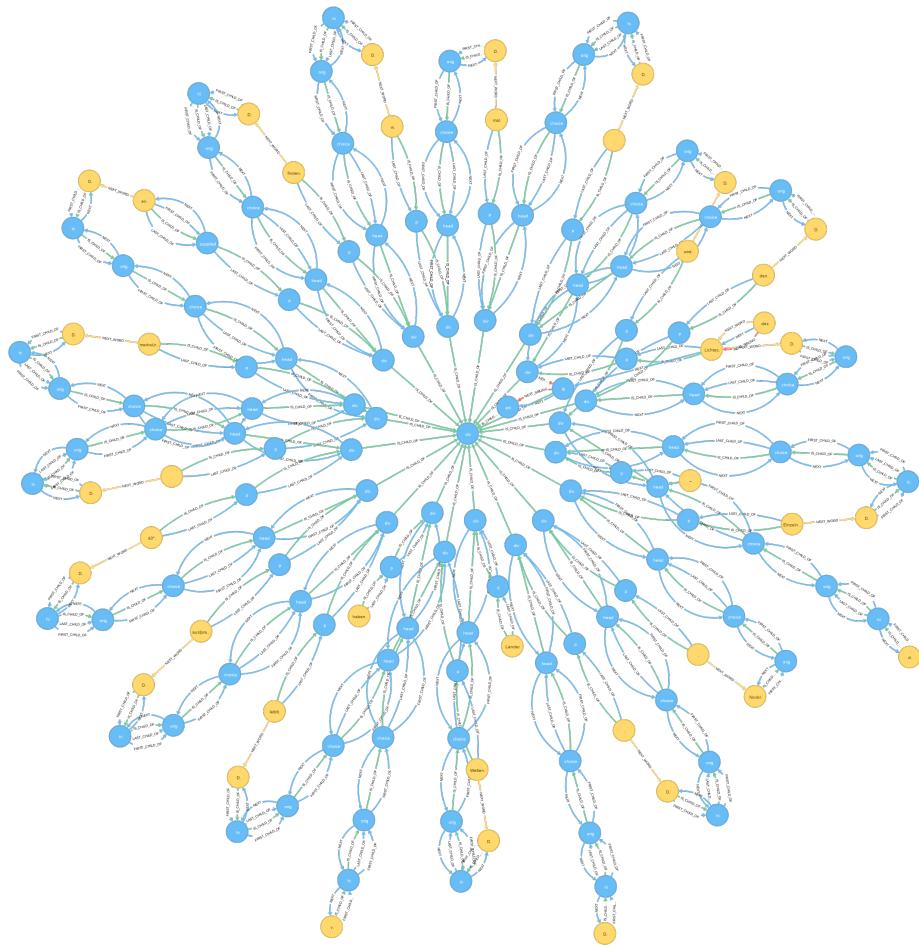


Abbildung 38: Struktur der ersten Kapitel mit dem jeweils ersten und letzten Wort.

21.2 Editorische Eingriffe

21.2.1 Hinzufügungen und Tilgungen

Die Elemente <add> und werden für Kennzeichnung von Tilgungen und Hinzufügungen des Autors oder von späteren Bearbeitern verwendet.

21.2.1.1 <add>-Element

Dabei können die Umstände der Änderungen beim <add>-Element mit dem @place-Attribut näher beschrieben, welches die in der folgenden Tabelle angegebenen Werte annehmen darf²⁷:

Element	@place-Wert	Bedeutung
<add>	superlinear	über der Zeile eingetragen
<add>	sublinear	unter der Zeile eingetragen
<add>	intralinear	innerhalb der Zeile eingetragen
<add>	across	über den ursprünglichen Text geschrieben
<add>	left	am linken Rand eingetragen
<add>	right	am rechten Rand eingetragen

Mit folgenden cypher-Query erhält man die Verteilung der Attributwerte.

```
MATCH (n:XmlTag {_name:'add'})  
RETURN n.place, count(n.place) AS Anzahl ORDER BY Anzahl DESC;
```

n.place	Anzahl
across	436
superlinear	268
intralinear	60
left	16
sublinear	2

21.2.1.2 -Element

Die mit dem -Element gekennzeichneten Tilgungen können mit dem @rendition-Attribut näher beschrieben werden, dessen mögliche Werte in der folgenden Tabelle angegeben sind²⁸.

²⁷Vgl. hierzu <http://deutschestextarchiv.de/doku/basisformat/msAddDel.html>.

²⁸Vgl. hierzu <http://deutschestextarchiv.de/doku/basisformat/msAddDel.html>.

Element	@rendition-Wert	Bedeutung
	#ow	Tilgung durch Überschreibung des ursprünglichen Textes
	#s	Tilgung durch Streichung
	#erased	Tilgung durch Radieren, Auskratzen o. ä.

Mit folgenden cypher-Query erhält man die Verteilung der Attributwerte.

```
MATCH (n:XmlTag { _name:'add' })
RETURN n.rendition, count(n.rendition) AS Anzahl
ORDER BY Anzahl DESC;
```

n.rendition	Anzahl
#ow	436
#s	268
#erased	60

21.2.1.3 Umbau von <add>- und -Elementen in einer <subst>-Umgebung

Der Umbau wird an einem Beispieltext der Seite 32 des Patzig-Manuskripts durchgeführt[^148e].

```
rendition="#aq">po&#x017F;thumi&#x017F;chen</hi>
Werke auf-<lb/>
gedeckt u. <subst><del rendition="#s">&#x017F;eine
Fehler</del><add
place="superlinear">die&#x017F;e</add></subst> zum
Theil erka<supplied reason="damage"
resp="#BF">n&#x0303;t,</supplied><lb/>
wen&#x0303; er redete von häßlichen u.
```

Abbildung 39: -Beispiel in der XML-Ansicht.

Im Graphen findet man die entsprechende Stelle mit folgendem cypher-Query.

```
MATCH
(w1:XmlWord)-[r1:NEXT_WORD]->
(w2:XmlWord)-[r2:FIRST_CHILD_OF]->
(t1)-[r3:FIRST_CHILD_OF]->
(s:XmlTag { _name:'subst', DtaID:8248})
<- [r4:LAST_CHILD_OF]-(t2)
<- [r5:LAST_CHILD_OF]-(w4:XmlWord)
-[r6:NEXT_WORD]->(w5:XmlWord),
(w2)-[r7:NEXT_WORD]->(w3)-[r8:NEXT_WORD]->(w4)
```

```
RETURN *;
```

Der Query gruppiert sich um den **s**-Knoten, der das **subst**-Element darstellt und es über die DtaID identifiziert. Vom **s**-Knoten ausgehend, folgt der Pfad einerseits über **FIRST_CHILD_OF**-Kanten zum **n3**-Knoten (add-Element) und zum **n2**-Knoten, der schließlich das Wort *seine* darstellt. Über die **LAST_CHILD_OF**-Kante geht es zum **n4**-Knoten (del-Element) zum **n5**-Wortknoten, der das Wort *diese* darstellt. Im zweiten Teil des MATCH-Befehls wird der Pfad zwischen dem Wort *seine* und *diese* ermittelt und schließlich alles ausgegeben.

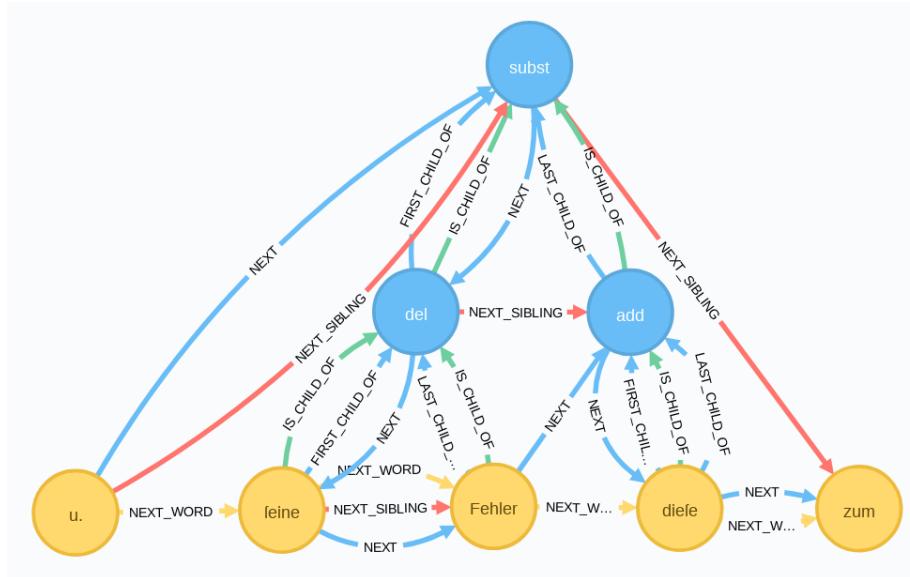


Abbildung 40: -Beispiel in der Graph-Ansicht.

cyper-Query für den umgebaut

```

MATCH
(w1:XmlWord)-[r1:NEXT_WORD]-
(w2:XmlWord)-[r2:FIRST_CHILD_OF]-
(t1)-[r3:FIRST_CHILD_OF]-
(s:XmlTag { _name: 'subst' , DtaID:8248})
-<[r4:LAST_CHILD_OF]-(t2)
-<[r5:LAST_CHILD_OF]-(w4:XmlWord)
-[r6:NEXT_WORD]->(w5:XmlWord),
(w2)-[r7:NEXT_WORD]->(w3)-[r8:NEXT_WORD]->(w4)
DELETE r1, r8
SET r8.variant_type='add'
CREATE (w1)-[:NEXT_WORD{variant_type:'add'}]->(w4)
CREATE (w1)-[:NEXT_WORD{variant_type:'del'}]->(w2)
CREATE (w3)-[:NEXT_WORD{variant_type:'del'}]->(w5)
```

```

SET r7.variant_type='del'
RETURN *;

```

Das Ergebnis erhält man über den folgenden Query.

```

MATCH
(w1:XmlWord)-[r1:NEXT_WORD]->
(w2:XmlWord)-[r2:FIRST_CHILD_OF]->
(t1)-[r3:FIRST_CHILD_OF]->
(s:XmlAttribute {_name:'subst', DtaID:8248})
<-[r4:LAST_CHILD_OF]-(t2)
<-[r5:LAST_CHILD_OF]-(w4:XmlWord)
-[r6:NEXT_WORD]->(w5:XmlWord),
(w2)-[r7:NEXT_WORD]->(w3)-[r8:NEXT_WORD]->(w4)

```

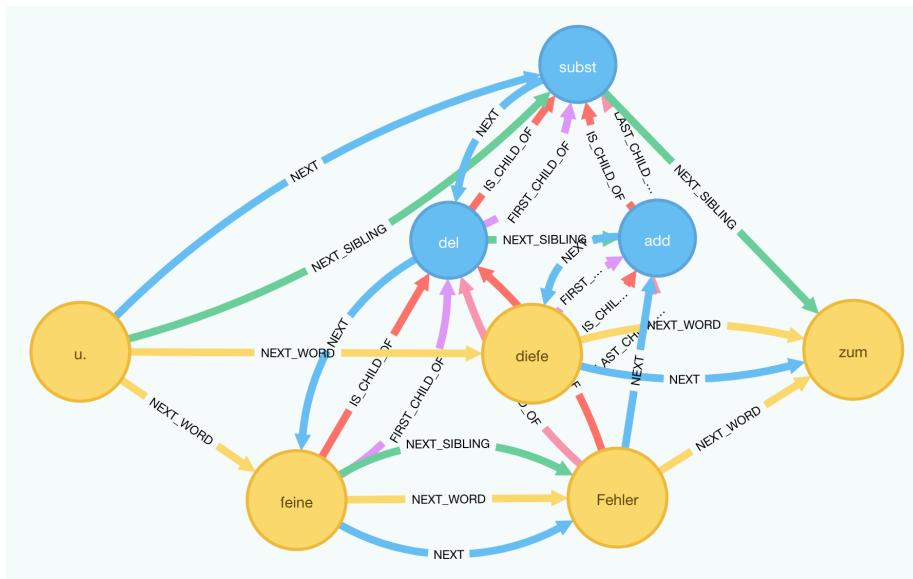


Abbildung 41: <subst>-Beispiel nach dem Graph-Umbau.

21.2.2 <choice>-Element

21.2.3 <sic> und <corr>-Elemente

22 Anhang

22.1 cypher-Befehle für den Import der Mitschrift von Patzig

Mit den folgenden Befehlen wird die Humboldt-Mitschrift von Patzig in die Graphdatenbank importiert, jedem Knoten zur Identifikation die DTA-URL als Property mitgegeben und die Knoten durchnummeriert. Die Nummerierung ist für das wiederholte Auffinden der in diesem Beitrag behandelten Textstellen notwendig.

```
// Alles löschen
MATCH(n) DETACH DELETE n;

// Patzig importieren
call
apoc.xml.import('http://www.deutsches-textarchiv.de/book/download_xml/patzig_msgermfol841842')
yield node return node;

// URL von Dokument auf alle Wort-Knoten kopieren:
match (d:XmlDocument)-[:NEXT_WORD*]->(w:XmlWord)
set w.url = d.url;

// Knoten durchzählen
MATCH p = (start:XmlDocument)-[:NEXT*]->(end:XmlTag)
WHERE NOT (end)-[:NEXT]->()
AND start.url = 'http://www.deutsches-textarchiv.de/book/download'
WITH nodes(p) as nodes, range(0, size(nodes(p))) AS indexes
UNWIND indexes AS index
SET (nodes[index]).DtaID = index;
```

22.2 Liste aller im Patzig-Manusskript vorkommenden Elemente sortiert nach Häufigkeit

cypher-Query zur Erstellung der Tabelle:

```
MATCH (n:XmlTag)
RETURN n._name,
count(n._name) AS Anzahl
ORDER BY Anzahl DESC;
```

Element	Anzahl
lb	16075
hi	3768
choice	2184
expan	1856
abbr	1856
supplied	1517
persName	925
note	914
add	782
del	644
unclear	526
subst	516
pb	422
gap	419
fw	357
p	238
metamark	234
reg	221
orig	221
corr	107
sic	107
div	64
head	63
item	51
rendition	22
space	20
ref	19
bibl	15
list	13
idno	12
surname	12
forename	12
figure	10
milestone	9
orgName	7
resp	6
respStmt	6
editor	6
cell	6
measure	4
titlePart	4
classCode	4
handNote	4
date	2

Element	Anzahl
edition	2
titleStmt	2
editionStmt	2
publisher	2
publicationStmt	2
title	2
author	2
pubPlace	2
row	2
biblFull	1
fileDesc	1
editorialDecl	1
text	1
physDesc	1
back	1
sourceDesc	1
address	1
country	1
textClass	1
titlePage	1
repository	1
encodingDesc	1
front	1
availability	1
msDesc	1
addrLine	1
byline	1
body	1
typeDesc	1
teiHeader	1
docTitle	1
licence	1
msIdentifier	1
profileDesc	1
tagsDecl	1
email	1
TEI	1
table	1
langUsage	1
docAuthor	1
extent	1
language	1
handDesc	1

22.3 Weitere Texte

22.3.1 Dokument vorbereiten

22.3.1.1 <lb/>-GraphElemente erstellen

Finde alle lb-Elemente, die direkt an einem Wortknoten stehen und ein Wort trennen.

```
MATCH (w0:XmlWord)-[:NEXT]->(n:XmlTag {_name:'lb'})-[:NEXT]->(w1:XmlWord)
WHERE w0.text =~ '.*-'
RETURN *;

MATCH (w0:XmlWord), (n:XmlTag {_name:'lb'}),  

p1=shortestPath((w0)-[:NEXT*..1]->(n))
//p1=shortestPath((w0)-[:NEXT*..2]->(n)-[:NEXT*..2]->(w1))
WHERE w0.text =~ '.*-'
RETURN p1;
```

<lb/>-Elemente per Hand entfernen, die Wörter trennen:

```
MATCH
(n2:XmlWord)-[:NEXT_WORD]->
(n3:XmlWord)-[:NEXT_WORD]->
(n4:XmlWord),
(n3)-[:NEXT]->(t1:XmlTag{_name:'lb'})-[:NEXT]->(n4)
WHERE n3.text =~ '.*-'
SET t1.before = n3.text,
t1.after = n4.text,
n4.text = left(n3.text,
size(n3.text)-1)+n4.text
CREATE (n2)-[:NEXT_WORD]->(n4)
CREATE (n2)-[:NEXT]->(t1)
CREATE (n2)-[:NEXT_SIBLING]->(t1)
DETACH DELETE n3
RETURN *;

MATCH
(n2:XmlWord)-[:NEXT_WORD]->
(n3:XmlWord)-[:NEXT_WORD]->
(n4:XmlWord),
(n3)-[:NEXT]->(t1:XmlTag{_name:'lb'})-[:NEXT]->(n4)
CREATE (n2)-[:NEXT_WORD]->(n4)
CREATE (n2)-[:NEXT]->(t1)
CREATE (n2)-[:NEXT_SIBLING]->(t1)
DETACH DELETE n3
RETURN *;

// lb-GraphElemente erstellen
```

```

MATCH p=(t1:XmlTag {_name:'lb'})-[:NEXT]->(n1:XmlWord)-[:NEXT_WORD*..20]->(n2:XmlWord)-[:NEXT]
CREATE (gt:GraphElement {_name:'lb'})
MERGE (gt)-[:FIRST_CHILD_OF]->(n1)
MERGE (gt)-[:LAST_CHILD_OF]->(n2)
RETURN *;

```

22.3.1.2 <lb/>-Elemente umwandeln

22.3.1.2.1 <lb/>-Elemente ohne Worttrennungen umwandeln

22.3.1.3 <fw>-Elemente aus der Textkette rausnehmen

Im Patzig-Manuskript wird am Ende jeder Seite das erste Wort der folgenden Seite vermerkt um neben der Seitennummerierung auch einen inhaltlichen Anhaltspunkt für die Reihenfolge der Seiten zu geben. Am Ende der Seite 6 befindet sich das Wort **Nachdem**. Mit Graph-Refactoring wird nun dieses Wort aus der Textkette herausgenommen, verbleibt aber als Information im Graphen. Zeit

22.4 [^148e]: Vgl. http://www.deutsches-textarchiv.de/book/view/patzig_msgerm

title: Tipps und Tricks layout: default order: 85 contents: true —

23 Inhalt

{:.no_toc}

- Will be replaced with the ToC, excluding the “Contents” header {:toc}

24 Mehrere Werte in einem CSV-Feld importieren

Beim Import von Daten im CSV-Format in die Graphdatenbank kann es vorkommen, dass in einem CSV-Feld mehrere Werte zusammen stehen. In diesem Abschnitt wird erklärt, wie man diese Werte auseinandernehmen, einzeln als Knoten anlegen und verknüpfen kann.

In der Regel ist es von Vorteil, zunächst das CSV-Feld als eine Property zu importieren und in einem zweiten Schritt auseinanderzunehmen.

Angenommen wir haben Personen importiert, die in der Property `abschluss` eine kommaseparierte Liste von verschiedenen beruflichen Abschlüssen haben, wie z.B. Lehre, BA-Abschluss, MA-Abschluss, Promotion.

In der Property `abschluss` steht zum Beispiel drin:

```
lic. theol., mag. art., dr. theol., bacc. art., bacc. bibl.  
theol.
```

Der Befehl hierzu sieht wie folgt auch:

```
MATCH (p:Person)  
FOREACH ( j in split(p.abschluss, ", ") |  
MERGE (t:Titel {name:j})  
MERGE (t)-[:ABSCHLUSS]-(p)  
);
```

Der Query nimmt die Liste von Abschlüssen jeweils beim Komma auseinander, erstellt mit dem `MERGE`-Befehl einen Knoten für den Abschluss (falls noch nicht vorhanden) und verlinkt diesen Knoten dann mit dem Personenknoten. Zu beachten ist, dass die im CSV-Feld gemeinsam genannten Begriffe konsistent benannt sein müssen.

25 MERGE schlägt fehl da eine Property NULL ist

Der `MERGE`-Befehl entspricht in der Syntax dem `CREATE`-Befehl, überprüft aber bei jedem Aufruf, ob der zu erstellende Knoten bereits in der Datenbank existiert. Bei dieser Überprüfung werden alle Propertys des Knoten überprüft. Falls also eine vorhandener Knoten eine Property nicht enthält, wird ein weiterer Knoten erstellt. Umgekehrt endet der `MERGE`-Befehl mit einer Fehlermeldung, wenn eine der Propertys, die er erstellen soll `NULL` ist.

Gerade beim Import von CSV-Daten leistet der `MERGE`-Befehl in der Regel sehr gute Dienste, da man mit ihm bereits beim Import einer Tabelle weitere Knotentypen anlegen und verlinken kann. Oft kommt es aber vor, dass man sich nicht sicher ist, ob eine entsprechende Property in allen Fällen existiert. Hier bietet es sich an, vor dem `MERGE`-Befehl mit einer `WHERE`-Clause die Existenz der Property zu überprüfen.

Im folgenden Beispiel importierten wir Personen aus einer CSV-Liste, bei denen pro Person jeweils eine ID, ein Name und manchmal ein Herkunftsland angegeben ist. Im ersten Schritt werden im `CREATE`-Statement die Personen erstellt und auch der Herkunftsland als Property angelegt, der aber auch `NULL` sein kann.

```
LOAD CSV WITH HEADERS FROM "file:///import.csv" AS line  
CREATE (p:Person {pid:line.ID_Person, name:line.Name, herkunft:line.Herkunft});
```

Im zweiten Schritt wird nun der `LOAD CSV`-Befehl nochmals ausgeführt und über die `WHERE`-Clause nur jene Fälle weiter bearbeitet, in denen die Property Herkunft nicht `NULL` ist. Nach der `WHERE`-Clause wird über den `MATCH`-Befehl zunächst der passende Personenknoten aufgerufen, anschließend per `MERGE`-Befehl der Ortsknoten erstellt (falls noch nicht vorhanden) und schließlich mit `MERGE` beide verknüpft.

```
LOAD CSV WITH HEADERS FROM "file:///import.csv" AS line
WHERE line.Herkunft IS NOT NULL
MATCH (p:Person {pid:line.ID_Person})
MERGE (o:Ort {ortsname:line.Herkunft})
MERGE (p)-[:HERKUNFT]->(o);
```

26 Knoten hat bestimmte Kante nicht

Am Beispiel der Regesta-Imperii-Graphdatenbank der Regesten Kaiser Friedrichs III. werden mit dem folgenden Cypher-Query alle Regestenknoten ausgegeben, die keine `PLACE_OF_ISSUE`-Kante zu einem `Place`-Knoten haben:

```
MATCH (reg:Regesta)
WHERE NOT
(reg)-[:PLACE_OF_ISSUE]->(:Place)
RETURN reg;
```

27 Häufigkeit von Wortketten

Am Beispiel des DTA-Imports von Berg Ostasien wird mit dem folgenden Query die Häufigkeit von Wortketten im Text ausgegeben:

```
MATCH p=(n1:Token)-[:NEXT_TOKEN]->(n2:Token)-[:NEXT_TOKEN]->(n3:Token)
WITH n1.text as text1, n2.text as text2, n3.text as text3, count(*) as count
WHERE count > 1 // evtl höherer Wert hier
RETURN text1, text2, text3, count ORDER BY count DESC LIMIT 10
```

28 Der `WITH`-Befehl

Da cypher eine deklarative und keine imperative Sprache ist gibt es bei der Formulierung der Querys Einschränkungen.^[^03a5] Hier hilft oft der `WITH`-Befehl weiter, mit dem sich die o.a. beiden Befehle auch in einem Query vereinen lassen:

```
LOAD CSV WITH HEADERS FROM "file:///import.csv" AS line
CREATE (p:Person {pid:line.ID_Person, name:line.Name, herkunft:line.Herkunft})
WITH line, p
```

```

WHERE line.Herkunft IS NOT NULL
MERGE (o:Ort {ortsname:line.Herkunft})
MERGE (p)-[:HERKUNFT]->(o);

```

Der `LOAD CSV`-Befehl lädt die CSV-Datei und gibt sie zeilenweise an den `CREATE`-Befehl weiter. Dieser erstellt den Personenknoten. Der folgende `WITH`-Befehl stellt quasi alles wieder auf Anfang und gibt an die nach ihm kommenden Befehle nur die Variablen `line` und `p` weiter.

29 Die Apoc-Bibliothek

Die Funktionalitäten sind bei neo4j in verschiedene Bereiche aufgeteilt. Die Datenbank selbst bringt Grundfunktionalitäten mit. Um Industriestandards zu genügen haben diese Funktionen umfangreiche Tests und Prüfungen durchlaufen. Weiteregehende Funktionen sind in die sogenannte *apoc-Bibliothek* ausgelagert, die zusätzlich installiert werden muss. Diese sogenannten *user defined procedures* sind benutzerdefinierte Implementierungen bestimmter Funktionen, die in cypher selbst nicht so leicht ausgedrückt werden können. Diese Prozeduren sind in Java implementiert und können einfach in Ihre Neo4j-Instanz implementiert und dann direkt von Cypher aus aufgerufen werden.²⁹

Die APOC-Bibliothek besteht aus vielen Prozeduren, die bei verschiedenen Aufgaben in Bereichen wie Datenintegration, Graphenalgorithmen oder Datenkonvertierung helfen.

29.1 Installation in neo4j

Die Apoc-Bibliothek lässt sich unter <http://github.com/neo4j-contrib/neo4j-apoc-procedures/releases/> herunterladen und muss in den plugin-Ordner der neo4j-Datenbank kopiert werden.

29.2 Installation unter neo4j-Desktop

In *neo4j-Desktop* kann die Apoc-Bibliothek jeweils pro Datenbank im Management-Bereich über den Reiter `plugins` per Mausklick installiert werden.

29.3 Liste aller Funktionen

Nach dem Neustart der Datenbank stehen die zusätzlichen Funktionen zur Verfügung. Mit folgendem Befehl kann überprüft werden, ob die Apoc-Bibliotheken installiert sind:

²⁹Vgl. <https://guides.neo4j.com/apoc> (zuletzt aufgerufen am 11.04.2018).

```
CALL dbms.functions()
```

Wenn eine Liste mit Funktionen ausgegeben wird, war die Installation erfolgreich. Falls nicht, sollte die Datenbank nochmals neu gestartet werden.

29.4 Dokumentation aller Funktionen

In der Dokumentation der apoc-Bibliothek sind die einzelnen Funktionen genauer beschrieben.

29.5 [^03a5]: Hierzu vgl. https://de.wikipedia.org/wiki/Deklarative_Programmierung zuletzt abgerufen am 12.6.2018.

title: Inhalt layout: default permalink: /contents/ nav: true order: 2 —

30 Inhalt

```
{% assign content_pages = site.pages | sort:"order" %} {% for my_page in content_pages %} {% if my_page.contents %} {{my_page.order}}. {{my_page.title }} {% endif %} {% endfor %} — layout: page title: Impressum permalink: /credits/ nav: true order: 5 —
```

Verantwortlich:

Dr. Andreas Kuczera Regesta Imperii Justus-Liebig Universität Gießen Historisches Institut, Mittelalter Otto-Behaghel-Str. 10 D-35394 Gießen Haus C Raum 244a

mailto:andreas.kuczera@adwmainz.de

<https://orcid.org/0000-0003-1020-507X>

www.graphentechnologien.de

Letzte Aktualisierung: Juni 2018

30.1 Graphentechnologien in den Digitalen Geisteswissenschaften von Andreas Kuczera ist lizenziert unter einer Creative Commons Attribution-ShareAlike 4.0 International Lizenz.

layout: default title: Info nav: true order: 1 —

Dieses Studienbuch führt in Graphentechnologien mit einem geisteswissenschaftlichen Fokus ein und findet sich unter <http://www.graphentechnologien.de>. Es

handelt sich momentan um eine Arbeitsfassung, die noch weiter entwickelt, ggf. umstrukturiert und ausgebaut wird. Für Rückmeldungen, Verbesserungen und Hinweise bin ich dankbar.

Die Regesten Heinrichs IV. sind unter anderem Gegenstand einer Lehreinheit zu Graphentechnologien in den digitalen Geisteswissenschaften im Rahmen des mained-Studiengangs zur Digitalität in den Geistes- und Kulturwissenschaften. Die Slides der Lehreinheit stehen unter CC-BY-Lizenz und sind erreichbar unter:

<https://digitale-methodik.adwmainz.net/mod5/5c/slides/graphentechnologien/#/step-1>

Weitere Informationen zu Graphentechnologien in den Digitalen Geistes- und Sozialwissenschaften finden Sie auf den Seiten der AG-Graphen des Dhd-Verbandes: <https://graphentechnologien.hypotheses.org/>. # Graphentechnologien Digitale Geisteswissenschaften rund um Graphentechnologien

<https://kuczera.github.io/Graphentechnologien/>