



Mira Höggerl, BSc

Graphdatenbanken zur Netzberechnung von EVU

MASTERARBEIT

Zur Erlangung des akademischen Grades
Diplom-Ingenieurin

eingereicht an der
Technischen Universität Graz

Betreuung: Assoc.Prof. Dipl.-Ing. (FH) Dr.techn. Johannes Scholz

Institut für Geodäsie - AG Geoinformation
Graz, März 2023

KURZFASSUNG

Diese Masterarbeit wurde in Zusammenarbeit mit der Energienetze Steiermark GmbH durchgeführt.

Das Ziel der vorliegenden Arbeit ist es, zu beantworten, welche Möglichkeiten Graphdatenbanken zur Netzberechnung von Energieversorgungsunternehmen bieten und welche Schritte für eine Überführung der Daten aus einer relationalen Datenbank in eine Graphdatenbank notwendig sind.

Zur Beantwortung der Forschungsfrage wurde ein qualitativer Vergleich zu den Datenbanksystemen erstens relationale Datenbanken und zweitens Graphdatenbanken durchgeführt, welcher dann bezogen auf den Anwendungsfall von Energieversorgungsunternehmen beurteilt wurde. Es wurde ein Datenmodell für die Überführung der Daten, welche in der Größe ausgewählter Ortsnetze vorliegen, aus einer relationalen Datenbank in die Graphdatenbank Neo4j erstellt. Voraussetzung dafür war eine saubere Topologie im relationalen Datenbanksystem.

Der qualitative Vergleich zeigte, dass beide Systeme Vor- und Nachteile haben, welche stark vom Anwendungsfall abhängen. Jedenfalls eignen sich Graphdatenbanken für Abfragen zu Netzberechnungen, was die Einfachheit der Abfragen im Implementierungsteil der Arbeit belegt.

ABSTRACT

This master thesis is done in cooperation with Energienetze Steiermark GmbH.

The goal of this thesis is to answer the question, which possibilities graph databases offer for the network calculation of electric utilities and what is necessary for a transfer of the data from a relational database into a graph database.

In order to answer the research question, a qualitative comparison of the database systems relational databases on the one side and graph databases on the other side was conducted, which was then evaluated in relation to the use case of electric utilities. Furthermore, a data model for the transfer of data, which is given in the size of selected local grids, from a relational

database to the graph database Neo4j was created. The prerequisite for this was a clean topology in the relational database system.

The qualitative comparison showed that both systems have advantages and disadvantages, which strongly depend on the use case. However, graph databases are suitable for queries for network calculations, which is proved by the simplicity of the queries in the implementation part of this thesis.

DANKSAGUNG

Mein Dank gilt all jenen Personen, die mir beim Entwickeln und Erstellen dieser Masterarbeit geholfen haben. Insbesondere gilt dies meinem Betreuer, Assoc.Prof. Dipl.-Ing. (FH) Dr.techn. Johannes Scholz, der mir jederzeit verlässlich zur Verfügung stand und immer hilfreiche Anregungen parat hatte. Danke für die gute Zusammenarbeit, das wertvolle Feedback und die umfassenden Fachkenntnisse.

Ich danke dem Abteilungsleiter der Geoinformationsabteilung der Energienetze Steiermark GmbH, Dipl.-Ing. Werner Samhaber, für die angenehme Zusammenarbeit, den wertvollen Rat und das Vertrauen. Danke für die ausführlichen Informationen und die zur Verfügung gestellten Daten. Des Weiteren gilt mein Dank meinen Arbeitskollegen bei der Energienetze Steiermark GmbH, die mir während meiner Forschung zur Seite gestanden sind und mir ihr großes Wissen und ihre Erfahrung weitervermittelt haben.

Ein besonderer Dank gilt auch meiner Kommilitonin Julia für die vielen Gespräche rund um die Masterarbeit und darüber hinaus für den starken emotionalen Rückhalt und die schöne gemeinsame Studienzeit. Danke auch an Matthias für die Ehrlichkeit, Rücksicht und Stütze in dieser Zeit.

Abschließend möchte ich mich bei meinen Eltern bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben und mir immer Sicherheit in herausfordernden Zeiten gegeben haben.

INHALTSVERZEICHNIS

Abkürzungsverzeichnis	x
1 Einleitung	1
1.1 Motivation	1
1.2 Aufgaben- und Problemstellung	1
1.3 Ziele der Arbeit und Lösungsansatz	2
1.4 Literaturüberblick	3
1.5 Gliederung der Arbeit	4
2 Theoretische Grundlagen	5
2.1 Relationale Datenbanken	5
2.1.1 Struktur relationaler Datenbanken	5
2.1.2 Relationale Algebra	7
2.1.3 Abfragesprache SQL	8
2.1.4 Transaktionen	10
2.1.5 Normalisierung	11
2.1.6 Grenzen relationaler Datenbanken	13
2.2 Graphdatenbanken	14
2.2.1 NoSQL	14
2.2.2 Graphentheorie	16
2.2.3 Traversieren von Graphen	17
2.2.4 Graphdatenbanken	17
2.2.5 Typische Graphdatenbankanwendungen	18
2.2.6 Datenmodellierung mit Graphdatenbanken	18
2.2.7 Neo4j	20
2.2.8 Abfragesprache Cypher	20
2.3 Netzberechnungen	23
2.3.1 Vektordatenmodell	23
2.3.2 OGC Simple Features	23
2.3.3 Topologische Features	24
2.3.4 Netzwerkdatenmodell	25
2.3.5 Topologien des Niederspannungsstromnetzes	25
3 Umsetzung	28
3.1 Methodik	28
3.2 Ausgangslage	29
3.3 Anforderungen EVU	29

4	Qualitative Evaluierung	31
4.1	Sicherheit	31
4.2	Einfachheit der Programmierung und Abfragesprache	32
4.3	Flexibilität	33
4.4	Performance von Netzberechnungen	34
4.5	Skalierbarkeit	34
4.6	Ausgereiftheit und Support	35
5	Implementierung	37
5.1	Datenmodelldefinition	37
5.1.1	Netzkomponenten	37
5.1.2	Datenmodell	39
5.2	Entwicklungsumgebung	42
5.3	Überführung der Daten von einer relationalen Datenbank in eine Graphdatenbank	42
5.3.1	MS SQL Server	43
5.3.2	FME Job	43
5.3.3	SQLite Datenbank	44
5.3.4	CSV Tabellen	45
5.3.5	Neo4j	46
5.4	Netzberechnungen	47
5.4.1	Abzweigeinfärbung	47
5.4.2	Ausfall einer Leitung	48
5.4.3	Netzverfolgung	50
5.4.4	Mehrere Ortsnetze	52
6	Ergebnisse und Fazit	55
6.1	Überblick der Qualitativen Evaluierung	55
6.2	Diskussion	57
6.3	Zusammenfassung der wissenschaftlichen Arbeitsergebnisse	59
6.4	Ausblick	59
	Literatur	61
	Anhang	64
.1	Transkript Experteninterview	64

ABBILDUNGSVERZEICHNIS

2.1	NoSQL Stores (Quelle: [7, S. 197])	14
2.2	Königsberger Brückenproblem	16
2.3	Labeld Property Graph Model	19
2.4	Punkt, Linie und Polygon	23
2.5	Topologieformen von Ortsnetzen (Quelle: [24, S. 61])	26
4.1	Vertikale vs. Horizontale Skalierung	35
5.1	Datenmodell für die Graphdatenbank Neo4j	39
5.2	Angewendetes Datenmodell	41
5.3	Umsetzungsablauf	43
5.4	ER-Diagramm der SQLite Tabellen	44
5.5	CSV Tabellen	46
5.6	Abzweigeinfärbung	48
5.7	Relation gelöscht	49
5.8	Nodes, die nicht mit der Station verbunden sind	50
5.9	Netzverfolgung Ausgangslage	51
5.10	Netzverfolgung Ergebnis	52
5.11	25 Ortsnetze	53
5.12	25 Ortsnetze gezoomt	53
5.13	1 Ortsnetz	54

TABELLENVERZEICHNIS

3.1	Anforderungen EVU	30
4.1	Sicherheit Services	32
5.1	Verwendete Hardware	42
5.2	Verwendete Software mit Version	42
6.1	Überblick der qualitativen Evaluierung	56

LISTINGS

2.1	Einfache SQL Abfrage	9
2.2	Einfache Cypher Abfrage	20
2.3	Erweiterte Cypher Abfrage	21
5.1	Node.csv extrahieren	45
5.2	Relation.csv extrahieren	45
5.3	Node.csv in Neo4j importieren	46
5.4	Create Index	47
5.5	Relation.csv in Neo4j importieren	47
5.6	Abzweigeinfärbung	47
5.7	Relation löschen	48
5.8	Nodes, die nicht mit der Station verbunden sind	49
5.9	Netzverfolgung-Abfrage zum Node mit Node_ID = 25	51
5.10	1 Ortsnetz auswählen	54

ABKÜRZUNGSVERZEICHNIS

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
EN	Energienetze Steiermark GmbH
ER	Entity Relationship
EVU	Energieversorgungsunternehmen
FME	Feature Manipulation Engine (Software)
GI	Geographic Information
GIS	Geoinformationssystem
HTTP	Hypertext Transfer Protocol
JVM	Java Virtual Machine
MS	Microsoft
NS	Niederspannung
NoSQL	Not only SQL
OGC	Open Geospatial Consortium
RDBMS	Relationales Datenbank Management System
SQL	Structured Query Language
SSL	Secure Sockets Layer

KAPITEL 1

EINLEITUNG

1.1 MOTIVATION

Die Energieversorgung in Österreich soll ab 2030 vollständig aus erneuerbarer Energien sichergestellt sein. Damit geht der Weg von einer überschaubaren Anzahl an großen Kraftwerken, welche hauptsächlich auf Kohle, Gas und Wasserkraft basieren, über auf neue Konzepte, welche von Photovoltaikanlagen am Einfamilienhaus bis hin zum Windpark reichen und in der Stromerzeugung vergleichsweise volatil sind. Zudem wird die private Nutzung von Strom zunehmen, da mit Strom betriebene Wärmepumpen sowie Ladestationen für E-Autos keine Einzelfälle mehr sind. Außerdem stellen auch Energiegemeinschaften, bei denen Strom an unterschiedlichen Örtlichkeiten des Stromnetzes erzeugt, genutzt und eingespeist wird, neue Herausforderungen an das Stromnetz. Denn das Grundprinzip, dass die Erzeugung und der Verbrauch von Strom immer im Gleichgewicht sein muss, ist zu jeder Zeit zu erfüllen. [1, S. 10–11].

Der genannte Trend bedingt eine große Anzahl von elektrischen Netzberechnungen, welche ein topologisch sauberes Netzwerk voraussetzen, um unterschiedliche Aufgaben wie Abzweigeinfärbung, Ausfall einer Leitung und Netzverfolgung effizient zu bewältigen. [3]

1.2 AUFGABEN- UND PROBLEMSTELLUNG

Die Arbeit wird in Zusammenarbeit mit der Energienetze Steiermark GmbH (EN) verfasst. Die vorhandenen Netzdaten der EN sind aktuell vorrangig auf Lagegenauigkeit optimiert. Es handelt sich um sogenannte Open Geospatial Consortium (OGC) Simple Features, welche in einem Relationalen Datenbank Management System (RDBMS) verwaltet werden. Der OGC Simple Features Standard [2] legt fest, wie geographische Daten dargestellt werden. Änderungen in den räumlichen Daten machen eine Neuberechnung der Topologie notwendig, welche mit der Software FME von Safe Software durchgeführt wird und relativ aufwendig ist.

In der Literatur geht der Trend für solche Anwendungsbeispiele in Richtung Graphdatenbanken, da RDBMS für die Menge an Daten, welche zum Beispiel Einspeiser, SmartMeter und Ladestationen mit sich bringen, nicht effizient genug sind [3]. Graphdatenbanken bieten also eine Reihe von Vorteilen für Netzberechnungen von Stromdaten. Daher sollen im Zuge dieser Arbeit die Möglichkeiten von Graphdatenbanken für Netzberechnungen von Energieversorgungsunternehmen (EVU) evaluiert werden. Außerdem soll aufgezeigt werden, wie Daten aus der relationalen Datenbank in eine Graphdatenbank überführt werden können und was die Anforderungen an die Daten dazu sind. Zusätzlich sollen Abfragen zu möglichen Netzberechnungen implementiert werden.

Die EN stellt Daten, welche in der Größe ausgewählter Ortsnetze vorliegen, zur Verfügung. Die Ortsnetze, gegeben in OGC Simple Features [2], sind in einer relationalen Datenbank gespeichert. Die Arbeit beschränkt sich auf die Niederspannungsebene des Stromnetzes, da Mittel- und Hochspannung in der Praxis gesondert gehandhabt werden.

1.3 ZIELE DER ARBEIT UND LÖSUNGSANSATZ

Den Grundstein der Arbeit bildet ein ausführlicher Literaturrechercheteil zu den Themen relationale Datenbanken, Graphdatenbanken und Netzberechnungen, wo insbesondere auf Topologie und Stromnetze eingegangen wird.

Mit dem gewonnenen Wissen wird darauf aufbauend eine qualitative Evaluierung zu den beiden Datenbanksystemen relationale Datenbanken und Graphdatenbanken durchgeführt. Das Ergebnis wird dann bezogen auf den Anwendungsfall von EVU beurteilt. Um die Anforderungen von EVU zu definieren, wird ein semistrukturiertes Interview durchgeführt.

Ein weiteres Ziel der Arbeit ist die Überführung der Daten aus der relationalen Datenbank in die Graphdatenbank Neo4j. Dazu wird zuerst ein Datenmodell definiert. Der Prozess wird zumindest für ein Testgebiet in der Größe eines Ortsnetzes veranschaulicht. Ein Ortsnetz ist dafür geeignet, weil es die wichtigsten Komponenten des Netzes beinhaltet und es ein geschlossenes System abbildet. Damit können die größten Hürden und Komplikationen entdeckt und analysiert werden. Es sollte aufzeigbar sein, welche Anforderungen und Maßnahmen für eine Überführung der vorhandenen Daten notwendig sind. Im Graphdatenbanksystem werden dann Abfragen zu Anwendungsbeispielen wie Abzweigeinfärbung, Ausfall einer Leitung und Netzverfolgung implementiert.

1.4 LITERATURÜBERBLICK

Die theoretischen Grundlagen beruhen hauptsächlich auf Fachbüchern. Die Theorie zu relationalen Datenbanken ist vor allem auf das Fachbuch *Relationale Datenbanken* von Studer [4] aufgebaut, da es basierend auf den mathematischen Grundlagen das Thema ausführlich erläutert. Aber auch das Werk *Springer Handbook of Geographic Information* von Kresse und Danko [5] ist für Erklärungen in vereinfachter Sprache hilfreich. Für die Grenzen von relationalen Datenbanken wird das Buch *Big Data: Grundlagen, Systeme und Nutzungspotenziale* von Fasel und Meier [6] herangezogen.

Für die Theorie der Graphdatenbanken ist *Graph databases* von Robinson, Efreim und Webber [7] das relevanteste Werk, da es spezialisiert auf Neo4j Graphdatenbanken von Grund auf einfach behandelt. Zu dem Thema NoSQL sind die Paper “Comparison between relational and NOSQL databases” von Sahatqija, Ajdari, Zenuni u. a. [8] und “Persisting big-data: The NoSQL landscape” von Corbellini, Mateos, Zunino u. a. [9] wertvoll. Die Graphentheorie aus mathematischer Sicht wird nach den Buchteilen “Graphentheorie” von Nickel, Rebennack, Stein u. a. [10] und “Graphentheorie” von Brandes [11] beschrieben. Das Traversieren von Graphen wird nach dem Buch *Entwurf und Analyse von Algorithmen* von Nebel und Wild [12] erklärt.

Die Grundlagen für Netzberechnungen werden aus dem Buch *Geographic information science and systems* von Longley, Goodchild, Maguire u. a. [13] genommen. Dabei ist wiederum das Buch *Springer Handbook of Geographic Information* von Kresse und Danko [5] für vereinfachte Erklärungen und Ergänzungen wichtig. Die darauf aufbauenden Topologien für das Niederspannungsnetz entstammen dem Buch *Elektroenergiesysteme* von Schwab und Börnick [14].

Für die qualitative Evaluierung gibt es vor allem viel Literatur über den Vergleich von relationalen Datenbanken und NoSQL Datenbanken. Für diese Arbeit sind die Papers “Comparison between relational and NOSQL databases” von Sahatqija, Ajdari, Zenuni u. a. [8] und “Relational Database vs NoSQL” von Patel und Eltaieb [15] relevant. Die Vertiefung auf Graphdatenbanken stammt von dem Paper “A comparison of a graph database and a relational database: A data provenance perspective” von Vicknair, Macias, Zhao u. a. [16] und der Masterarbeit “Comparison of Graph Databases and Relational Databases When Handling Large-Scale Social Data” von Chen [17]. Der Performancevergleich bei Netzberechnungen beruht auf den Papers “The shortest path algorithm performance comparison in graph and relational database on a transportation network” von Miler, Medak und Odobašić [18], “Performance Analysis of RDBMS and No SQL Databases”

von Sharma, Sharma und Bunde [19] und der Masterarbeit “Comparison of Graph Databases and Relational Databases When Handling Large-Scale Social Data” von Chen [17]. Dabei ist zu erwähnen, dass auffällig viele Papers in diesem Bereich auf “A comparison of a graph database and a relational database: A data provenance perspective” von Vicknair, Macias, Zhao u. a. [16] referenzieren.

1.5 GLIEDERUNG DER ARBEIT

Nach dem Einleitekapitel 1 werden im Kapitel 2 die theoretischen Grundlagen zu den Datenbanken relationale Datenbanken einerseits und Graphdatenbanken andererseits gelegt. Außerdem werden mit dem Vektordatenmodell die Grundsteine für Netzwerkberechnungen vorgestellt. Abschließend werden in diesem Kapitel noch die unterschiedlichen Topologien des Niederspannungsstromnetzes veranschaulicht. Im Kapitel 3 wird die verwendete Methodik näher erläutert. Dafür werden die Ausgangslage sowie die Anforderungen von EVU beschrieben. Die qualitative Evaluierung befindet sich im Kapitel 4. Die Beschreibung zur Überführung der gegebenen Daten von einer relationalen Datenbank in eine Graphdatenbank sowie die Netzwerkberechnungen werden in Kapitel 5 behandelt. Im Kapitel 6 werden die Ergebnisse der qualitativen Evaluierung und der Implementierung zusammengefasst und diskutiert. Abschließend wird eine Zusammenfassung der wissenschaftlichen Arbeitsergebnisse sowie ein Ausblick gegeben.

KAPITEL 2

THEORETISCHE GRUNDLAGEN

Den Grundstein der Theorie in dieser Arbeit bilden die beiden Datenbanksysteme relationale Datenbanken und Graphdatenbanken. Im Anschluss wird das Thema Netzwerkberechnungen und somit der Begriff Topologie näher beleuchtet. Dabei wird insbesondere auf das Stromnetz eingegangen.

2.1 RELATIONALE DATENBANKEN

Das relationale Datenbankmodell wurde 1970 zum ersten Mal von Codd vorgestellt und erlangte aufgrund der Einfachheit große Popularität [20]. In diesem Kapitel wird zuerst die Struktur von relationalen Datenbanken erläutert, indem das Relationenmodell vorgestellt und in weiterer Folge näher auf die relationale Algebra eingegangen wird. Außerdem werden die Abfragesprache SQL, Transaktionen sowie die Normalisierung von relationalen Datenbanken behandelt. Schließlich werden die Grenzen von relationalen Datenbanken aufgezeigt.

2.1.1 Struktur relationaler Datenbanken

Relationale Datenbanken bauen auf das Relationenmodell auf, ein mathematisches Konzept, mit welchem relationale Datenbanken formal beschreibbar sind. Es speichert Daten in Form von Relationen. Eine n -stellige Relation wird als eine Tabelle mit n Spalten dargestellt. [4, S. 11]

Grundbegriffe

Die wichtigsten Begriffe des Relationenmodells sind Attribut, Domäne, Schema und Instanz.

- *Attribute* spiegeln die Namen der Spalten wider. Sie stellen also die Eigenschaften der Objekte dar.

- Unter *Domäne* wird die Menge der Werte, welche Attribute annehmen können, verstanden. In der Praxis ist das der Datentyp. In jeder Domäne gibt es den speziellen Wert `Null`, welcher für unbekannte Attribute verwendet wird.
- Das *Schema* beschreibt die verwendeten Attribute und Domäne.
- Eine Relation ist dann eine *Instanz* des Schemas.

Die Menge aller verwendeten Schemata ergibt dann das relationale Datenbankschema. Die relationale Datenbank stellt dann das verwendete relationale Datenbankschema mit den aktuellen Werten der Relationen dar. [4, S. 12–14]

Schlüssel

Schlüssel werden zur eindeutigen Identifizierung und Unterscheidung verschiedener Objekte benötigt. Es gibt Primärschlüssel und Fremdschlüssel.

Um einen *Primärschlüssel* zu bilden, muss eine Sequenz von Attributen definiert werden. Über den Primärschlüssel sind die Elemente einer Instanz eindeutig identifizierbar. In der Praxis wird dafür meist ein eigenes Feld als ID angelegt.

Mit *Fremdschlüssel* wird auf eine andere Tabelle referenziert, wodurch verschiedene Schemata in Beziehung gesetzt werden. [4, S. 14–18]

Constraints

Constraints oder Integritätsbedingungen liefern Zusicherungen für Daten. Es gibt 2 Arten von Integritätsbedingungen: statische und dynamische.

Statische Integritätsbedingungen oder auch strukturelle Regeln beschreiben einen Zustand, der erfüllt sein muss. Dazu gehören Unique Constraints, Not Null Constraints, Primary Key Constraints und References Constraints.

- Unter *Unique Constraint* wird verstanden, dass die Werte eines Attributes oder einer bestimmten Attributsequenz eindeutig sein müssen. Das heißt, jede Kombination darf nur einmal vorkommen. Das gilt allerdings nicht für Werte, die `Null` sind.
- Der *Not Null Constraint* besagt, dass die Werte festgelegter Attribute nicht `Null` annehmen dürfen.

- Der *Primary Key Constraint* setzt sich aus dem Unique Constraint und dem Not Null Constraint zusammen. Diese beiden Constraints müssen für alle Attribute des Primary Keys gelten. Dadurch sind alle Elemente einer Relation eindeutig identifizierbar.
- Der *References Constraint* gilt für eine Attributsequenz, die auf einen Primary Key in einem anderen Schema referenziert und gewährleistet, dass nur Fremdschlüssel (Foreign Keys) eingefügt werden, welche tatsächlich im referenzierten Schema vorkommen. Die Attribute im Fremdschlüssel dürfen den Wert Null annehmen, wenn keine Referenz vorhanden und gleichzeitig kein Not Null Constraint gesetzt ist. Somit können konsistente Beziehungen zwischen Schemata hergestellt werden kann.

Dynamische Integritätsbedingungen oder auch Verhaltensregeln müssen bei der Veränderung von Daten erfüllt sein. Das kann beinhalten, ob Daten veränderbar sind oder nicht. [4, S. 18–22]

2.1.2 Relationale Algebra

Die relationale Algebra dient als Abfragesprache des Relationenmodells. Im Folgenden werden die Grundoperationen (Projektion, kartesisches Produkt, Selektion, Umbenennung, Vereinigung und Differenz) näher beschrieben, mit welchen sich aus einer oder zwei Relationen R und S eine neue Relation berechnen lassen. Sie beruhen auf der Mengentheorie. [4, S. 43–44]

- Der *Projektion* Operator $\pi(R)$ ist unär. Er gibt eine neue Relation mit verminderten Attributen zurück.
- Das *kartesische Produkt* $R \times S$ ist eine Menge von geordneten Paaren, welche jedes Element der einen Menge mit jedem Element der zweiten Menge kombiniert.
- Der *Selektion* Operator $\sigma_{\Theta}(R)$ ist unär und gibt anhand einer Bedingung eine verminderte Anzahl an Tupels zurück.
- Der *Umbenennung* Operator

$$\rho_{a/b}(R) = \{t[a/b] : t \in R\} \quad (2.1)$$

ist ein unärer Operator. Hierbei wird die Spalte mit Namen a zu b umbenannt.

- Der *Vereinigung* Operator

$$R \cup S = \{t \mid t \in R \vee t \in S\} \quad (2.2)$$

ist ein binärer Operator, welcher aus zwei Mengen eine Menge zurückgibt. In der neuen Menge sind alle Elemente, welche in zumindest einer ursprünglichen Menge vorkommen.

- Der *Differenz* Operator

$$R - S = R \setminus S = \{t \mid t \in R \wedge t \notin S\} \quad (2.3)$$

ist ein binärer Operator, welcher aus zwei Mengen eine Menge zurückgibt, in welcher nur Elemente aus der ersten Menge, welche nicht in der zweiten Menge vorkommen, sind.

Basierend auf den Grundoperationen lassen sich die beiden weiteren Operationen definieren.

- Der *Durchschnitt* Operator

$$R \cap S = (R \setminus (R \setminus S)) = \{t \mid t \in R \wedge t \in S\} \quad (2.4)$$

ist ein binärer Operator und gibt nur die Elemente zurück, welche in beiden Mengen vorkommen.

- Der *Join* Operator $R \bowtie S$ ist ein binärer Operator, welcher Verbindungen zwischen Relationen erlaubt.

Mit den erwähnten Operatoren lassen sich Abfragen auf das Relationenmodell definieren. Das folgende Kapitel zeigt, wie die Theorie der relationalen Algebra in der Abfragesprache SQL angewandt wird. [4, S. 43–55] [5, S. 72–73]

2.1.3 Abfragesprache SQL

Die Abfragesprache für relationale Datenbanken heißt Structured Query Language (SQL). Sie wurde in den 1970er durch IBM Research entwickelt und ist seit 1987 ein ISO Standard. Mit SQL sind aber nicht nur Abfragen, sondern auch die Definition der Datenstruktur, die Modifikation der Daten in einer Datenbank sowie das Aufbauen von Sicherheitsbedingungen durchführbar. [4, S. 73] [5, S. 73]

SQL Anweisungen können in 3 Kategorien unterteilt werden.

- SQL-Daten Anweisungen, welche Tabellen und Spalten abfragen und ändern. Dazu zählen die Statements:
 - SELECT (fragt Tabellen und Views in der Datenbank ab)
 - INSERT (fügt der Tabelle Zeilen hinzu)
 - UPDATE (ändert Spalten in den Tabellenzeilen)
 - DELETE (löscht Zeilen von der Tabelle)
 - SQL-Transaktion Anweisungen, welche Transaktionen kontrollieren. Dazu gehören:
 - COMMIT (tätigt die aktuelle Transaktion)
 - ROLLBACK (macht die aktuelle Transaktion rückgängig)
 - SQL-Schema Anweisungen, welche das Schema verwalten. Dazu zählen die Statements:
 - CREATE TABLE
 - CREATE VIEW
 - DROP TABLE
 - DROP VIEW
 - GRANT (erteilt Rechte für Tabellen und Views für User)
 - REVOKE (entfernt die Rechte für Tabellen und Views von Usern)
- [5, S. 73]

Einfache Abfrage

Eine einfache, allgemeine Abfrage besteht aus den 3 Klauseln SELECT, FROM und WHERE.

```
1  SELECT Field1, Field2
2  FROM TableA
3  WHERE Bedingung
```

Listing 2.1: Einfache SQL Abfrage

Mit der SELECT Klausel (in relationaler Algebra: Projektion) werden die Attribute, die beim Ergebnis der Abfrage aufgelistet werden sollen, ausgewählt. Mit der FROM Klausel (in relationaler Algebra: kartesisches Produkt) wird bestimmt, welche Relationen für die Abfrage durchgesehen werden. Mit der WHERE Klausel (in relationaler Algebra: Selektion) werden die Attribute gefiltert. [4, S. 75]

Gruppierung und Aggregation

Tupel können auch durch eine GROUP BY Klausel nach Attributen gruppiert werden. Auf die entstehenden Gruppen können dann die Aggregatsfunktionen COUNT, SUM, MIN, MAX und AVG angewandt werden. Mit der HAVING Klausel kann auch eine Selektion nach der Gruppenbildung durchgeführt werden. [4, S. 91]

Die Theorie zur Abfragesprache SQL ist bis hierhin noch lange nicht abgeschlossen. Sie ist aber für den Inhalt dieser Arbeit nicht von Relevanz und somit wird hier nicht weiter in die Tiefe gegangen. In weiterer Folge wird die Abfrageoptimierung noch angeschnitten, da die Performance von Abfragen für diese Arbeit interessant ist.

Abfrageoptimierung

Durch Abfrageoptimierung wird die Ausführung von Abfragen effizienter gestaltet. Das wird vor allem bei Abfragen mit großen Datenmengen tragend.

Eine Möglichkeit, um das Suchen nach Daten zu beschleunigen, liefern *Indizes*. Typische Realisierungen dafür sind B+-Bäume oder Hash-Funktionen.

Des Weiteren ist die Effizienz von Abfragen durch *logische Abfrageoptimierung* steigerbar. Darunter versteht man das Umformulieren einer Abfrage, welche zwar das selbe Ergebnis liefert, aber zum Beispiel kleinere Zwischenergebnisse generiert und somit weniger Speicherplatz benötigt.

Eine dritte Möglichkeit stellt die *physische Optimierung* dar. Damit ist die Auswahl an Algorithmen, welche die Operationen der relationalen Algebra umsetzen, gemeint. Dieser Teil ist stark vom Datenbanksystem abhängig und wird vom Durchschnittsanwender in der Regel nicht beachtet. [4, S. 125]

2.1.4 Transaktionen

Eine Transaktion beschreibt die Folge von Datenbankankweisungen, welche eine Einheit bilden und als diese Einheit ausgeführt werden. Das heißt, wenn im Laufe dieser Ausführung der Datenbankankweisungen ein Fehler auftritt, werden alle bisherigen Schritte rückgängig gemacht. Dadurch befindet sich die Datenbank im Zustand vor Beginn der Transaktion.

Relationale Datenbanken erfüllen die ACID (Atomicity, Consistency, Isolation, Durability) Eigenschaften, welche im folgendem genauer erläutert werden.

- *Atomicity*: Bei einer Transaktion werden entweder alle oder keine der Operationen ausgeführt. Die Änderungen auf der Datenbank sind somit atomar.
- *Consistency*: Der Datenbankzustand ist vor und nach der Transaktion korrekt.
- *Isolation*: Die Abarbeitung einer Transaktion wird auf der Datenbank isoliert durchgeführt. Das heißt, mehrere gleichzeitig ausgeführte Transaktionen beeinflussen sich nicht gegenseitig.
- *Durability*: Die Änderungen von Transaktionen sind von Dauer. Sie können also nicht durch mögliche spätere Fehler verloren gehen. [4, S. 145–146]

2.1.5 Normalisierung

Durch ein schlecht gewähltes Datenbankschema können Anomalien entstehen. In einem solchen Schema werden mehrere Konzepte auf einmal modelliert. Wird das Relationsschema so weit zerlegt, dass pro Schema nur noch ein Konzept abgebildet wird, werden Anomalien vermieden. Man spricht dann von einem Schema in Normalform. Der Prozess dafür wird Normalisierung genannt. [4, S. 165]

Anomalien

Es gibt 3 Anomalien, welche entstehen, wenn ein Schema mehrere Konzepte modelliert.

- Die *Update-Anomalie* tritt auf, wenn ein Update auf einer Tabelle Auswirkungen auf mehrere Zeilen hat. Das kann zu inkonsistenten Daten führen, wenn nicht alle Zeilen geändert werden.
- Die *Insert-Anomalie* tritt auf, wenn Daten nicht vollständig in eine Tabelle eingefügt werden können.
- Die *Deletion-Anomalie* tritt auf, wenn ein Löschvorgang zu unerwarteten Datenverlusten führt.

Ist ein Schema in Normalform, können gewisse Anomalien nicht mehr auftreten. [4, S. 165–166]

Funktionale Abhängigkeit

Die funktionale Abhängigkeit beschreibt die Beziehung zwischen Attributen innerhalb einer Tabelle und definiert, wie ein Wert eines Attributs den Wert eines anderen Attributs bestimmt. [4, S. 167]

Die Normalisierung von relationalen Datenbanken zielt darauf ab, die funktionalen Abhängigkeiten innerhalb der Daten zu identifizieren und darauf aufbauend die Daten in mehrere Tabellen aufzuteilen. Damit werden Redundanzen und Anomalien vermieden. Dies wird durch den Einsatz von Normalformen (1NF, 2NF, 3NF, BCNF usw.) erreicht, welche festlegen, wie die funktionale Abhängigkeiten innerhalb der Daten berücksichtigt werden müssen. [4, S. 177]

Normalformen

Eine Relation ist in *1. Normalform (1NF)*, wenn jedes Attribut nur atomare Werte annimmt. Die Werte der Attribute stellen auch keine Arrays dar. Werte, welche sich wiederholen, werden in einer extra Tabelle gespeichert. Alle zusammengehörigen Daten werden mit einem Primary Key identifiziert.

Eine Relation in *2. Normalform (2NF)* erfüllt die Kriterien der 1NF und setzt voraus, dass nonkey Attribute ganz vom Primary Key abhängen. Das heißt, dass in einer Tabelle nur Daten zu einer Entität in der echten Welt vorliegen sollen. Um die 2NF umzusetzen, müssen zusätzlich Tabellen für die Menge an Werten, welche funktional abhängig sind, erstellt werden.

Eine Relation ist in *3. Normalform (3NF)*, wenn sie in 2NF ist und wenn alle nonkey Attribute untereinander unabhängig sind. Für die 3NF müssen also alle Felder, welche nicht vom Primary Key abhängen, eliminiert werden. [5, S. 69–71]

Zusätzlich gibt es noch die Boyce-Codd Normalform (BCNF), die 4. Normalform (4NF) und die 5. Normalform (5NF). Diese stellen in dieser Reihenfolge noch strengere Anforderungen an die Relation. In der Praxis wird jedoch der Normalisierungsprozess meist bei der 3. Normalform abgeschlossen. Je höher die Normalform, desto höher ist das Niveau der Normalisierung, aber auch desto komplexer ist das Datenmodell und desto mehr Tabellen sind erforderlich. [21, S. 161]

2.1.6 Grenzen relationaler Datenbanken

Relationale Datenbanken gelangen an ihre Grenzen, wenn die Datenmenge die Kapazität einer physischen Maschine übersteigt. Das Konzept der relationalen Datenbanken wurde zu einer Zeit entwickelt, wo Programme auf nur einer Maschine liefen. Es gibt heute schon auch relationale Datenbanken im Clusterverbund, jedoch implizieren die Einhaltung der ACID Kriterien einen erhöhten Aufwand, was sich auf die Performance auswirkt. Beispielsweise wird bei einem Aktiv-Aktiv Cluster von MS SQL Server die Eintragung von neuen Datensätzen von allen Servern bestätigt, was zu einer erhöhten Latenzzeit führt.

Des Weiteren gibt es auch analytische Applikationen, zum Beispiel die Regressionsanalyse, bei der die normalisierte und dadurch atomare Speicherung von Daten zu komplexeren und aufwändigeren Operationen führt. [6, S. 110–111]

Zu den Grenzen bezogen auf Skalierbarkeit, Flexibilität des Datenmodells und Performance bei Netzberechnungen wird in Kapitel 4 Qualitative Evaluierung näher ins Detail gegangen.

2.2 GRAPHDATENBANKEN

Graphdatenbanken gehören zu den NoSQL Datenbanken, welche zu Beginn dieses Kapitels beleuchtet werden, und bauen auf die Graphentheorie auf. Basierend darauf werden in weiterer Folge typische Graphdatenbankanwendungen veranschaulicht und die Datenmodellierung mit Graphdatenbanken beschrieben. Letztendlich wird die Graphdatenbank Neo4j mit ihrer Abfragesprache Cypher vorgestellt.

2.2.1 NoSQL

NoSQL steht für Not only SQL und ermöglicht das Speichern und Verarbeiten von Daten außerhalb des klassischen relationalen Schemas. Die beliebige Erweiterung von Datenstrukturen gehört zu den größten Vorteilen von NoSQL Datenbanken gegenüber den relationalen Datenbanken. [6, S. 111 ff.].

NoSQL Datenbanken werden als Alternative zu relationalen Datenbanken gesehen. Im Allgemeinen sind sie unstrukturiert, das heißt, sie besitzen kein fixes Schema. [9] NoSQL Datenbanken sind vor allem für Anwendungen mit hoher Anzahl an Read und Write Operationen, wo sich auch das Schema ändern kann, geeignet. Der Fokus der meisten NoSQL Systeme liegt auf hoher Performance, hohe Verfügbarkeit, Datenreplikation und Skalierbarkeit. [8]

NoSQL Datenbanken werden in 4 Kategorien unterteilt, welche im Folgenden näher erläutert werden und in Abbildung 2.1 bildlich dargestellt sind.

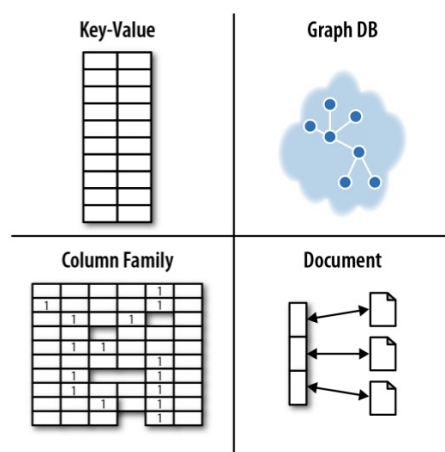


Abbildung 2.1: NoSQL Stores (Quelle: [7, S. 197])

Key-Value Stores

In Key-Value Datenbanken werden die Daten in Form von Zeilen, JSON oder auch ein selbst definiertes Datenformat unter einem Schlüssel (Key) gespeichert. Der Schlüssel ist wiederum in einer Hash-Tabelle abgelegt. Diese Datenbanken eignen sich für schnelles und effizientes Datenmanagement auf verteilten Systemen. Zu den bekanntesten Beispielen zählt die DynamoDB von Amazon. [8]

Column Family Systeme

In Column Family Systemen, auch Wide-column Datenbanken genannt, werden die Daten in Spalten gespeichert, anstatt in Zeilen wie bei relationalen Datenbanken. Das erlaubt erhöhte Flexibilität in der Datendefinition. [9]

BigTable wurde von Google für Gmail, Google Maps und Website Indexing entwickelt und zählt zu einer der ersten Technologien, welche dieses Schemamodell verwendet. [8]

Document Stores

Document-oriented oder auch Document-based Datenbanken zählen zu den meist genutzten NoSQL Datenbanken. Dabei werden Daten in Dokumenten gespeichert, welche eine Reihe an Felder mit Attributen beinhalten. Die gängigsten Formate hierfür sind XML, JSON oder BSON. Document-oriented Datenbanken funktionieren ähnlich wie Key-Value Datenbanken, hierbei entspricht der Key der Document-ID und der Value dem Dokument (JSON, XML, ...). Die Abfragen werden an die Dokumentenfelder gestellt. Beispiele für Document-based Datenbanken sind MongoDB und CouchDB[8], [9]

Graphdatenbanken

Bei Graphdatenbanken werden die Daten in einer Graphstruktur gespeichert, also durch Knoten und Kanten repräsentiert. [9] Auf dieses Konzept wird in den Kapiteln 2.2.2 bis 2.2.8 noch genauer eingegangen.

2.2.2 Graphentheorie

Die Entstehung der Graphentheorie wird auf Euler (1736) und dem Königsberger Brückenproblem zurückgeführt. Es basiert auf der Topographie der Stadt Königsberg (heute Kaliningrad in Russland) und beweist, dass es aufgrund der ungeraden Anzahl an Brücken, welche die 4 Landteile verbinden, keinen Rundweg gibt, der alle 7 Brücken genau einmal überquert [5, S. 316].

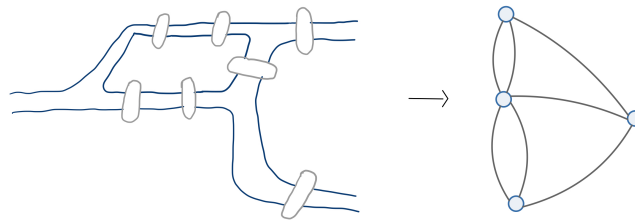


Abbildung 2.2: Königsberger Brückenproblem

Die Graphentheorie ist ein Zweig der Mathematik, welcher sich in der zweiten Hälfte des letzten Jahrhunderts rasant ausbreitete und mittlerweile auch ein wichtiger Teil der Informatik geworden ist. [11, S. 345]

Ein *Graph* $G = (V, E)$ besteht aus einer Menge an *Knoten* (engl.: vertices) $V = \{v_1, \dots, v_n\}$ und einer Menge an *Kanten* (engl.: edges) $E = \{e_1, \dots, e_m\}$. Jede Kante $e = (i, j)$ wird durch eine Inzidenzabbildung ein Knotenpaar zugeordnet. Wenn das Knotenpaar ungeordnet ist, wird von einem *ungerichteten* Graphen gesprochen. Wenn die Kante eine Richtung hat, spricht man von gerichteten Kanten und einem *gerichteten* Graphen oder auch *Digraph*. Die Knoten einer ungerichteten Kante werden *Endknoten* genannt. Bei einer gerichteten Kante nennt man die Knoten *Anfangs-* beziehungsweise *Endknoten*. Zwei Kanten sind *parallel*, wenn zwei gerichtete Kanten denselben Anfangs- und Endknoten haben. Geht eine Kante $e = (i, i)$ von Knoten i zum selben Knoten i , bildet sie eine *Schlinge*. Ein gerichteter Graph heißt *schlicht*, wenn er keine gerichteten Kanten und keine Schlingen aufweist. [10, S. 120] Ein Graph heißt *gewichtet*, wenn Kanten Informationen zugeordnet sind. Das kann beispielsweise bei einem Straßennetzwerk die Distanz oder die geschätzte Fahrzeit sein. [12, S. 102] Ist ein Knoten Anfangs- oder Endknoten einer Kante, so sind der Knoten und die Kante *inzident*. Zwei Knoten, die mit derselben Kante inzident sind, heißen *benachbart* oder *adjazent*. [10, S. 120]

Entitäten können mit Knoten und die Beziehung zwischen Entitäten können mit Kanten dargestellt werden. Mit dieser Struktur können unter-

schiedlichste Situationen abgebildet werden. Zu den bekanntesten Anwendungsbeispielen zählen soziale Netzwerke und Straßennetzwerke, aber auch Lieferketten, der Bau einer Rakete oder eine Anamnese für Populationen ist mit der Graphentheorie modellierbar [7, S. 1].

2.2.3 Traversieren von Graphen

Traversieren von Graphen bedeutet das systematische Besuchen aller Knoten eines Graphens. Dies ist für effiziente Graph-Algorithmen notwendig. Zwei wichtige Methoden dafür sind die Tiefensuche (engl. Depth-First-Search, kurz DFS) und die Breitensuche (engl. Breadth-First-Search, kurz BFS).

- Die *Tiefensuche* ist eine rekursive Methode, die zuerst alle Nachfolger eines Knotens durchgeht, bevor der Algorithmus zum nächsten Knoten zurückkehrt. Sie durchsucht den Graph also nach dem Motto zuerst in die Tiefe.
- Die *Breitensuche* ist eine iterative Methode, bei der die Knoten auf der gleichen Ebene (also auf gleicher Tiefe) vor den Knoten auf tieferen Ebenen besucht werden. Das Motto lautet hierbei zuerst in die Breite.

Mit Traversierungen lassen sich Graphenprobleme wie zum Beispiel die topologische Sortierung, die Frage, ob ein Knoten von einem anderen aus erreichbar ist, die Suche nach dem kürzesten Weg oder der Überprüfung auf Zyklen lösen. [12, S. 297, 311]

2.2.4 Graphdatenbanken

Graphdatenbanken gehören wie bereits erwähnt zu den NoSQL Datenbanken. Anders als bei relationalen Datenbanken, wo die Beziehungstabelle im Maximalfall exponentiell mit der Anzahl der Knoten wächst, werden bei den Graphdatenbanken die Knoten mit den jeweils dazugehörigen Kanten als verschachtelte Listen abgespeichert. Somit ist klar, welcher Knoten mit welchen Knoten wie in Verbindung steht. Außerdem werden weder eine Beziehungstabelle noch Indizes für die Nachbarbestimmung benötigt. Es können auch an jeder beliebigen Stelle neue Knoten oder Kanten dem Graphen hinzugefügt werden. Abfragen holen nicht alle Informationen einer Entität oder Relation ab, sondern starten bei einem Knoten und wandern dann durch den Graph. [6, S. 122].

Bei Graphdatenbanken werden im Gegensatz zur Graphentheorie Knoten meist als Nodes und Kanten als Relations bezeichnet. Daher wird in weiterer Folge auch dieser Terminus verwendet.

2.2.5 Typische Graphdatenbankanwendungen

Graphdatenbanken eignen sich also vor allem für Anwendungen, für welche Konnektivität und Topologie wichtig ist. Die meisten Anwendungen wären auch mit relationalen Datenbanken umsetzbar, jedoch aufgrund mehrerer JOIN Operationen weniger effizient.

Zu den klassischen Anwendungsfällen zählen an erster Stelle soziale Netzwerke. Hierbei werden die Beziehungen (Relations) zwischen einzelnen Personen (Nodes) quantifiziert. Die dabei auftretenden rekursiven Datenstrukturen sind mit Graphdatenbanken performanter abbildbar als mit relationalen Datenbanken. Des Weiteren zählen Empfehlungssysteme zu den typischen Anwendungen, welche von Graphdatenbanken profitieren. Sie empfehlen den Nutzern basierend auf diversen Informationen, welche Produkte ihnen gefallen könnten. Dies findet vor allem bei den Werbungen von Google oder den Freundesvorschlägen von Facebook bekannte Anwendungsfälle. Auch die Bioinformatik bietet ein weiteres klassisches Anwendungsgebiet für Graphdatenbanken. Hier werden Informationen zu Genen, Proteinen, und Enzymen in Verbindung gesetzt. [22] Graphdatenbanken werden aber auch in Bereichen wie in der Medizin, Einzelhandel, Öl und Gas, Medien und Gaming verwendet. [7, S. viii].

Viele Firmen haben ihre eigenen Graphdatenbank Implementationen entwickelt. Beispiele hierfür sind Open Graph von Facebook, Knowledge Graph von Google oder FLockDB von Twitter. [22]

2.2.6 Datenmodellierung mit Graphdatenbanken

Im Allgemeinen ist Modellierung die Abstrahierung der realen Welt, angetrieben durch ein bestimmtes Ziel. Bei Repräsentationen mit Graphen besteht im Vergleich zu anderen Modellierungstechniken eine große Ähnlichkeit zwischen dem logischen und physischen Modell. Vor allem bei relationalen Datenmanagement-Techniken kommt es zu großen Abweichungen zu unserer natürlichen Sprache [7, S. 26].

Labeled Property Graph Model

Die bekannteste Form eines Graphenmodells heißt *Labeled Property Graph* und weist die folgenden Merkmale auf. Ein Graph setzt sich aus Nodes und Relations zusammen. Nodes besitzen Eigenschaften und können mit einem oder mehreren Labels gekennzeichnet werden. Relations stellen Beziehungen dar und sind gerichtet, haben also einen Start- und einen Endnode. Zudem haben auch Relations Eigenschaften und werden mit Labels gekennzeichnet. [7, S. 4].

Abbildung 2.3 zeigt ein einfaches Beispiel für ein Labeled Property Graph Model.

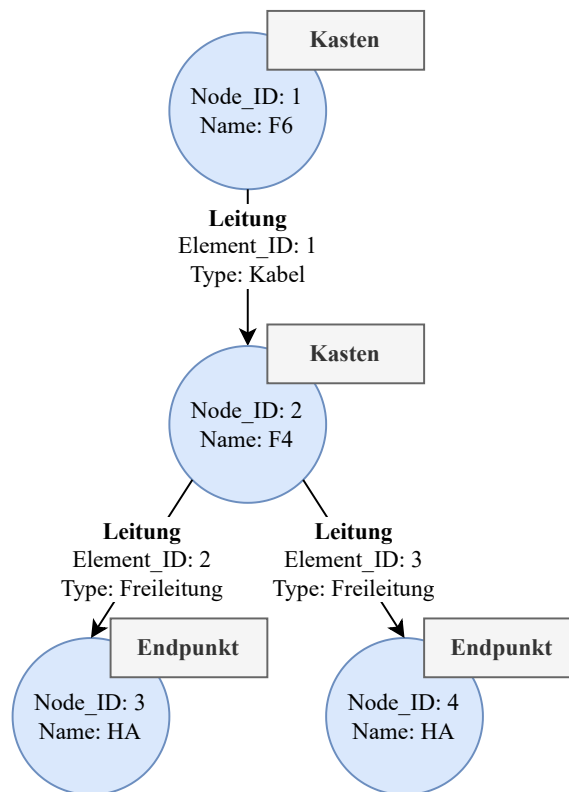


Abbildung 2.3: Labeled Property Graph Model

Vergleich mit relationalen Datenbanken

Im folgenden werden die Elemente von relationalen Datenbanken mit denen der Graphdatenbanken gleichgesetzt. Tabellen in relationalen Datenbanken können mit Graphen in Graphdatenbanken verglichen werden. Für die Zei-

len bei relationalen Datenbanken stehen bei Graphdatenbanken die Nodes, deren Attribute und Werte den Spalten in den Tabellen entspricht. Der Join Operator bei relationalen Datenbanken kann mit der Traversierung verglichen werden. [19]

Das impliziert, dass Graphdatenbanken für Abfragen zu Konnektivität und Topologie effizienter sind [22]. Durch den Join Operator werden zur Abfragezeit Primary Key und Foreign Key gematcht, was sehr rechen- und speicherintensiv ist und exponentielle Kosten mit sich bringt [23]. Die Architektur von Neo4j ist hingegen für schnelles Management, Speicherung und Traversierung von Nodes und Relations ausgelegt [22].

2.2.7 *Neo4j*

Da die Umsetzung dieser Masterarbeit mit Neo4j erfolgt, wird hier näher auf die Graphdatenbank Neo4j eingegangen, welche zu den bekanntesten Graphdatenbanken zählt [6, S. 122].

Die Vision von Neo4j war es, die geprüften Features von relationalen Datenbanken wie Transaktionen, ACID und Triggers zu behalten, jedoch das Datenmodell an die Graphentheorie anzupassen um die erwähnten Nachteile von relationalen Datenbanken für verbundene Daten zu eliminieren [7, S. viii]. Die Datenbank kann entweder über eine API für verschiedene Programmiersprachen angesprochen werden oder über das WebGUI mittels der dazugehörigen Abfragesprache Cypher [6, S. 123].

2.2.8 *Abfragesprache Cypher*

Cypher ist eine Graphdatenbank-Abfragesprache mit großer Affinität zur Repräsentation von Graphen als Diagramme. Das macht Cypher einfach zu lernen, lesen und zu verstehen. Cypher ist zwar auf Neo4j spezialisiert, bietet aber dennoch eine gute Basis für alle weiteren Abfragesprachen für Graphdatenbanken. Im Fachbuch *Graph databases* wird Cypher als das Mittel beschrieben, um die Datenbank zu fragen, ob es Daten gibt, die ein spezifisches Muster matchen. [7, S. 27–28]

So wie SQL besteht auch Cypher aus mehreren Klauseln. Eine einfache, allgemeine Abfrage besteht aus einer MATCH Klausel, auf welche eine RETURN Klausel folgt. Der folgende Codeausschnitt zeigt die einfachste Cypher Abfrage für Neo4j. Dabei werden alle Nodes mit den dazugehörigen Relations der Datenbank zurückgegeben. [7, S. 29]

```

1 MATCH (n)
2 RETURN n

```

Listing 2.2: Einfache Cypher Abfrage

Die MATCH Klausel ist sozusagen das Herzstück jeder Abfrage. Es beinhaltet den Teil, durch welchen Daten zu einer Spezifikation gefunden werden sollen. Mit der RETURN Klausel wird festgelegt, welche Nodes, Relations oder Eigenschaften von den gematchten Daten zurückgegeben werden sollen.

Für die Darstellung von Nodes und Relations werden in Cypher ASCII-Zeichen verwendet. Nodes werden mit runden Klammern gezeichnet und Relations mit Paaren von Strichen und je nach Richtung mit Größer-als- oder Kleiner-als-Zeichen gezeichnet. Zwischen den beiden Strichen wird in eckigen Klammern mit vorgesetztem Doppelpunkt der Relationname angegeben. Analog wird der Nodename in den runden Klammern mit einem vorangesetzten Doppelpunkt definiert. Vor dem Doppelpunkt kann ein Identifier angeführt werden. Die Key-Value-Paare der Eigenschaften von Nodes und Relations werden in geschwungenen Klammern angegeben. [7, S. 30–31]

Der folgende Codeausschnitt zeigt eine erweiterte Cypher Abfrage.

```

1 MATCH (k:Kasten {Name:"F4"})-[:Leitung]->(e:Endpunkt)
2 RETURN k, e

```

Listing 2.3: Erweiterte Cypher Abfrage

Dabei wird nach einem Node mit dem Label **Kasten** und der Eigenschaft **Name="F4"** gesucht. Das Ergebnis dazu wird in den Identifier **k** gespeichert. Mit diesem Identifier kann für die restliche Abfrage auf diesen Node referenziert werden. Der Startnode **k** ist Teil des Patterns **(k:Kasten {Name:"F4"})-[:Leitung]->(e:Endpunkt)**, welches noch alle weiteren Nodes mit Label **Endpunkt**, welche mit der Relation mit Label **Leitung** von Node mit Identifier **k** wegzeigen, auf den Identifier **e** bindet. Zurückgegeben werden im Codebeispiel alle Nodes, welche an die Identifier **k** und **e** gebunden sind.

Im folgenden werden noch weitere Abfrageklauseln von Cypher aufgelistet.

- WHERE (bietet Kriterien zum Filtern der Ergebnisse)
- CREATE und CREATE UNIQUE (erstellt Nodes und Relations)
- MERGE (stellt sicher, dass das angegebene Muster im Graphen vorhanden ist, indem entweder vorhandene Nodes und Relations wiederverwendet oder neue Nodes und Relations erstellt werden)

- DELETE (löscht Nodes, Relations und Eigenschaften)
- SET (setzt Eigenschaftswerte)
- FOREACH (führt eine Aktualisierungsaktion für jedes Element in einer Liste durch)
- UNION (vereint die Ergebnisse von zwei oder mehreren Abfragen zusammen)
- WITH (verkettet aufeinanderfolgende Abfrageteile und leitet die Ergebnisse von einer Abfrage zur nächsten weiter) [7, S. 31]

Weitere bekanntere Abfragesprachen für Graphdatenbanken wären SPARQL und Gremlin. SPARQL ist eine RDF Abfragesprache und Gremlin zählt als eine imperative, pfadbasierte Abfragesprache. [7, S. 27]

2.3 NETZWERKBERECHNUNGEN

Um Netzwerkberechnungen näher zu erklären, wird in diesem Kapitel zuerst auf das zugrundeliegende GI Datenmodell Vektordaten, insbesondere auf den Unterschied zwischen Simple Features und Topologische Feature, eingegangen. Darauf aufbauend werden die Kennzeichen eines Netzwerkdatenmodells beschrieben. Abschließend wird das Niederspannungsstromnetz und dessen gängigen Topologien beleuchtet.

2.3.1 Vektordatenmodell

Das Vektordatenmodell eignet sich vor allem für die Repräsentation von diskreten Objekten. Zu den Vorteilen von Vektordaten zählen unter anderem die Speichereffizienz, qualitative kartographische Ergebnisse sowie Werkzeuge für Kartenprojektionen, Overlay-Prozesse oder kartographische Analysen.

Im Vektordatenmodell werden die Objekte im 2-dimensionalen Raum in 3 Kategorien, den Geometric Primitives, unterteilt: Punkt, Linie und Polygon. Dabei wird ein Punkt als Koordinatenpaar gespeichert, eine Linie besteht aus einer Reihe geordneter Koordinatenpaare und ein Polygon wird aus einer oder mehreren Liniensegmenten, welche geschlossen sind, dargestellt.

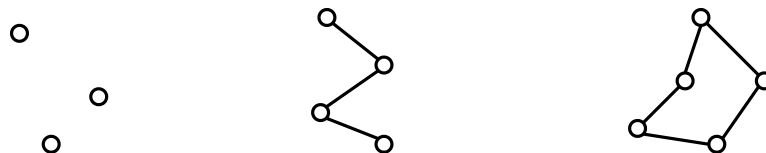


Abbildung 2.4: Punkt, Linie und Polygon

Üblicherweise werden die Objekte im Vektordatenmodell als Features bezeichnet. Das Vektordatenmodell kann in Simple Features, Topologische Features und Netzwerkdatenmodell unterteilt werden. [13, S. 158]

2.3.2 OGC Simple Features

Simple Features Access (ISO 19152) zählt zu den OGC Implementation Standards und beschreibt Simple Features in den folgenden 2 Teilen.

- Common Architecture: Der erste Teil beschreibt ein Datenmodell für Simple Features. Simple Features werden als Features mit 2D-Geometrie

beschrieben und in die Klassen Point, LineString, Line, LinearRing, Polygon, MultiPoint, MultiLineString, MultiPolygon und GeometryCollection unterteilt. Für diese Klassen werden Zugriffsmethoden, Spatial Analysis Methoden (beispielsweise Buffer, Union), Methoden für Topologietests (beispielsweise Intersects, Contains, Touches) sowie weitere Basismethoden wie zum Beispiel minimum bounding rectangle zur Verfügung gestellt.

- SQL Option: Der zweite Teil beschreibt ein SQL Schema, welches Speicherung, Abfrage und Aktualisierung von Features unterstützt.

Dies ermöglicht Datenaustausch und Interoperabilität von Simple Features zwischen unterschiedlichen Anwendungen. [5, S. 81–82, 546] Für GI Anwendungen bringt vor allem die einfache Erstellung und Speicherung sowie das schnelle Abrufen und Darstellen von Simple Features Vorteile. Allerdings sind aufgrund der einfachen Datenstruktur keine komplexeren Abfragen wie Netzwerkanalysen möglich. [13, S. 158–159]

2.3.3 Topologische Features

Topologische Features sind grundsätzlich Simple Features, welche mit topologischen Regeln strukturiert werden. Topologie ist die Wissenschaft von geometrischen Beziehungen. Unter geometrische Beziehungen werden hierbei nicht-metrische Eigenschaften von geographischen Objekten, welche auch bei einer Verzerrung des Raumes bestehen bleiben, verstanden. Wenn zum Beispiel eine Karte gezerzt wird, ändern sich Eigenschaften wie Distanz und Winkel, die topologischen Eigenschaften wie beispielsweise Adjazenz bleiben jedoch unverändert.

Für topologische Daten gibt es mehrere Validierungstests:

- Netzwerkkonnektivität: Alle Netzwerkelemente müssen verbunden sein, um einen Graph zu bilden. Das heißt, verbundene Elemente müssen gesnappt werden, was bedeutet, dass sie einen gemeinsamen Koordinatenwert aufweisen.
- Linienschnittpunkte: Es muss unterscheidbar sein, ob sich zwei Linien schneiden und es somit einen Schnittpunkt gibt (zum Beispiel eine Straßenkreuzung) oder ob sich die zwei Linien nicht kreuzen (zum Beispiel bei einer Brücke).
- Überlappung: Es ist wichtig, klar zu definieren, welche Polygone sich wie überlappen. Beispielsweise wird diese Validierung im Grundbuch tragend, wo der Besitz von Grundstücken eindeutig sein soll.

- Mehrfache Linien: In einem sauberen topologischen Netzwerk dürfen keine Linien doppelt oder mehrfach vorkommen, welche jedoch oft im Datenerfassungsprozess entstehen können. [13, S. 159–160, 162]

Die Topologie kann auf 2 unterschiedliche Arten gebildet werden. Einerseits gibt es sogenannte *Batch topology builders*, welche aus CAD, Vermessungsdaten, Simple Features oder andere unstrukturierte Daten über einen meist iterativen Prozess eine Topologie generieren. In der Regel müssen die Daten dann im Anschluss noch manuell korrigiert werden. Andererseits gibt es noch die interaktive Topologiebildung, welche beim Hinzufügen von neuen Objekten in die Datenbank die Topologie bildet. Das geschieht in der GIS Bearbeitungssoftware. [13, S. 206]

2.3.4 Netzwerkdatenmodell

Das Netzwerkdatenmodell gehört zum Vektordatenmodell und ist eine spezieller Fall des Topologischen Featuremodells. Bei Netzwerken wird zwischen den zwei Typen strahlenförmig und ringförmig unterschieden. Bei strahlenförmigen Netzen, auch Baumnetze genannt, gibt es immer eine Stromaufwärtsbeziehungsweise Stromabwärtsrichtung. Klassische Beispiele hierfür wären Regenentwässerung- oder Fließsysteme. Bei ringförmigen Netzen gibt es Verschneidungen. Ein Beispiel hierfür wären Wasserverteilnetze, welche bewusst so aufgebaut wurden, um auch im Störfall möglichst viele Endpunkte zu versorgen.

Modelliert werden Netzwerke mit einer Knoten-Kanten-Struktur, welche bereits aus Kapitel 2.2.2 bekannt ist. Kanten können dabei auch noch gewichtet werden. [13, S. 162]

2.3.5 Topologien des Niederspannungsstromnetzes

Für das Ortsnetz auf Niederspannungsebene gibt es theoretisch drei unterschiedliche Topologien: das Strahlennetz, Ringnetz und Maschennetz. Praktisch gibt es auch Mischformen davon. [14, S. 463]

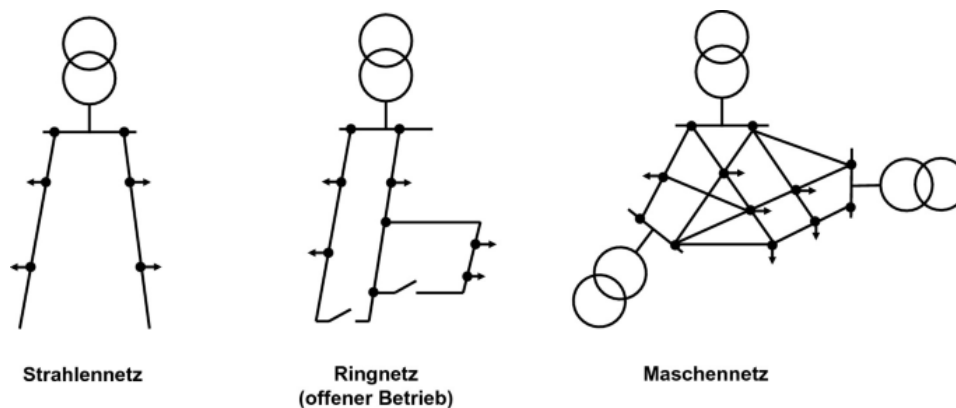


Abbildung 2.5: Topologieformen von Ortsnetzen (Quelle: [24, S. 61])

Strahlennetz

Bei einem Strahlennetz verlaufen die Versorgungsleitungen strahlenförmig von der Einspeisung weg. Eine Straße kann zum Beispiel von einem Strahl versorgt werden. Zu den Vorteilen von Strahlennetzen zählen der geringe Planungsaufwand, die große Übersichtlichkeit bei der Fehlersuche und die geringen Anforderungen an den Netzschutz. Zu den Nachteilen gehören der Ausfall aller Verbraucher bei einem Kurzschluss sowie die bei zunehmenden Abstand zur Einspeisung größer werdenden Spannungsabfälle, Leistungsverluste und die erschwerte Einhaltung der Nullungsbedingungen. [14, S. 445] Aufgrund der einfachen und übersichtlichen Betriebsweise werden die meisten Ortsnetze als Strahlennetz betrieben [14, S. 463].

Ringnetz

Bei Ringnetzen ist eine betriebsmäßige Verbindung der Enden zweier Strahlen möglich. Im Falle einer Störung wird nur der betroffene Halbring vom Netzschutz abgeschaltet. Nach dem manuellen Herausnehmen des betroffenen Leitungsstückes und Zuschalten der betrieblich vorgesehenen Querverbindungen, können die restlichen Verbraucher versorgt werden. Der Vorteil von Ringnetzen ist also die höhere Versorgungssicherheit. Bei geschlossenen Ringen sind auch die Spannungshaltung verbessert sowie die Verluste vermindert. Allerdings muss das Wartungspersonal besser qualifiziert sein. [14, S. 445–446]

Maschennetz

Bei einem Maschennetz werden die Abgänge mehrfach versorgt. Dadurch ist eine große Versorgungssicherheit gegeben, da bei einem Kurzschluss nur kleine Bereiche betroffen sind. Je nach Größe des Maschennetzes erfolgt die Einspeisung durch einen oder mehrere Transformatoren im Zentrum des Netzes. Das Maschennetz wird dann einfach beziehungsweise mehrfach gespeistes Maschennetz genannt. [14, S. 446–447] Im Maschennetz ist es aufgrund der Speisung von mehreren Netzknoten am einfachsten, die Versorgungssicherheit, die Spannungserhaltung und den Lastausgleich zu gewährleisten. [14, S. 463]

KAPITEL 3

UMSETZUNG

In diesem Kapitel wird zu Beginn die gewählte Methodik ausführlicher erläutert. Im Anschluss wird die Ausgangslage der Umsetzung, also die gegebenen Daten, näher beschrieben. Zuletzt werden die Anforderungen von EVU für die qualitative Evaluierung aufgezeigt.

3.1 METHODIK

Die Umsetzung setzt sich aus zwei Teilen zusammen. Zum einen wird eine qualitative Evaluierung zwischen relationalen Datenbanken und Graphdatenbanken durchgeführt. Zum anderen wird die Überführung der Daten aus der relationalen Datenbank in eine Graphdatenbank durchgeführt und darauf aufbauend Abfragen zu Netzberechnungen erstellt.

Als qualitative Evaluierung werden die Systeme relationale Datenbanken und Graphdatenbanken anhand der Kriterien Sicherheit, Einfachheit der Programmierung und Abfragesprache, Flexibilität, Performance von Netzberechnungen, Skalierbarkeit sowie Ausgereiftheit und Support verglichen. Die Kriterien wurden gewählt, da sie als gängige Eigenschaften für Datenbanksysteme gelten. Zuerst werden die Kriterien im Kapitel 4 allgemein ausgearbeitet und im Anschluss in Kapitel 6.2 basierend auf den Anforderungsszenarien der EVU bewertet. Um die Anwendungsanforderungen zu erarbeiten, wird ein Experteninterview in Form eines semistrukturierter Interviews, welches als qualitatives Interview eingeordnet werden kann, durchgeführt. Die Zusammenfassung dazu befindet sich in Kapitel 3.3.

Für den Überführungsteil werden die Ausgangsdaten in einer relationalen Datenbank von der Energienetze Steiermark GmbH (EN) zur Verfügung gestellt. Im Implementierungsteil im Kapitel 5 wird aufgezeigt, welche Schritte für eine Überführung notwendig sind und was die Anforderungen an die Daten im relationalen System sind. Mit den Daten in der Graphdatenbank werden dann im Anschluss die Anwendungsfälle Abzweigeinfärbung, Ausfall einer Leitung und Netzverfolgung erläutert. Die Umsetzung wird dokumentiert und die entscheidenden Schritte werden hervorgehoben. Gegebenenfalls

werden Schwierigkeiten und Hindernisse klar deklariert.

3.2 AUSGANGSLAGE

Derzeit liegen die Netzwerkdaten der EN als OGC Simple Features in einer MS SQL Server Datenbank. Simple Features bedeutet, dass keine topologische Information der Daten gespeichert ist. Zu den unterschiedlichen Netzwerkkomponenten wie Leitung, Station, Abzweig, Kasten, Endpunkt usw., welche näher in Kapitel 5.1.1 beschrieben werden, gibt es jeweils eigene Tabellen.

Um topologische Netzwerke zu erstellen, gibt es FME Jobs, welchen die Geometrie der Netze übergeben werden und in einer SQLite Datenbank resultieren. Auf die FME Jobs wird im Rahmen dieser Masterarbeit nicht näher eingegangen. Die SQLite Datenbank stellt somit die Datengrundlage für diese Arbeit dar und besteht aus den Tabellen Terminal, Node und Element. In der Node Tabelle befinden sich alle Netzwerkknoten mit einer Node_ID. In der Element Tabelle sind alle Leitungen mit einer Element_ID aufgelistet. Die Terminal Tabelle enthält die Information, mit welchen Leitungen die Knoten verbunden sind. Eine detailliertere Beschreibung der SQLite Tabellen befindet sich im Kapitel 5.3.

3.3 ANFORDERUNGEN EVU

In der Tabelle 3.1 werden die Anforderungen von EVU an ein Datenbanksystem diskutiert. Dafür wurde ein Experteninterview mit dem Abteilungsleiter der Geoinformationsabteilung der EN Dipl.-Ing. Werner Samhaber durchgeführt. Das dazugehörige Transkript befindet sich im Anhang .1.

Tabelle 3.1: Anforderungen EVU

Sicherheit	EVU zählen zur kritischen Infrastruktur, wodurch die Sicherheit ein sehr wichtiges Thema und mehr oder weniger die Grundvoraussetzung ist. Deshalb werden die Systeme auch laufend hinsichtlich IT-Sicherheit nach den ISO 27001 Richtlinien geprüft.
Einfachheit der Programmierung und Abfragesprache	Die Einfachheit der Programmierung und der Abfragesprache ermöglicht schnellere Anpassungen und Verbesserungen am System. Das ist aufgrund von Kundenanforderungen sowie der Dynamik der Entwicklungen im Energieumfeld wichtig.
Flexibilität	Aufgrund aktueller Digitalisierungsinitiativen mit agilen, modernen Entwicklungsumgebungen ist die Flexibilität und Anpassbarkeit von eingesetzten Softwarelösungen von Vorteil.
Performance von Netzberechnungen	Die Performance von Netzberechnungen ist vor allem für die Anwendbarkeit der Softwarelösungen wichtig, da Anwender performante Systeme erwarten.
Skalierbarkeit	Die Wichtigkeit der Skalierbarkeit hängt von der Datenmenge ab. Im regionalen Bereich sind die Datenmengen eher klein und somit ist die Skalierbarkeit aktuell weniger relevant. Für andere zukünftige Anwendungsbereiche wäre das Kriterium Skalierbarkeit zukünftig schon beachtenswert.
Ausgereiftheit und Support	Auch die Ausgereiftheit eines Systems sowie der dazugehörige Support ist für EVU notwendig, da die Systeme meist einige Jahre im Betrieb sind.

KAPITEL 4

QUALITATIVE EVALUIERUNG

Bei der qualitativen Evaluierung wurden die Kriterien Sicherheit, Einfachheit der Programmierung und Abfragesprache, Flexibilität, Performance von Netzwerkberechnungen, Skalierbarkeit sowie Ausgereiftheit und Support gewählt. In diesem Kapitel werden die Kriterien aus neutraler Sicht beleuchtet und anschließend im Kapitel Ergebnisse 6.1 zusammengefasst und aus Sicht von Energieversorgungsunternehmen (EVU) bewertet.

4.1 SICHERHEIT

Datensicherheit ist ein wichtiges Kriterium von Datenbanksystemen, vor allem wenn es um persönliche Daten oder Daten aus kritischer Infrastruktur geht.

Für relationale Datenbanken gibt es ausgereifte Mechanismen, welche die Sicherheit der Services gewährleisten. Dennoch gibt es bei relationalen Datenbanken Gefahren wie SQL Injection, cross-site scripting, rootkits und schwache Kommunikationsprotokolle. [8], [17]

Bei NoSQL Datenbanken kommt es aufgrund von Sharding, dem Verteilen der Daten auf mehrere Server, zu Herausforderungen in der Sicherheit. [8]

Die Sicherheit eines Datenbanksystems kann in unterschiedliche Services unterteilt werden: Authentifizierung, Datenintegrität, Vertraulichkeit und Datenbank-Logs. Mit Authentifizierung wird der Zugriff einzelner Nutzer auf die Datenbank kontrolliert. Unter Datenintegrität wird die Korrektheit, Vollständigkeit und Konsistenz der Daten verstanden. Ob Daten wohl sicher sind und nicht unkontrolliert nach außen gelangen, fällt unter Vertraulichkeit. Datenbank-Logs ermöglichen es die Datenbank zu überwachen und verdächtige Tätigkeiten oder Unregelmäßigkeiten zu detektieren.

Eine Übersicht der genannten Security Services speziell für MS SQL Server und Neo4j stellt Tabelle 4.1 dar.

Tabelle 4.1: Sicherheit Services

Kategorie	relationale Datenbanken (MS SQL Server)	Graphdatenbanken (Neo4j)
Authentifizierung	MS SQL Server verwendet das User-Password Schema als Authentifizierung.	Neo4j verwendet SSL Protokolle um Clients zu authentifizieren. Jedoch gibt es keine Methode um die Authentifizierung des Servers zu gewährleisten.
Datenintegrität	MS SQL Server erfüllt die ACID Kriterien und gewährleistet somit Datenintegrität und zuverlässige Transaktionen.	Neo4j erfüllt die ACID Kriterien und gewährleistet somit Datenintegrität und zuverlässige Transaktionen.
Vertraulichkeit	MS SQL Server bietet unterschiedliche Verschlüsselungsoptionen.	In Neo4j werden die Daten klar gespeichert, was die Vertraulichkeit mindert.
Datenbank-Logs	Bei MS SQL Server gibt es den Transactionlog und den Errorlog. Im Transactionlog stehen alle Transaktionen und Datenbankmodifikationen [25]. Im Errorlog stehen benutzerdefinierte Ereignisse und gewisse Systemereignisse [26].	Neo4j hat 5 unterschiedliche Logfiles: neo4j.log für allgemeine Information, bug.log für die Fehlersuche, http.log für Requests der HTTP API, gc.log als Garbage Collection von JVM, query.log für ausgeführte Abfragen.

Quelle: [17], [27]

Zusammenfassend haben relationale Datenbanken ausgereifere Sicherheitsmechanismen als Graphdatenbanken.

4.2 EINFACHHEIT DER PROGRAMMIERUNG UND ABFRAGESPRACHE

Die Einfachheit der Programmierung ist abhängig von der Anwendung.

Das Traversieren von Graphen zum Beispiel ist in Neo4j einfach, da die Neo4j API Methoden dafür bereitstellt. Bei relationalen Datenbanken ist dieser Anwendungsfall komplizierter, da er mehrere Schleifen, Rekursionen oder teure Join-Operatoren beinhalten kann. Hingegen sind Suchabfragen

bei relationalen Datenbanken einfacher, da sie über eine simple WHERE Bedingung erfolgen. Bei Graphdatenbanken hängt die Performance von der Indizierung ab. Neo4j verwendet Lucene, wodurch Abfragen von Text performanter sind als numerische Datentypen und Datumsformate.

Mit relationalen Datenbanken kann man auch Graphdaten speichern, wofür auch eine Graphenoberfläche programmierbar ist. Die umgekehrte Variante ist nicht zwingend möglich. [17]

Structured Query Language (SQL) ist die standardisierte Abfragesprache für relationale Datenbanken. Damit können komplexe Abfragen über standardisierte Schnittstellen durchgeführt werden [8].

Für Graphdatenbanken gibt es keine standardisierte Abfragesprache [8]. Neo4j verwendet Cypher als Abfragesprache.

Im Bezug auf das Wechseln von einer Implementation in eine andere haben relationale Datenbanken aufgrund ihrer einheitlichen Abfragesprache SQL einen klaren Vorteil gegenüber den Graphdatenbanken, wo es mehrere Abfragesprachen und APIs gibt. [16]

4.3 FLEXIBILITÄT

Ein Datenbanksystem gilt als flexibel, wenn nachträgliche Änderungen am Datenmodell möglich sind.

Relationale Datenbanken setzen strukturierte Daten voraus. Das Datenbankschema muss vor dem Hinzufügen der Daten definiert werden und wird somit als statisch angesehen. Falls im Zuge der Entwicklung einer Softwareanwendung dennoch Änderungen am Datenmodell notwendig sind, können große Probleme auftreten. Die Änderungen müssen genau durchdacht werden, da Servicefehler, verschlechterte Performance und große Anpassungen in der übrigen Anwendung die Folge sein können [8].

Not only SQL (NoSQL) Datenbanken und somit auch Graphdatenbanken sind hingegen für strukturierte, halb-strukturierte und unstrukturierte Daten geeignet. Das heißt, dass das Datenbankschema dynamisch ist und somit nicht im Vorhinein definiert werden muss. Datentyp- und Datenstrukturänderungen können ohne Probleme im Nachhinein vollzogen werden. Daher werden NoSQL Datenbanken gerne für agile und skalierbare Projekte, welche sich ständig entwickeln, verwendet [8].

4.4 PERFORMANCE VON NETZWERKBERECHNUNGEN

Dieses Kriterium ist aufgrund der vielen Abhängigkeiten wie Arbeitsspeicher, Rechenleistung, Größe des Datensatzes sowie Unsicherheiten in der Implementierung der Netzwerkberechnung schwer zu verallgemeinern.

[18] beschäftigt sich mit der Performance des Shortest Path Algorithmus und kommt zu dem Entschluss, dass Graphdatenbanken für Routingabfragen, welche den kompletten Graph betreffen, aufgrund des großen Speicherbedarfs nicht geeignet sind. Die Stärke von Graphdatenbanken liegt eher auf kürzeren Graphtraversierungen, wie man sie bei sozialen Netzwerken findet. Jedoch ist die Performance von Neo4j im Vergleich zum relationalen pgRouting in den meisten Fällen besser.

[19] erwähnt, dass die Performance von Neo4j stark von der Art des Aufrufs abhängig ist. Es macht einen Unterschied ob man die Neo4j Datenbank über eine Java Anwendung oder über die Cypher Abfrage Plattform anspricht.

[17] gelangt zu dem Ergebnis, dass Neo4j vor allem bei großen Datenmengen schneller als die relationalen Datenbank MySQL ist und das auch bei Graphtraversierungen.

4.5 SKALIERBARKEIT

Die Skalierbarkeit bezieht sich auf die Fähigkeit einer Datenbank, ihre Leistung und Kapazität zu erhöhen, um eine größere Anzahl von Benutzern und Abfragen zu bewältigen, ohne dass die Leistung und Verfügbarkeit beeinträchtigt werden. Sie ist somit für die Auswahl des Datenbanksystems ein wichtiges Kriterium.

Relationale Datenbanken bauen auf einer vertikalen Skalierbarkeit auf, was bedeutet, dass bei einem Zuwachs der Daten der Server aufgestockt werden muss. Das heißt, die Kapazität von CPU, RAM und Festspeicher muss erhöht werden, um die Rechenleistung zu erhöhen, was Kosten mit sich bringt. Die vertikale Skalierung wurde für relationale Datenbanken gewählt, um eine gleichzeitige Nutzung zu ermöglichen.

NoSQL Datenbanken und somit auch Graphdatenbanken verwenden eine horizontale Skalierbarkeit. Das gibt einen großen Vorteil bei wachsenden Datenmengen, da für zusätzlichen Speicher und Rechenleistung einfach zusätzliche Server ergänzt werden können, was vergleichsweise günstiger ist. [8], [15]

Abbildung 4.1 zeigt das Prinzip der vertikalen und horizontalen Skalierung grafisch veranschaulicht.

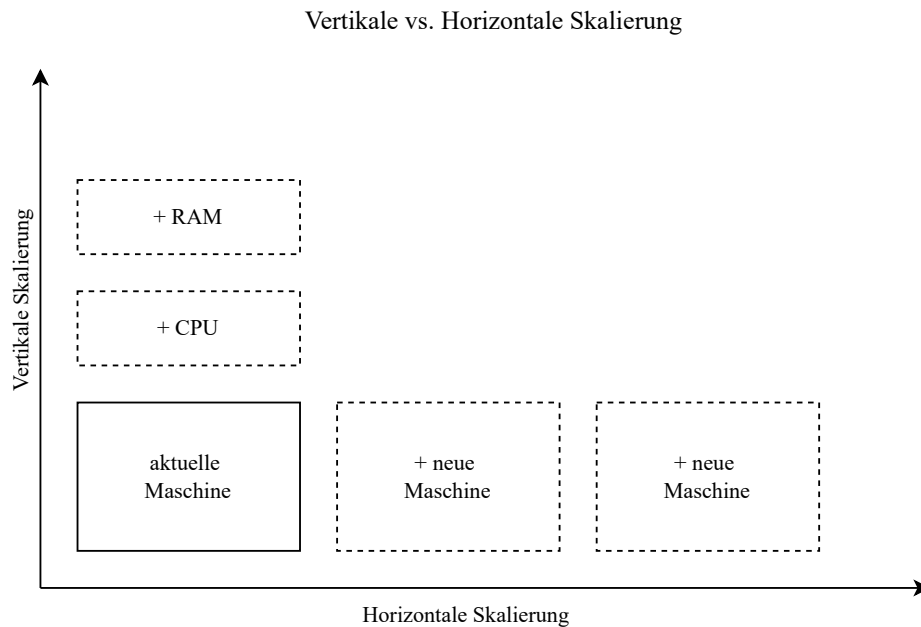


Abbildung 4.1: Vertikale vs. Horizontale Skalierung

4.6 AUSGEREIFTHEIT UND SUPPORT

Ausgereiftheit kann mit dem Alter eines Systems und der Anzahl der Nutzer in Verbindung gesetzt werden, denn je mehr ein System verwendet wird, desto mehr wird es getestet und desto mehr Bugs werden gefunden. Daraus entstehen stabile Systeme. Auch der Support steigt in der Regel mit der Ausgereiftheit eines Systems. Denn auch hier gilt, je öfter ein System verwendet wird, desto öfter wird darüber in unterschiedlichen Bereichen wie Industrie oder Forschung gesprochen. [16]

Relationale Datenbanken werden schon lange und auch von vielen unterschiedlichen Branchen verwendet. Dementsprechend sind diese sehr ausgereift, stabil und bieten viel Support. Die große Nutzeranzahl motiviert auch die größten Anbieter wie Oracle oder Microsoft finanziell, weiterhin den Standard von Performance und Support hoch zu halten. Auch die einheitliche Abfragesprache SQL begünstigt die Menge an Support, da so der Support für eine Implementation meist ohne Probleme auch für andere Implementation gültig ist. [16], [17]

Graphdatenbanken hingegen gibt es vergleichsweise noch weniger lang. Auch die Anwenderzahl ist geringer, was die finanzielle Motivation der Anbieter reduziert. Dazu kommt auch, dass unterschiedliche Graphdatenbank Implementationen unterschiedliche Abfragesprachen verwenden, was den Support für Graphdatenbanken verringert, da der Support möglicherweise nicht für alle Implementationen anwendbar ist. Neo4j bietet aber soliden Support auf ihrer Webseite. Jedoch würde ein großer Teil des Supports wegfallen, wenn das Unternehmen dahinter zusammenbrechen würde. Es ist allerdings davon auszugehen, dass die Beliebtheit von Graphdatenbanken weiter wachsen wird und somit auch der Support sowie die Ausgereiftheit weiter steigen wird. [16], [17]

KAPITEL 5

IMPLEMENTIERUNG

In diesem Kapitel wird zuerst näher auf die Definition des Datenmodells eingegangen. Danach wird die für die weitere Implementierung verwendete Entwicklungsumgebung angeführt. Im Anschluss werden die einzelnen Schritte der Überführung der gegebenen Daten von einer relationalen Datenbank in eine Graphdatenbank beschrieben. Darauf aufbauend werden dann die geforderten Netzwerkberechnungen veranschaulicht.

5.1 DATENMODELLDEFINITION

Für die Datenmodelldefinition werden zuerst die Netzkomponenten des Niederspannungsnetzes vorgestellt. Daraufhin wird das entwickelte Datenmodell präsentiert.

5.1.1 Netzkomponenten

Im folgenden werden die wichtigsten Netzkomponenten des Niederspannungsstromnetzes der Energienetze Steiermark GmbH (EN), welche jeweils in eigenen Tabellen in einer relationalen Datenbanken gespeichert sind, näher beschrieben.

Station

Jedes Ortsnetz hat eine Station, welche aus einem oder mehreren Transformatoren besteht. Dies ist zwar nicht netzwerktechnisch abgebildet, dennoch kommt hierher der Begriff Transformatorstation. Die Station hat eine eindeutige Bezeichnung, den Technischen Platz, und transformiert die elektrische Energie vom Mittelspannungsnetz ins Niederspannungsnetz. In Österreich gibt es im Dezember 2021 81.126 Transformatorstationen [28].

Abzweig

Aus einer Station gehen mehrere Abzweige ab. Abzweige gibt es nicht in baulicher Form, sind aber netzwerktechnisch relevant. Für einen Netzbetreiber ist es interessant zu wissen, was alles an welchem Abzweig hängt.

Kasten

Ein Kasten, Kabelverteilschrank oder Schaltschrank bildet einen Knotenpunkt ab. Im Störfall können so über Sicherungen fehlerhafte Strecken abgeschaltet werden. Baulich befinden sich Kästen oft an Straßenkreuzungen. [14, S. 465] Die Bezeichnungen F1-F6 und Ähnliches bezieht sich dabei auf die Bauart.

Stützpunkt

Zu den Stützpunkten zählen die unterschiedlichen Masten. Sie sind netzwerktechnisch nicht relevant und daher auch nicht im Datenmodell abgebildet.

Endpunkt

Der Endpunkt stellt das Ende einer Leitung dar. Dies ist in den meisten Fällen ein Hausanschluss, kann aber auch beispielsweise vom Typ Pumpwerk, Messschrank, E-Ladesäule, Hochbehälter, Baustrom oder ein sonstiger Endpunkt sein.

Muffe

Muffen sind nur baulich relevant und kommen vor, wenn zwei Leitungen miteinander verbunden werden. Sie haben netzwerktechnisch keine Funktion und werden daher auch nicht im Datenmodell berücksichtigt.

Trennstelle

Durch das Schalten von Trennstellen kann ein Netz über andere Wege versorgt werden. Sie befindet sich baulich meistens in einem Kasten. Trennstellen werden im Datenmodell dieser Arbeit allerdings nicht behandelt, könnten aber beispielsweise über ein Attribut im Kasten oder auf der Leitung modelliert werden.

Leitung

Bei den Verbindungen zwischen den Nodes spricht man im Stromkontext von Leitungen. Dabei wird unter Freileitungen (überirdisch) und Kabel (unterirdisch) unterschieden.

5.1.2 Datenmodell

Abbildung 5.1 zeigt das entwickelte Datenmodell für die Graphdatenbank.

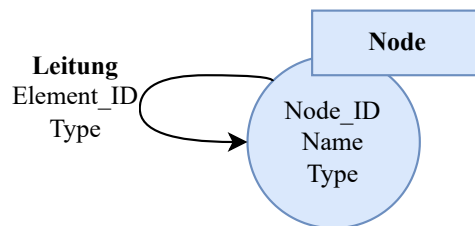


Abbildung 5.1: Datenmodell für die Graphdatenbank Neo4j

Das Datenmodell ist so simpel wie nötig für die gewünschten Anwendungsbeispiele, auf welche im Kapitel 5.4 näher eingegangen wird, gehalten. Alle Nodes werden mit Label Node in die Graphdatenbank geladen und mit der Relation Leitung verbunden. Die Klassifizierung der zuvor beschriebenen Netzwerkkomponenten geschieht über das Attribut Type im Node. Der Type der Leitung gibt an, ob es sich um ein Kabel oder um eine Freileitung handelt. Der Name bei den Nodes kann für Visualisierungszwecke verwendet werden.

Vergleich zum relationalen Datenbankmodell

Im relationalen Datenbankmodell wird strikt nach den einzelnen Netzwerkkomponenten unterschieden, das heißt, für jede Netzwerkkomponente gibt es eine eigene Tabelle in der Datenbank. Der Grund dafür ist die große Anzahl an Metadaten, welche es zu den unterschiedlichen Netzwerkkomponenten gibt. Diese werden für diverse Anwendungen benötigt, jedoch nicht für topologische Abfragen, auf welchen der Fokus in dieser Arbeit liegt. Daher ist das Datenmodell der Graphdatenbank nur aufs wesentliche beschränkt und somit stark vereinfacht.

Anmerkungen zum Datenmodell

Die sogenannten Relations der Graphdatenbank verlangen eine Richtung. Bei Wechselspannung gibt es allerdings keine Richtung, da sich die Stromrichtung in regelmäßigen Abständen ändert. Da die gegebenen Ortsnetze topologisch ein Strahlennetz darstellen, wird die Richtung im erstellten Datenmodell von der Station zum Endverbraucher definiert.

Für den Fall von Maschen- oder Ringnetzen müsste man das Datenmodell überdenken und dementsprechend anpassen. Theoretisch könnte jede Relation doppelt, also in beide Richtungen, modelliert werden, was jedoch auch ein Anpassen der Abfragen implizieren würde.

Beispiel für Datenmodell

Abbildung 5.2 zeigt zur Veranschaulichung ein fiktives Ortsnetz. Dabei ist zu erkennen, welche unterschiedlichen Node Typen es gibt und welche Eigenschaften diese besitzen. Zudem ist erkennbar, welche Nodes in der Praxis wie verbunden werden können.

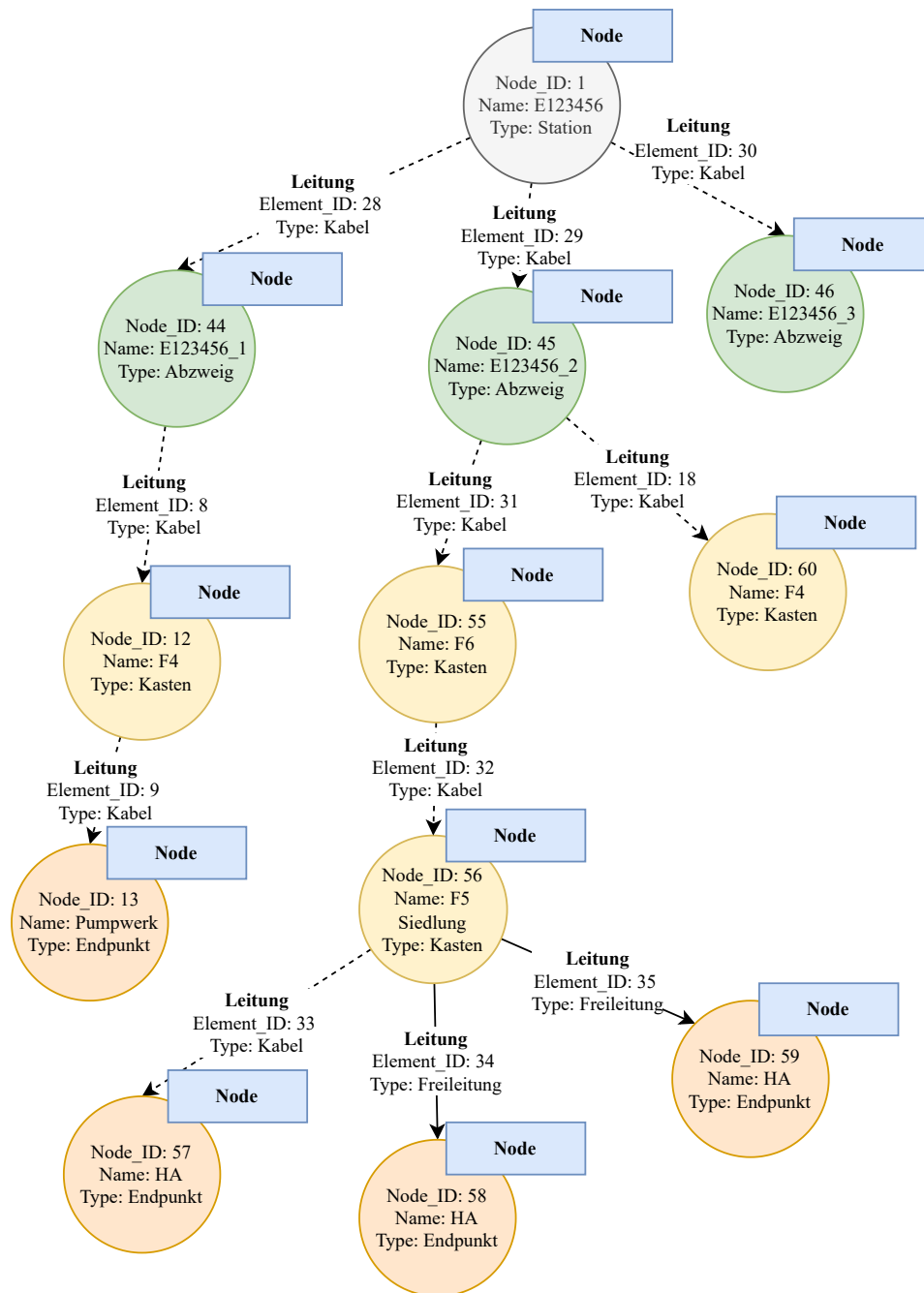


Abbildung 5.2: Angewendetes Datenmodell

5.2 ENTWICKLUNGSUMGEBUNG

Der Umsetzungsteil wurde auf einem Apple MacBook Air (2022) durchgeführt. Die folgende Tabelle beinhaltet die detaillierte Auflistung der verwendeten Hardware.

Tabelle 5.1: Verwendete Hardware

Prozessor	Apple M2 Chip mit 8-Core CPU, 8-Core GPU und 16-Core Neural Engine
Arbeitsspeicher	8 GB gemeinsamer Arbeitsspeicher

Die verwendete Software mit Version ist in folgender Tabelle aufgelistet.

Tabelle 5.2: Verwendete Software mit Version

Software	Version	Open Source
MacOS Ventura	13.1	Nein
Neo4j Desktop	1.5.2	Ja
Neo4j enterprise Edition	4.4.5	Ja
DB Browser for SQLite	3.12.2	Ja

5.3 ÜBERFÜHRUNG DER DATEN VON EINER RELATIONALEN DATENBANK IN EINE GRAPHDATENBANK

Der grobe Ablauf der Überführung der Daten von Simple Features aus der relationalen MS SQL Server Datenbank in die Graphdatenbank Neo4j ist in Abbildung 5.3 dargestellt. Die einzelnen Schritte werden in den darauffolgenden Unterkapiteln näher beschrieben.

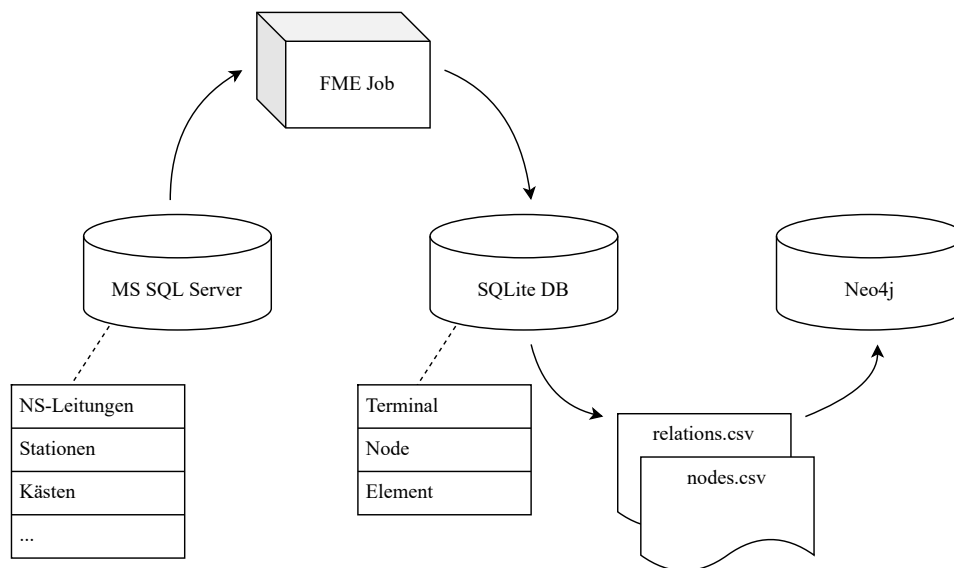


Abbildung 5.3: Umsetzungsablauf

5.3.1 MS SQL Server

Die ursprünglichen Ausgangsdaten sind die Leitungsdaten in Form von OGC Simple Features in einer MS SQL Server Datenbank. Diese Daten werden im GIS dargestellt und werden für Leitungsbeauskunftungsabfragen verwendet. Für Netzberechnungen sind vor allem die Tabellen zu NS-Leitung, Station, Abzweig, Kasten und Endpunkt interessant.

5.3.2 FME Job

Da bei OGC Simple Features keine topologische Information vorhanden ist, werden die Daten mit Hilfe von FME Jobs aufbereitet, um ein Netzwerk in Knoten-Kanten-Struktur zu erhalten. Dieser Prozess ist nicht zu verallgemeinern und stark an die Daten der EN gebunden, da er jede Menge Abfragen beinhaltet, welche die Besonderheiten der vorhandenen Daten verarbeitet. Aus diesem Grund wird hier nicht bis ins Detail auf diesen Teil eingegangen.

Das Input des FME Jobs stellen die OGC Simple Features der MS SQL Server Datenbank dar. Als Output des FME Jobs resultieren die 3 Tabellen Node, Element und Terminal in einer SQLite Datenbank, welche ein topologisch sauberes Netzwerk abbilden.

5.3.3 SQLite Datenbank

Abbildung 5.4 zeigt das Entity-Relationship-Diagramm der Tabellen in der SQLite Datenbank.

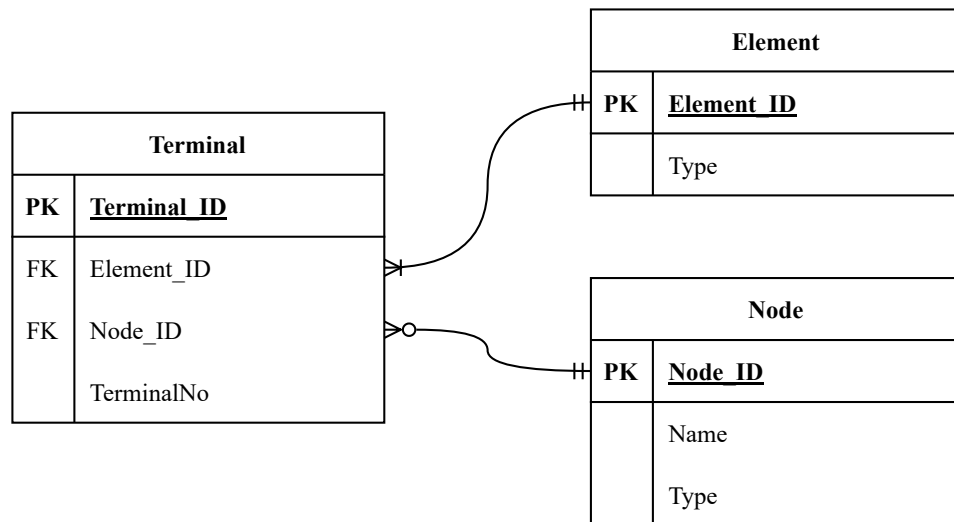


Abbildung 5.4: ER-Diagramm der SQLite Tabellen

In Node sind alle Nodes des Netzwerkes mit einer fixen Node_ID gespeichert. Zusätzlich sind die Attribute Name und Type vorhanden. Name stellt dabei eine Bezeichnung des Nodes dar und Type den Typ des Nodes (Station, Abzweig, Kasten oder Endpunkt).

In Element sind alle Relations mit einer Element_ID vorhanden. In dieser Tabelle gibt es das Attribut Type. In diesem Fall stellt Type den Typ der Leitung dar. Das kann Freileitung oder Kabel sein.

In Terminal ist gespeichert, mit welcher Relation die Nodes verbunden sind. Dies wird über die Felder Element_ID, Node_ID und TerminalNo umgesetzt. TerminalNo kann die Werte 1 oder 2 annehmen und sagt aus, ob es sich bei der jeweiligen Node_ID um einen Start- oder Endnode handelt. Bei Terminal=1 ist der Node mit der Node_ID der Startnode zum Element mit Element_ID. Bei TerminalNo=2 handelt es sich bei der Node_ID um den Endnode des Elements. Das heißt, für eine Relation zwischen 2 Nodes sind 2 Terminal Einträge vorhanden.

5.3.4 CSV Tabellen

In der Graphdatenbank Neo4j werden Nodes und Relations gespeichert. Dafür wurden die 3 SQLite Tabellen noch einmal zusätzlich im DBBrowser mit SQL-Abfragen aufbereitet und als *nodes.csv* und *relations.csv* Files exportiert.

Der Codeausschnitt 5.1 zeigt die Extrahierung der Nodes. Dabei wurde die bereits vorhandene Node Tabelle, welche wie zuvor beschrieben alle Nodes zwischen Leitungen beinhaltet, auf die benötigten Felder beschränkt.

```
1 SELECT Node_ID, Name, Type
2 FROM Node
```

Listing 5.1: Node.csv extrahieren

Der Codeausschnitt 5.2 zeigt die Extrahierung der Relations. Aufgrund des davor beschriebenen Aufbaus der Terminal Tabelle wurde hier eine Vereinfachung der Tabelle durchgeführt. Die entstehende *relation.csv* beinhaltet pro Relation nur noch einen Zeileneintrag. Hierbei wurde darauf geachtet, dass nur die Terminaleinträge zu den Leitungen ausgewählt werden. Dafür wurde nach der Element_ID gruppiert und nur die Zeilen mit einer Element_ID, wovon es genau 2 gibt, herangezogen. Für andere Anwendungen, die hier nicht relevant sind, befinden sich in der Tabelle nämlich auch Einträge, wo die Element_ID genau einmal vorkommt. Durch die Gruppierung wurden diese hier ausgeschlossen. Dann wurde die Node_ID mit TerminalNo=1 als Startnode also in der exportierten CSV Tabelle als Node_ID1 verwendet. TerminalNo=2 liefert den Endnode, also Node_ID2.

```
1 SELECT n1.element_id AS Element_ID,
2       n1.node_id AS Node_ID1,
3       n2.node_id AS Node_ID2,
4       e.type
5 FROM
6   (SELECT a.element_id,
7          a.node_id,
8          a.terminalno
9   FROM terminal a
10  JOIN
11   (SELECT element_id,
12          Count(*)
13   FROM terminal GROUP BY element_id
14   HAVING Count(*) = 2) b
15  ON a.element_id = b.element_id
16  WHERE terminalno = 1 ) n1
```

```

17 JOIN
18   (SELECT a.element_id,
19         a.node_id,
20         a.terminalno
21   FROM terminal a
22   JOIN
23     (SELECT element_id,
24           Count(*)
25     FROM terminal GROUP BY element_id
26     HAVING Count(*) = 2) b
27   ON a.element_id = b.element_id
28   WHERE terminalno = 2 ) n2
29 ON n1.element_id = n2.element_id
30 JOIN element e
31 ON n1.element_id = e.element_id

```

Listing 5.2: Relation.csv extrahieren

Der Inhalt der entstandenen CSV Tabellen wird in der Abbildung 5.5 dargestellt.

Relations	Nodes
Element_ID	Node_ID
Node_ID1	Name
Node_ID2	Type
Type	

Abbildung 5.5: CSV Tabellen

5.3.5 Neo4j

Die beiden entstandenen CSV Tabellen wurden dann in Neo4j importiert. Dazu wurden die Tabellen in den von Neo4j vorgegebenen Import Ordner gelegt. In der Neo4j Browser Oberfläche wurden die Tabellen dann mittels den Code Ausschnitten 5.3 und 5.5 importiert.

```

1  LOAD CSV WITH HEADERS FROM "file:///nodes.csv" AS node
2  CREATE (n:Node)
3  SET n = node

```

Listing 5.3: Node.csv in Neo4j importieren

Um die Performance zu erhöhen, wurde für die Nodes vor dem Laden der Relations noch ein Index mit dem Codeausschnitt 5.4 generiert [29].

```
1 CREATE INDEX FOR (n:Node) ON (n.Node_ID)
```

Listing 5.4: Create Index

Der nächste Codeausschnitt zeigt das Importieren der Relations. Dafür werden anhand der Node_ID1 und Node_ID2 die bereits vorhandenen Nodes gematcht.

```
1 LOAD CSV WITH HEADERS FROM "file:///relations.csv" AS
  relation
2 MATCH (node1:Node {Node_ID: relation.Node_ID1}), (node2:Node
  {Node_ID: relation.Node_ID2})
3 CREATE (node1) - [:Leitung {Element_ID: relation.Element_ID,
  Type: relation.Type}] -> (node2)
```

Listing 5.5: Relation.csv in Neo4j importieren

5.4 NETZWERKBERECHNUNGEN

Im folgenden werden Anwendungsbeispiele aus dem EVU Bereich beschrieben. Dazu wird zuerst kurz die Anwendung erklärt und im Anschluss die dazu notwendige Abfrage mit Ergebnis dargestellt.

5.4.1 Abzweigeinfärbung

Abzweigeinfärbungen sind für EVU interessant, da es wichtig ist zu wissen, welche Kästen, Endpunkte oder sonstige Punkte an welchem Abzweig hängen.

In Neo4j wurde dafür den darunterliegenden Nodes eines Abzweiges ein neuer Label gesetzt. Dieser Schritt wurde mit der Abfrage aus dem Codeausschnitt 5.6 durchgeführt. Wichtig dabei ist, dass die Node_ID des Abzweiges bekannt sein muss. Anzumerken ist auch, dass diese Abfrage voraussetzt, dass alle Relations vom Abzweig wegzeigen müssen.

```
1 MATCH (n:Node)<-[:Leitung*]-(abzweig:Node {Node_ID:"13"})
2 SET n:Abzweig13
3 RETURN n, abzweig
```

Listing 5.6: Abzweigeinfärbung

Abbildung 5.6 zeigt die Abzweigeinfärbung anhand eines Ortsnetzes. Dabei ist in der Mitte des Graphs die Station mit den direkt benachbarten Abzweigen sichtbar. Die restlichen Nodes haben je nach Abzweig ein eigenes Label (siehe Legende auf der rechten Seite) und sind somit je nach Abzweig unterschiedlich farblich dargestellt.

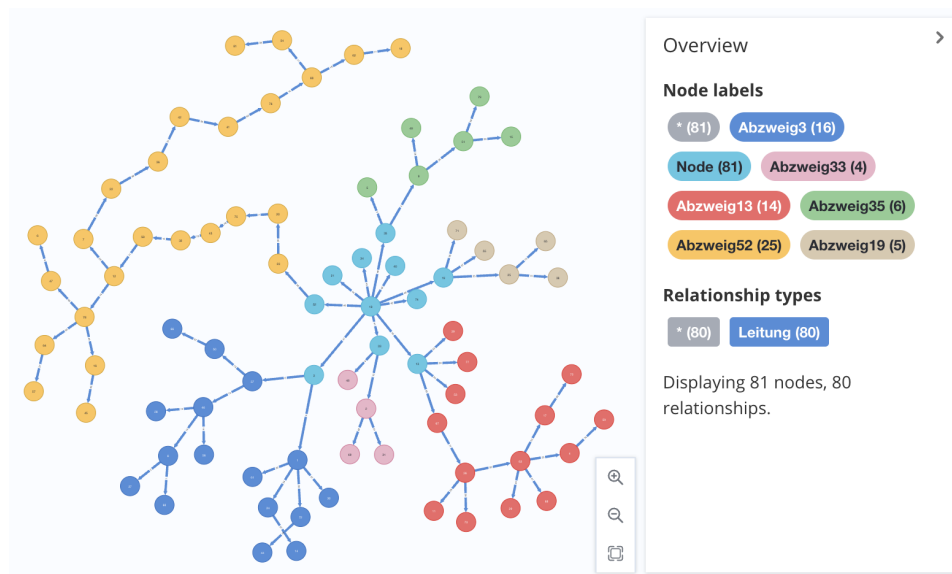


Abbildung 5.6: Abzweigeinfärbung

5.4.2 Ausfall einer Leitung

Auch das Wissen von nicht verbundenen Nodes zu einer Station im Ortsnetz, zum Beispiel im Falle einer kaputten Leitung, ist für EVU wichtig.

In Neo4j wurde dieser Anwendungsfall mit folgenden Abfragen nachgestellt. Zuerst wurde eine Leitung über die Element_ID mit dem Code Abschnitt 5.7 entfernt.

```
1 MATCH ()-[r:Leitung{Element_ID:"71"}]->()
2 DELETE r
```

Listing 5.7: Relation löschen

Das Ergebnis ist in Abbildung 5.7 dargestellt. Auf der linken Seite befinden sich die von der Station abgetrennten Nodes.

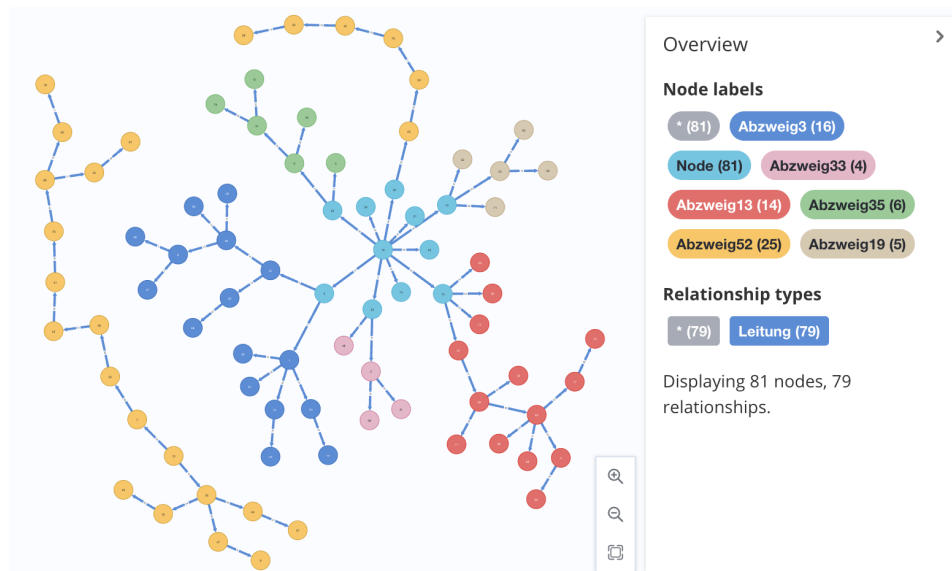


Abbildung 5.7: Relation gelöscht

Mit dem folgenden Codeausschnitt werden dann die Nodes, welche nicht mit der Station verbunden sind, zurückgegeben. Dafür werden zuerst alle Nodes, welche mit der Station verbunden sind in `connected` gespeichert und anschließend alle Nodes ausgewählt, welche nicht in `connected` sind, zurückgegeben. Zeile 1 wendet hierbei das gleiche Prinzip wie die Abzweigeinfärbung im Codeausschnitt 5.6 an. Die Node_ID ist hierbei die der Station.

```

1 MATCH (n:Node)<-[:Leitung*]-(station:Node{Node_ID:"10"})
2 WITH (COLLECT(n) + station) AS connected
3 MATCH (c:Node)
4 WHERE NOT c IN connected
5 RETURN c

```

Listing 5.8: Nodes, die nicht mit der Station verbunden sind

Das Ergebnis ist in der folgenden Abbildung dargestellt.

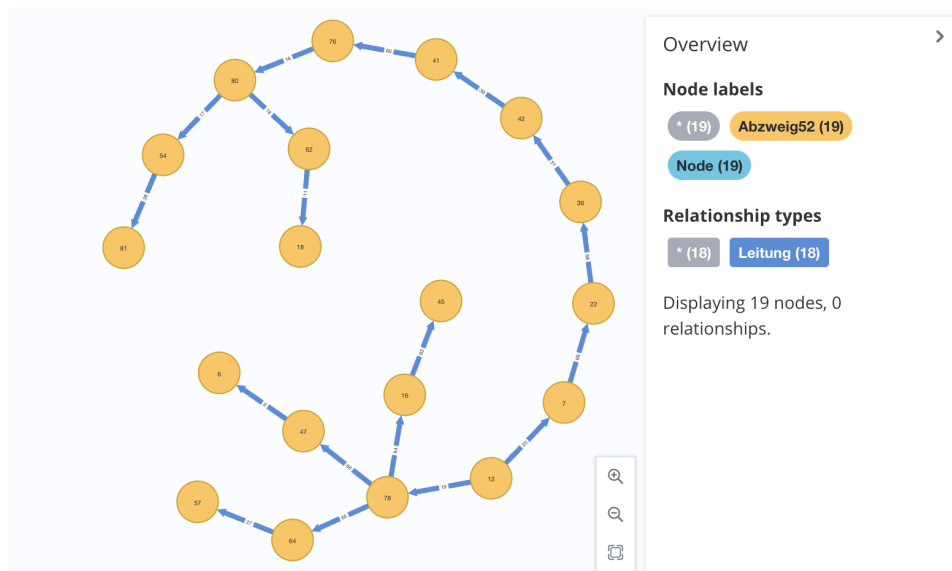


Abbildung 5.8: Nodes, die nicht mit der Station verbunden sind

5.4.3 Netzverfolgung

Mit Netzverfolgung ist die Route gemeint, wie ein bestimmter Node mit der Station verbunden ist. Dazu zeigt 5.9 die Ausgangslage für dieses Anwendungsbeispiel. Der ausgewählte Node stellt den bestimmten Node dar, die Station ist mit rotem Pfeil markiert.

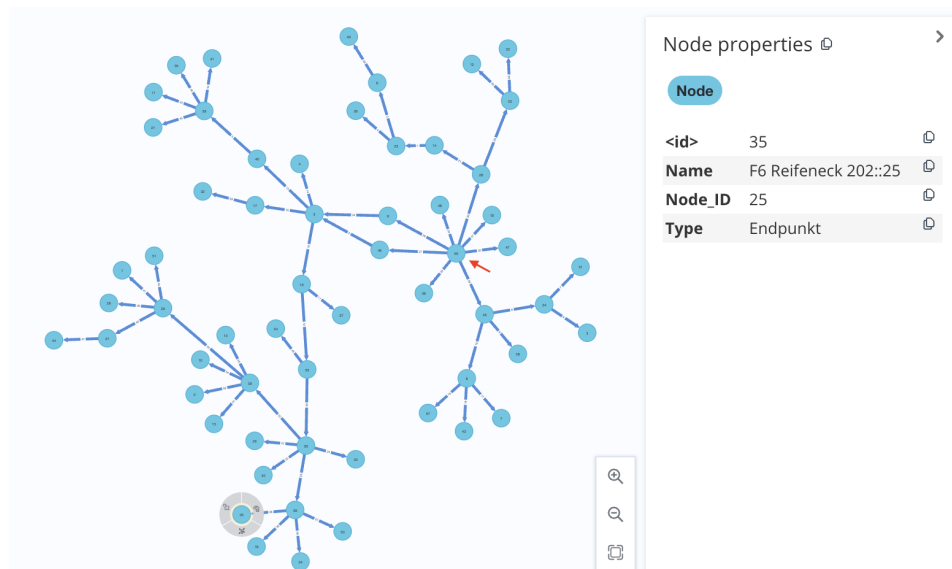


Abbildung 5.9: Netzverfolgung Ausgangslage

Im Folgenden wird die Abfrage zur Netzverfolgung gezeigt. Dabei wird der Node, zu welchem die Netzverfolgung gemacht wird, als **querynode** definiert. Über den restlichen Teil werden alle Nodes, von welchen eine Relation wegzeigt, in **n** gespeichert. Das ***** nach **Leitung** bedeutet, dass wirklich alle davor liegenden Nodes einbezogen werden. Dadurch, dass die Station der Node ist, von dem alle Relationen weggehen, ist sie hiermit der letzte Node.

```

1 MATCH (querynode:Node {Node_ID:"25"})<-[r:Leitung*]-(n:Node)
2 RETURN querynode, n

```

Listing 5.9: Netzverfolgung-Abfrage zum Node mit Node_ID = 25

Das Ergebnis des Codeausschnittes ist in Abbildung 5.10 dargestellt.

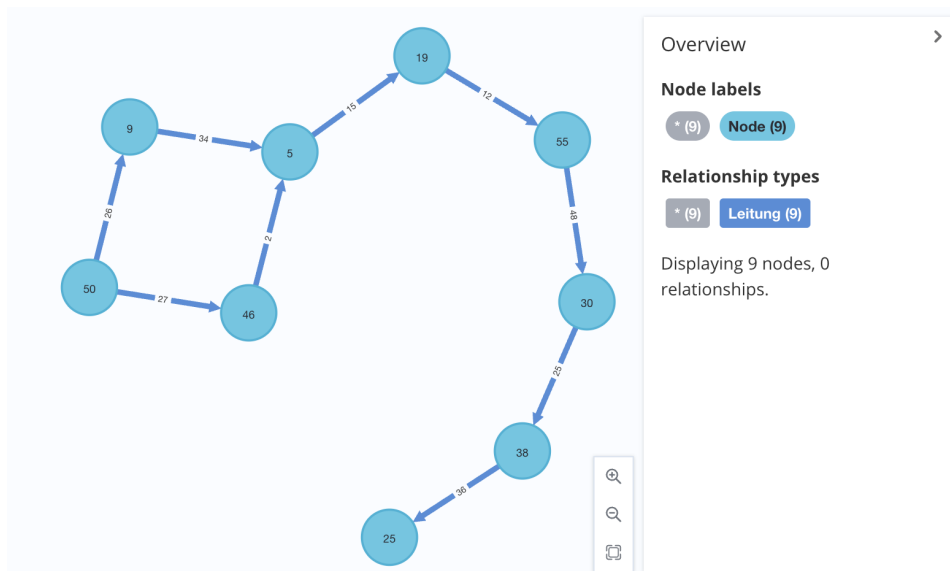


Abbildung 5.10: Netzverfolgung Ergebnis

5.4.4 Mehrere Ortsnetze

Zum Schluss wurden noch mehrere Ortsnetze in Neo4j geladen, um zu zeigen, wie der Umgang mit größeren Datenmengen ausschauen würde. Die in diesem Beispiel gewählten 25 Ortsnetze beinhalten 1789 Nodes und 1783 Relations. Die Ortsnetze sind nicht miteinander verbunden. Abbildung 5.11 zeigt die 25 Ortsnetze in der Neo4j Oberfläche, wo aufgrund der Menge an Daten keine Details mehr sichtbar sind.

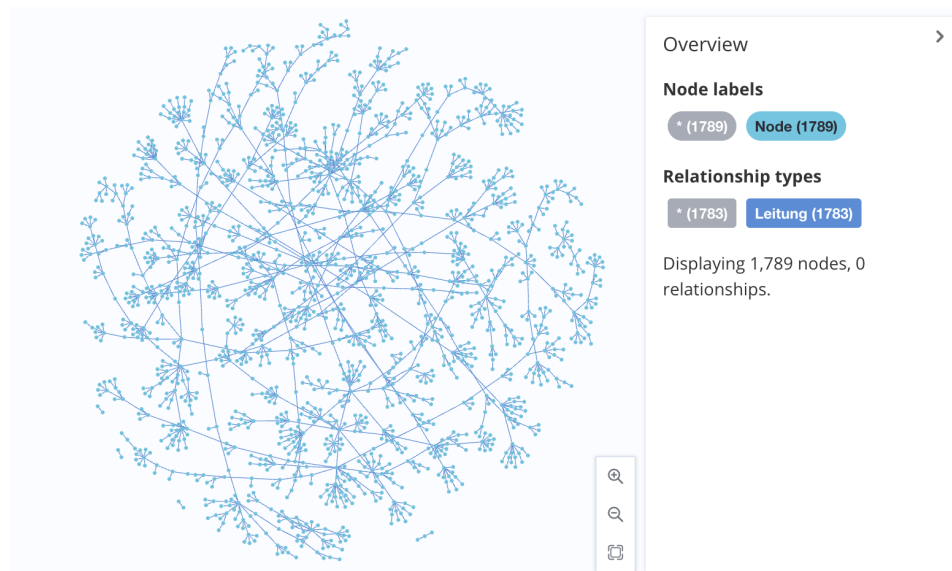


Abbildung 5.11: 25 Ortsnetze

Daher wurde in der Abbildung 5.12 der selbe Inhalt gezoomt dargestellt.

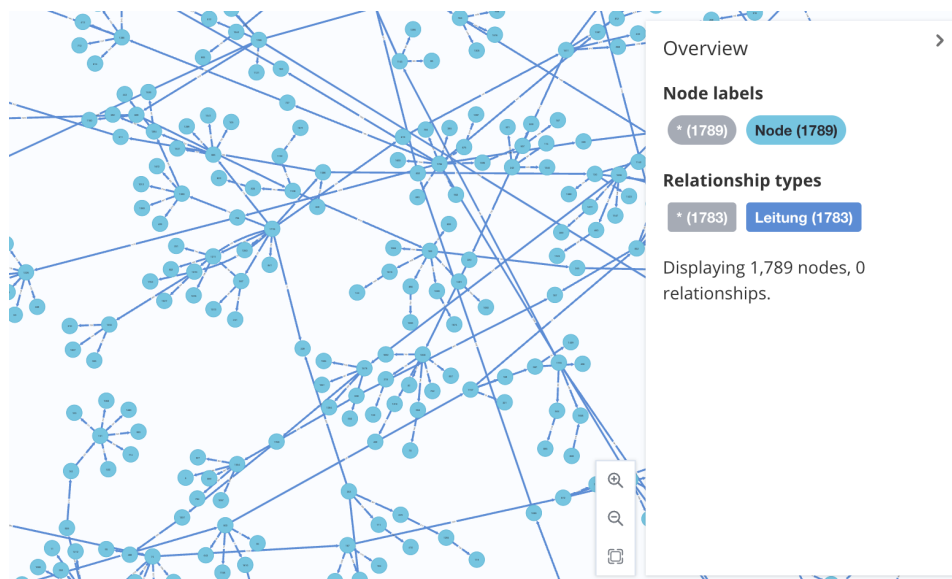


Abbildung 5.12: 25 Ortsnetze gezoomt

Da die beschriebenen Anwendungsfälle sich immer nur auf ein Ortsnetz beziehen, wird im Codeausschnitt 5.10 gezeigt, wie man bezogen auf die Node.ID der Station ein Ortsnetz auswählen kann. Das funktioniert wieder aufgrund des Datenmodells und der Eigenschaft, dass die Richtung aller

Relations von der Station weggerichtet ist. Wären also von der Station abgetrennte Nodes dabei, würden diese verloren gehen und bei dieser Abfrage nicht zurückgegeben werden. Das unterstreicht die Wichtigkeit einer sauberen Topologie.

```
1 MATCH (n:Node)<-[:Leitung*]-(station:Node {Node_ID:"1775"})
2 RETURN n, station
```

Listing 5.10: 1 Ortsnetz auswählen

Das Ergebnis zu diesem Beispiel in Neo4j wird in Abbildung 5.13 dargestellt.

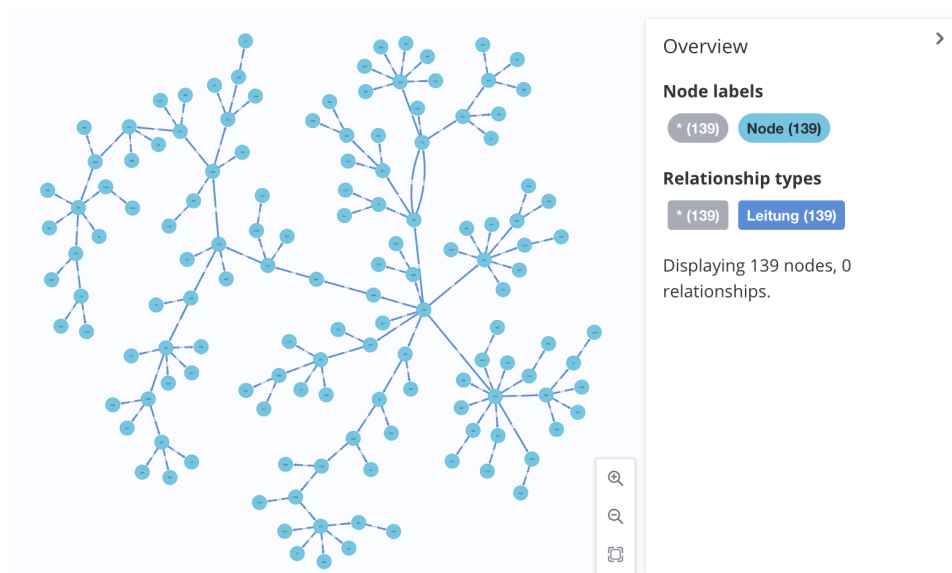


Abbildung 5.13: 1 Ortsnetz

KAPITEL 6

ERGEBNISSE UND FAZIT

Dieses Kapitel zeigt zuerst einen Überblick des qualitativen Vergleichs von relationalen Datenbanken und Graphdatenbanken, welcher dann für den Anwendungsfall von Energieversorgungsunternehmen (EVU) beurteilt wird, sowie die Ergebnisse der Implementierung. Die Ergebnisse werden im Anschluss diskutiert. Abschließend werden die wissenschaftlichen Arbeitsergebnisse zusammengefasst und ein Ausblick gegeben.

6.1 ÜBERBLICK DER QUALITATIVEN EVALUIERUNG

Tabelle 6.1 stellt basierend auf Kapitel 4 einen Überblick der qualitativen Evaluierung dar. Es werden dabei die Vor- und Nachteile von relationalen Datenbanken gegenüber Graphdatenbanken aufgezeigt. Diese werden dann basierend auf den im Kapitel 3.3 beschriebenen Anforderungen von EVU nach der im Kapitel 3.1 beschriebenen Methodik beurteilt.

Tabelle 6.1: Überblick der qualitativen Evaluierung

Kriterium	relationale Datenbank	Graphdatenbank
Sicherheit	ausgereifte Mechanismen; Dennoch Gefahren wie SQL-Injection, cross-site scripting, rootkits, schwache Kommunikationsprotokolle; erfüllt ACID Kriterien; Gibt Verschlüsselungsoptionen für Daten	Herausforderungen aufgrund von Sharding; Neo4j erfüllt ACID Kriterien; Daten werden klar gespeichert
Einfachheit der Programmierung und Abfragesprache	schlecht bei Traversieren von Graphen; gut bei Suchabfragen; SQL ist eine standardisierte Abfragesprache	gut bei Traversieren von Graphen; Performance von Suchabfragen hängt von Indizierung ab; Die meisten Graphdatenbank Implementierungen haben eine eigene Abfragesprache
Flexibilität	Datenmodell statisch; Datentyp und -struktur im Nachhinein nicht änderbar	Datenmodell dynamisch; Datentyp und -struktur im Nachhinein änderbar
Performance von Netzberechnungen	langsamer bei Traversierung von Graphen	schneller bei Traversierung von Graphen
Skalierbarkeit	vertikale Skalierung; Aufstockung teuer	horizontale Skalierung; Aufstockung günstiger
Ausgereiftheit und Support	gibt es schon lange + große Nutzeranzahl + gleiche Abfragesprache → ausgereift und stabil, viel Support	gibt es weniger lang + geringere Nutzerzahl + unterschiedliche Abfragesprachen → weniger Support

6.2 DISKUSSION

Als Diskussion werden zunächst die Ergebnisse zusammengefasst und interpretiert, bevor auf die Beschränkungen der Forschung eingegangen wird.

Zusammenfassung der Ergebnisse

Das Ergebnis der qualitativen Evaluierung ist wie erwartet nicht eindeutig. Sowohl relationale Datenbanken als auch Graphdatenbanken haben ihre Vor- und Nachteile, welche wiederum stark vom Anwendungsfall abhängen. Selbst innerhalb eines EVU variieren die Anwendungen so stark, dass nicht für alle Anwendungen ein System das Bessere ist. Speziell für Netzwerkberechnungen und topologische Abfragen gibt es mehr Vorteile bei Graphdatenbanken als bei relationalen Datenbanken. Auch die mangelnde Performance von relationalen Datenbanken bei topologischen Abfragen spricht für die Verwendung von Graphdatenbanken. Da sich das Datenmodell durch unterschiedliche Entwicklungen immer wieder ändert, holen sich Graphdatenbanken auch beim Punkt Flexibilität große Pluspunkte. Hingegen haben die ausgereiften Sicherheitsmechanismen von relationalen Datenbanken eine hohe Priorität für EVU. Da die Skalierbarkeit der Datenbanksysteme für EVU im regionalen Bereich aufgrund der überschaubaren Datenmenge für die derzeitigen Anwendungen eher nebensächlich ist, liegen bei diesem Kriterium die Systeme gleich auf. Auch hinsichtlich Einfachheit der Programmierung und Abfragesprache gibt es keine signifikanten Unterschiede. Allerdings liegen die relationalen Datenbanken im Punkt Ausgereiftheit und Support vor den Graphdatenbanken.

Die Literatur ist sich im Großen und Ganzen einig und kommt zu ähnlichen Erkenntnissen. Hier gibt es vor allem viele Vergleiche zwischen SQL und NoSQL Datenbanken. Da der Begriff NoSQL Datenbanken unterschiedliche Systeme (Key-Value Stores, Column Family Systeme, Document Stores, Graphdatenbanken) zusammenfasst, sind die Vergleiche jedoch nicht sehr konkret.

Im Implementierungsteil der Arbeit wurde zu Beginn ein Datenmodell, mit welchem die angestrebten Anwendungsbeispiele umsetzbar sind, vorgestellt. Darauf aufbauend wurde der Weg der Daten aus der relationalen MS SQL Server Datenbank in die Graphdatenbank Neo4j beschrieben. Dabei ist vor allem die Notwendigkeit der topologischen Information der Daten im relationalen System hervorzuheben. Anschließend wurden die Anwendungsbeispiele Abzweigeinfärbung, Ausfall einer Leitung und Netzverfolgung für jeweils ein Ortsnetz vorgestellt. Abschließend wurde gezeigt, wie die Anwendung auf mehrere Ortsnetze ausgeweitet werden kann.

Interpretation der Ergebnisse

Im Allgemeinen sind Graphdatenbanken nicht dazu entwickelt worden, relationale Datenbanken generell abzulösen, sondern können je nach Anwendungsfall eine weitere Option bieten. Auch für die konkreten Anwendungsbeispiele der Arbeit stellen Graphdatenbanken eine gute Erweiterung dar. Jedoch gibt es zahlreiche weitere Anwendungen, welche auf denselben Daten des Netzes aufbauen und bei denen die Geometrie von Relevanz ist. Diese geht jedoch bei der Graphdatenbank in der vorgestellten Form verloren. Man müsste also zwei Systeme führen, welche bestmöglich immer gleich aktuell sind.

Entscheidend für den Erfolg der Umsetzung war die bereits vorhandene Topologie in der SQLite Datenbank, welche aus der geometrischen Information mit Hilfe des FME Jobs hergestellt wurde. Dabei war wichtig, dass man auf die spezifischen Eigenschaften der Daten eingeht, was diesen Teil sehr individuell macht und erfreulicherweise in Zusammenarbeit mit der Energienetze Steiermark GmbH (EN) geglückt ist. Denn Daten sind in der Praxis leider nicht so sauber wie in der Theorie und Datenmodelle können über einen langen Zeitraum sehr komplex werden.

Die in der Einleitung angeführten gewünschten Anwendungsbeispiele konnten mit der dazu aufgesetzten Graphdatenbank erfolgreich umgesetzt werden. Die Umsetzung der Abfragen war einfacher als erwartet. Das spricht dafür, dass sich Graphdatenbanken gut für Netzwerkabfragen eignen und auch die Abfragesprache Cypher sehr benutzerfreundlich ist.

Beschränkungen der Forschung

Als eine Limitierung dieser Arbeit kann die gewählte Methodik des qualitativen Vergleiches gesehen werden. Dabei wird viel Spielraum für Interpretation gelassen und auch die Anforderungen basierend auf dem semistrukturierten Interview sind eher spezifisch auf die EN bezogen. Es wurde allerdings bewusst keine quantitative Evaluierung von Performance gemacht, da dieser viele Einflussfaktoren zugrunde liegen, was ein aussagekräftiges Ergebnis erschwert.

Auch das Datenmodell kann auf jeden Fall noch ausgefeilt werden, um noch mehr Information zu modellieren und Aspekte wie Gewichtungen oder Attribute für Schaltmodellierungen einzubringen. Für die Beantwortung der Forschungsfragen reicht aber auch das gewählte Datenmodell.

6.3 ZUSAMMENFASSUNG DER WISSENSCHAFTLICHEN ARBEITERGEBNISSE

Im Zuge dieser Masterarbeit wurden Stromdaten mehrerer Niederspannungs-ortsnetze der EN aus einer relationalen Datenbank in eine Graphdatenbank überführt. Dazu wurde zuerst ein Datenmodell definiert und darauf aufbauend die Anwendungsbeispiele Abzweigeinfärbung, Ausfall einer Leitung und Netzverfolgung umgesetzt. Die Voraussetzung dafür war die erstellte Topologie in der relationalen SQLite Datenbank. Die Abfragen für die genannten Anwendungsbeispiele sind erstaunlich kurz und einfach, was für die Eignung von Graphdatenbanken für Netzwerkberechnungen von EVU spricht.

Zusätzlich wurde eine qualitative Evaluierung zu den beiden Datenbanksystemen relationale Datenbanken und Graphdatenbanken durchgeführt. Dafür wurde zuerst ein semistrukturiertes Interview mit der EN gemacht, um das Ergebnis des qualitativen Vergleichs auf den Anwendungsfall von EVU zu bewerten, da die Beurteilung der dazu gewählten Kriterien stark vom Anwendungsfall abhängen. Auch wenn Graphdatenbanken viele Vorteile mit sich bringen und als vielversprechend gelten, werden sie die relationalen Datenbanken nicht ablösen. Es geht darum, dass man je nach Anwendung das geeignetere System wählt. So wird es auch bei der EN der Fall sein. Das gewonnene Wissen wird in den Ausbau der bisherigen Systeme einfließen, indem es in ein Webinterface verpackt und als zusätzliche Anwendung für Netzwerkberechnungen verwendet wird.

Der Wert dieser Arbeit liegt auf der Spezialisierung der qualitativen Evaluierung im Bereich von EVU. Denn in der Literatur gibt es für andere Anwendungsbereiche viele qualitative und quantitative Vergleiche zwischen relationalen Datenbanken und Graphdatenbanken und noch öfter zwischen relationalen Datenbanken und den relativ allgemein gesteckten NoSQL Datenbanken.

6.4 AUSBLICK

Es ist zu erwarten, dass die vielfältige Verarbeitung an Daten für EVU weiterhin zunehmen wird. Neben vielen anderen Anwendungen wächst auch die Anzahl an Netzwerkberechnungen stetig an.

Daher ist das Erstellen eines ausgereiften Datenmodells von besonderer Bedeutung. Dafür ist einerseits Expertise und Erfahrung von EVU und andererseits auch die Kenntnis über alle geeigneten Datenbanksysteme erforderlich. Darüber hinaus wäre eine allgemeine Norm des Datenmodells wünschenswert, damit auch der Wissenstransfer zwischen Netzbetreibern

leichter möglich ist. Außerdem werden die Daten auch für den Export für Simulations- und Analysefunktionen von externer Software benötigt.

Graphdatenbanken befinden sich derzeit gerade im Aufschwung, wodurch in den nächsten Jahren viele neue Entwicklungen und Anwendungen zu erwarten sind.

LITERATUR

- [1] “Stromnetze als Garanten der Versorgungssicherheit,” *LAND AM STROM Jahresbericht Oesterreichs Energie 2021*, Oesterreichs Energie, Hrsg., S. 10–11, Juni 2021. Adresse: https://oesterreichsenergie.at/fileadmin/user_upload/Oesterreichs_Energie/Publikationsdatenbank/Jahresbericht/2021/OEE_Jahresbericht_2021.pdf (besucht am 29.05.2022).
- [2] Open Geospatial Consortium, Hrsg., *OpenGIS Implementation Standard for Geographic information - Simple Feature Access - Part 1: Common Architecture*, 2011. Adresse: <http://www.opengis.net/doc/is/sfa/1.2.1>.
- [3] S. Liu, G. Lin, B. Chai u. a., “The future of graph database applications: An electric utility perspective,” in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, Chengdu, China: IEEE, Dez. 2017, S. 223–227. DOI: 10.1109/ITNEC.2017.8284941.
- [4] T. Studer, *Relationale Datenbanken*, 2. Auflage. Berlin, Heidelberg: Springer Vieweg, 2019, ISBN: 978-3-662-58976-2.
- [5] W. Kresse und D. M. Danko, *Springer Handbook of Geographic Information*, 1. Auflage. Berlin, Heidelberg: Springer, 2012, ISBN: 978-3-540-72678-4.
- [6] D. Fasel und A. Meier, *Big Data: Grundlagen, Systeme und Nutzungspotenziale*, 1. Auflage. Wiesbaden: Springer Fachmedien, 2016, ISBN: 978-3-658-11589-0.
- [7] I. Robinson, E. Efrem und J. Webber, *Graph databases*, 2. Auflage. Sebastopol, CA: O’Reilly Media, 2015, ISBN: 978-1-4919-3200-1.
- [8] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi und F. Ismaili, “Comparison between relational and NOSQL databases,” in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia: IEEE, Mai 2018, S. 216–221. DOI: 10.23919/MIPRO.2018.8400041.
- [9] A. Corbellini, C. Mateos, A. Zunino, D. Godoy und S. Schiaffino, “Persisting big-data: The NoSQL landscape,” *Information Systems*, Jg. 63, S. 1–23, 2017, ISSN: 03064379. DOI: 10.1016/j.is.2016.07.009.

- [10] S. Nickel, S. Rebennack, O. Stein und K.-H. Waldmann, “Graphentheorie,” in *Operations Research*, 3. Auflage, Berlin, Heidelberg: Springer, 2022, S. 119–150, ISBN: 978-3-662-65346-3.
- [11] U. Brandes, “Graphentheorie,” in *Handbuch Netzwerkforschung*, C. Stegbauer und R. Häußling, Hrsg., 1. Auflage, Wiesbaden: VS Verlag für Sozialwissenschaften, 2010, S. 345–353, ISBN: 978-3-531-92575-2.
- [12] M. Nebel und S. Wild, *Entwurf und Analyse von Algorithmen* (Studienbücher Informatik), 2. Auflage. Wiesbaden: Springer Vieweg, 2018, ISBN: 978-3-658-21155-4.
- [13] P. A. Longley, M. F. Goodchild, D. J. Maguiere und D. W. Rhind, *Geographic information science and systems*, 4. Auflage. Hoboken, NJ: Wiley, 2015, ISBN: 978-1-118-67695-0.
- [14] A. J. Schwab und S. Börnick, *Elektroenergiesysteme: Erzeugung, Transport, Übertragung und Verteilung elektrischer Energie*, 1. Auflage. Berlin Heidelberg: Springer, 2006, ISBN: 978-3-540-29664-5.
- [15] T. Patel und T. Eltaieb, “Relational database vs NoSQL,” *Journal of Multidisciplinary Engineering Science and Technology (JMEST)*, Jg. 2, Nr. 4, S. 691–695, Apr. 2015, ISSN: 3159-0040.
- [16] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen und D. Wilkins, “A comparison of a graph database and a relational database: A data provenance perspective,” in *Proceedings of the 48th Annual Southeast Regional Conference*, Bd. 10, Oxford, MS, USA: Association for Computing Machinery, Apr. 2010, S. 1–6. DOI: 10.1145/1900008.1900067.
- [17] Y. Chen, “Comparison of graph databases and relational databases when handling large-scale social data,” Masterarbeit, University of Saskatchewan, Saskatoon, Kanada, 2016, 91 S. Adresse: <https://harvest.usask.ca/bitstream/handle/10388/7434/CHEN-THESIS-2016.pdf?isAllowed=y&sequence=1> (besucht am 06.06.2022).
- [18] M. Miler, D. Medak und D. Odobašić, “The shortest path algorithm performance comparison in graph and relational database on a transportation network,” *PROMET - Traffic&Transportation*, Jg. 26, Nr. 1, S. 75–82, Feb. 2014. DOI: 10.7307/ptt.v26i1.1268.
- [19] M. Sharma, V. D. Sharma und M. M. Bundele, “Performance Analysis of RDBMS and No SQL Databases: PostgreSQL, MongoDB and Neo4j,” in *2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, Jaipur, India: IEEE, Nov. 2018, S. 1–5. DOI: 10.1109/ICRAIE.2018.8710439.
- [20] R. Angles und C. Gutierrez, “Survey of graph database models,” *ACM Computing Surveys*, Jg. 40, Nr. 1, S. 1–39, Feb. 2008, ISSN: 0360-0300. DOI: 10.1145/1322432.1322433.

- [21] L. Piepmeyer, *Grundkurs Datenbanksysteme*, 1. Auflage. München: Carl Hanser Verlag, 2011, ISBN: 978-3-446-42875-1.
- [22] J. J. Miller, “Graph database applications and concepts with neo4j,” in *Proceedings of the southern association for information systems conference*, Atlanta, GA, USA, März 2013, S. 141–147.
- [23] Neo4j. “Transition from relational to graph database - neo4j,” Neo4j Developer. (2023), Adresse: <https://neo4j.com/developer/graph-db-vs-rdbms/> (besucht am 03.02.2023).
- [24] M. Linnemann, “Unterschiedliche Akteure, Marktrollen und ihre Bedeutung,” in *Energiewirtschaft für (Quer-)Einsteiger*, M. Linnemann, Hrsg., 1. Auflage, Wiesbaden: Springer Vieweg, 2021, S. 17–212, ISBN: 978-3-658-33144-3.
- [25] Microsoft. “The transaction log (SQL server),” Microsoft Learn. (2022), Adresse: <https://learn.microsoft.com/en-us/sql/relational-databases/logs/the-transaction-log-sql-server> (besucht am 14.11.2022).
- [26] Microsoft. “View the SQL server error log (SSMS),” Microsoft Learn. (2022), Adresse: <https://learn.microsoft.com/en-us/sql/relational-databases/performance/view-the-sql-server-error-log-sql-server-management-studio> (besucht am 14.11.2022).
- [27] K. Gorman, A. Hirt, D. Noderer u. a., *Introducing Microsoft SQL Server 2019*, 1. Auflage. Birmingham, UK: Packt Publishing Ltd, 2019, ISBN: 978-1-83882-621-5.
- [28] E-Control, Hrsg., *STATISTIKBROSCHÜRE 2022*, 2022. Adresse: <https://www.e-control.at/documents/1785851/1811582/E-Control-Statbro-2022-Deutsch.pdf/4ee3d3d6-6eb2-32ce-c516-d3b036d1d2ff?t=1668007223110> (besucht am 29.12.2022).
- [29] Neo4j. “Indexes for search performance,” Neo4j Docs. (2022), Adresse: <https://neo4j.com/docs/cypher-manual/5/indexes-for-search-performance/> (besucht am 30.11.2022).

ANHANG

.1 TRANSKRIPT EXPERTENINTERVIEW

Interviewpartner: Dipl.-Ing. Werner Samhaber

Datum: 11. Jänner 2023 um 13:30 Uhr

Ort: Büro Energienetze Steiermark GmbH

I: Hallo nochmal und herzlichen Dank, dass du dir die Zeit nimmst, meine Fragen zu beantworten.

B: Gar kein Problem. Ich beantworte gerne deine Fragen.

I: Es geht um die qualitative Evaluierung zwischen den Datenbanksystemen relationale Datenbanken und Graphdatenbanken. Für den Vergleich wurden unterschiedliche Kriterien gewählt und ausgearbeitet. Diese sollen dann aus Sicht von EVU interpretiert werden, wofür ich gerne die Anforderungen von EVU zu diesen Kriterien mit diesem Interview abstecken möchte. Für eine gemeinsame Gesprächsbasis würde ich nun gerne die Kriterien vorlegen und gemeinsam durchgehen.

[...]

I: Ich werde nun in Folge die Kriterien aufzählen und bitte dich zu jedem Kriterium dein Input zu geben.

I: Sicherheit

B: Aus Sicht eines EVU als Betreiber einer kritischen Infrastruktur ist die IT-Sicherheit der betriebenen/eingesetzten Systeme von hoher Wichtigkeit. So werden auch laufend alle Systeme hinsichtlich IT-Sicherheit nach den ISO 27001 Richtlinien geprüft.

I: Einfachheit der Programmierung und Abfragesprache

B: Eine rasche Reaktion auf Kundenforderungen sowie die Dynamik der Entwicklungen im Energieumfeld erfordern eine zeitnahe Aktualisierung und Verbesserung der eingesetzten Softwarekomponenten. Dies gelingt wesentlich leichter mit standardisierten, einfachen Programmier- bzw. Scriptsprachen. Proprietäre Softwarekomponenten mit speziellen Parametrierumgebungen treten dabei in den Hintergrund.

I: Flexibilität

B: Die aktuellen Digitalisierungsinitiativen in EVU's mit agilen, modernen Entwicklungsumgebungen erfordern in Zukunft erhöhte Anforderungen an Anpassbarkeit und Flexibilität der eingesetzten Softwarelösungen. Die Integration bzw. das Zu-

sammenspiel einzelner Softwarekomponenten müssen einfach implementierbar sein.

I: Performance von Netzberechnungen

B: Die Anwender der eingesetzten Softwarelösungen akzeptieren (im Besonderen im Web-Umfeld) nur mehr performante Anwendungen, mit einfacher Bedienbarkeit. Gerade im Funktionsumfeld von Web-GIS Lösungen mit der sehr interaktiven Benutzung von Spezialfunktionen wie Netzwerkverfolgung sind kurze Antwortzeiten erwünscht.

I: Skalierbarkeit

B: Die Skalierbarkeit von Softwarelösungen ist im regionalen Bereich mit kleinen Datenmengen noch weniger relevant. Wenn man aber an Cloud-Lösungen, ggf. als Serviceportal für Dritte denkt, ist dieses Kriterium zukünftig beachtenswert.

I: Ausgereiftheit/Support

B: Da Softwarelösungen meist einige Jahre in Betrieb sind, ist ein laufender Support / Weiterentwicklung notwendig.

I: Somit sind wir schon am Ende angelangt. Ich danke dir für diesen wertvollen Input und deine Zeit.

B: Sehr gerne.