



**Hochschule
Bonn-Rhein-Sieg**
University of Applied Sciences

Projektarbeit

PostgresSQL - Rekursion auf Basis generischer Stored Procedures

Fachbereich Informatik
Referent: Prof. Dr. Harm Knolle

eingereicht von:
Jennifer Wittling

Sankt Augustin, den 09.12.2018

Zusammenfassung

Problemstellung und Erkenntnisinteresse

In den letzten Jahren haben Graphdatenbanken an Bedeutung gewonnen, da sich mit diesen bestimmte Fragestellungen besonders schnell lösen lassen. Graphdatenbanken haben den Vorteil, dass sich insbesondere Beziehungen zwischen Objekten gut abbilden und sehr performant abfragen lassen. Bei relationalen Datenbanken ist es zur Darstellung von Beziehungen zwischen Objekten erforderlich die verschiedenen Tabellen mittels des JOIN Operators zu verknüpfen. Diese Verknüpfungen können schnell zu einem großen Rechenaufwand und langen Laufzeiten führen. Es soll am Beispiel von Postgres untersucht werden, ob und wie sich Graphen in relationalen Datenbanken abbilden lassen. Weiterhin soll analysiert werden, ob und für welche Problemstellungen es sinnvoller ist Graphen in einer relationalen Datenbank statt einer Graphdatenbank abzubilden. Ist es zukünftig notwendig für die performante Verarbeitung steigender Datenmengen auf neue Technologien, wie Graphdatenbanken zu schwenken oder lassen sich die klassischen relationalen Datenbanken so erweitern, dass diese Problemstellungen ähnlich effizient lösen können.

Aktueller Forschungsstand

NoSQL Datenbanken und insbesondere Graphdatenbanken sind im Gegensatz zu den relationalen Datenbanken flexibler und bei der Lösung bestimmter Probleme weniger rechen- und speicherintensiv. Insbesondere wenn es um die Auflösung von Beziehungen bzw. um die Traversierung über einen Graphen geht, bieten Graphdatenbanken Vorteile gegenüber den herkömmlichen relationalen Datenbanken. In der Praxis wurde jedoch auch die Beobachtung gemacht, dass durch die Verwendung von Stored Procedures die Traversierung über einen Graphen mittels einer relationalen Datenbank ähnlich schnell umgesetzt werden kann, wie mit einer Graphdatenbank.

Zielsetzung

Es soll das Modell als grundlegender technologische Aspekt von Graphdatenbanken kurz erläutert werden. Zielsetzung dieser Arbeit ist es einen Graphen in der relationalen Datenbank Postgres abzubilden und zu vergleichen, wie sich die Traversierung über diesen Graphen effizient umsetzen lässt. Zunächst soll die Umsetzung mittels klassischer SQL Operationen erfolgen. Anschließend sollen die Problemstellungen mittels Stored Procedures, sowie der Rekursion mittels PL/SQL gelöst werden. Die Ergebnisse der verschiedenen Vorgehensweisen sollen miteinander verglichen werden.

Inhaltsverzeichnis

Zusammenfassung	II
1 Graph-Datenbanken - Grundlegende technologische Aspekte	2
1.1 Modell	2
1.1.1 Graph	2
1.1.2 Reguläre Graphen	3
1.1.3 Planare Graphen	4
1.1.4 Property Graphen	4
1.1.5 k-Partite Graphen	5
1.1.6 Hypergraphen	6
2 Graph-Datenbanken und -Frameworks - Ausgewählte Systeme	8
2.1 PostgreSQL	8
2.1.1 Visitenkarte des Systems	8
3 Graph-Datenbanken im praktischen Einsatz: OLTP	10
3.1 PostgresSQL: OLTP	10
3.1.1 Ausgewählte Use Cases	10
3.1.2 CSV-Import	10
3.1.3 Beurteilung	11
4 Graph-Datenbanken im praktischen Einsatz: OLAP	12
4.1 PostgreSQL: OLAP	12
4.1.1 Benchmark	12
4.1.2 Standard SQL	13
4.1.3 Stored Procedures	13
4.1.4 PL/SQL-Recursion	13
4.1.5 Datenbankzugriffe	13
4.1.6 Zugriffsart Aggregation	13
4.1.7 Zugriffsart Traversierung	13
4.1.8 Interpretation der Ergebnisse	13
Literaturverzeichnis	14
Eidesstattliche Erklärung	16

1 Graph-Datenbanken - Grundlegende technologische Aspekte

1.1 Modell

Ein Modell ist eine vereinfachte bzw. abstrahierte Darstellung von realen Gegenständen, Sachverhalten oder Problemen. Durch die Modellierung soll die Realität auf die wichtigsten Einflussfaktoren reduziert werden. Die Graphentheorie spielt eine zentrale Rolle bei der Modellierung von Problemen und Sachverhalten, da sich Graphen sehr gut zur Darstellung vernetzter Daten eignen. Aus diesem Grund haben Graphen in den letzten Jahren auch eine wichtige Rolle in der Datenbankwelt erhalten. Im folgenden sollen verschiedene Arten von Graphen kurz vorgestellt werden.

1.1.1 Graph

Ein Graph ist mathematisch folgendermaßen definiert:

Definition. *Ein Graph $G = (V, E, \gamma)$ ist ein Tripel bestehend aus:*

- V , einer nicht leeren Menge von Knoten(vertices)
- E , einer Menge von Kanten (edges) und
- γ , einer Inzidenzabbildung (incidence relation), mit
 $\gamma : E \longrightarrow \{X | X \subseteq V, 1 \leq |X| \leq 2\}$

Ein Knoten $a \in V$ und eine Kante $e \in E$ heißen inzident (incident) genau dann wenn a entweder Anfangs- oder Endecke von e ist. Es gilt $a \in \gamma(e)$. Zwei Knoten $a, b \in V$ heißen adjazent(adjacent) genau dann wenn es eine Kante e gibt die zu a und b inzident ist. Es gilt $\exists e \in E : \gamma(e) = \{a, b\}$.¹

Ein Knoten repräsentiert ein Element in einem Graphen. Die Kanten stellen die Beziehung zwischen den einzelnen Knoten her. In einem einfachen Graphen kann eine Kante immer nur jeweils zwei Knoten miteinander verbinden.

¹[Bec18, Seite 21]

Graphen können gerichtet oder ungerichtet sein. Gerichtete Graphen zeichnen sich dadurch aus, dass die Kanten eine zugewiesene Richtung besitzen. Um die Beziehung zwischen zwei Knoten genauer zu definieren, lassen sich die Kanten gewichten. Dabei werden den Kanten in der Regel numerische Werte zugeordnet und man bezeichnet diese Graphen als Gewichtete Graphen. Hat eine Kante als Start- und Endknoten den selben Knoten, verbindet also den Knoten mit sich selber, spricht man von einer Schlinge. Liegen zwischen zwei Knoten eines Graphen mehr als eine Kante, nennt man diese Multikante. Schlingen und Multikanten dürfen in einem einfachen Graphen nicht auftauchen.² Multigraphen hingegen erlauben Multikanten. In Pseudographen sind sowohl Multikanten als auch Schleifen möglich.

Werden Kanten und Knoten eines Graphs vertauscht entsteht der Kantengraph bzw. Line-Graph des jeweiligen Graphen $L(G)$. Zwei Graphen können isomorph sein. Der Grad eines Knoten bezeichnet die Anzahl der inzidenten Kanten des Knoten. Dabei werden Schleifen doppelt gezählt.³ Sind bei einem Graphen alle Knoten mit allen übrigen Knoten verbunden spricht man von einem vollständigen Graphen:

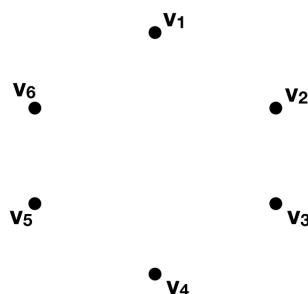
$$K_n = ([n], \binom{[n]}{2})$$

Ein ungerichteter Graph heißt zusammenhängend, falls zwischen zwei beliebigen Knoten a und b aus V es einen ungerichteten Weg mit a als Startknoten und b als Endknoten gibt.⁴

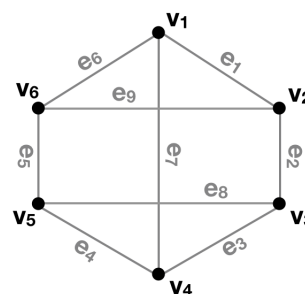
1.1.2 Reguläre Graphen

Bei regulären Graphen haben alle Knoten den selben Knotengrad. Als Knotengrad wird die Anzahl direkter Nachbarn, also alle Knoten die über eine Kante direkt mit dem betrachteten Knoten verbunden sind, bezeichnet.⁵ Abbildung 1.1.2 zeigt einen regulären Graphen mit Knotengrad null und einen mit einem Grad von drei.

0-Regulärer Graph



3-Regulärer Graph



²Vgl. [SF05]

³Vgl. [Rah17, Seite 13]

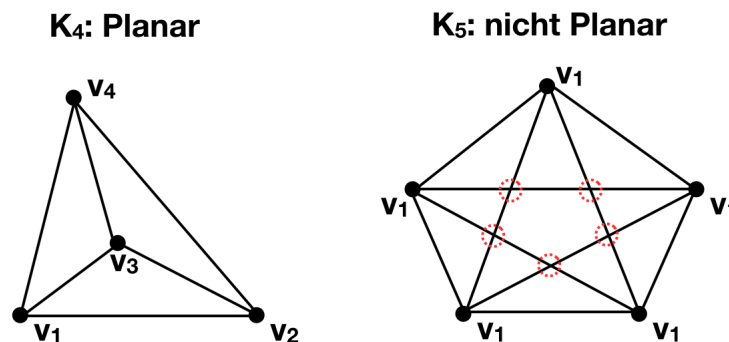
⁴[KN12, 36-38]

⁵[Fel03]

1.1.3 Planare Graphen

Planare Graphen lassen sich in der Ebene ohne Überschneidung der Kanten zeichnen.⁶ Eine Darstellung eines Graphen G in der Ebene ohne Kantenüberkreuzungen wird planare Einbettung von G genannt.

In Abbildung 1.1.3 sind die vollständigen Graphen K_4 und K_5 abgebildet, wobei es sich bei K_4 um einen planaren Graphen und bei K_5 um einen nicht planaren Graphen handelt. K_5 ist nicht planar, da sich dieser in der Ebene nicht ohne Überschneidungen der Kanten zeichnen lässt. Die Überschneidungen sind in der Abbildung rot markiert.



Eulerscher Polyedersatz

Für zusammenhängende planare Graphen besagt der Eulersche Polyedersatz, dass die Anzahl der Knoten minus die Anzahl der Kanten plus die Anzahl der Gebiete zwei ergibt:

$$n - m + f = 2$$

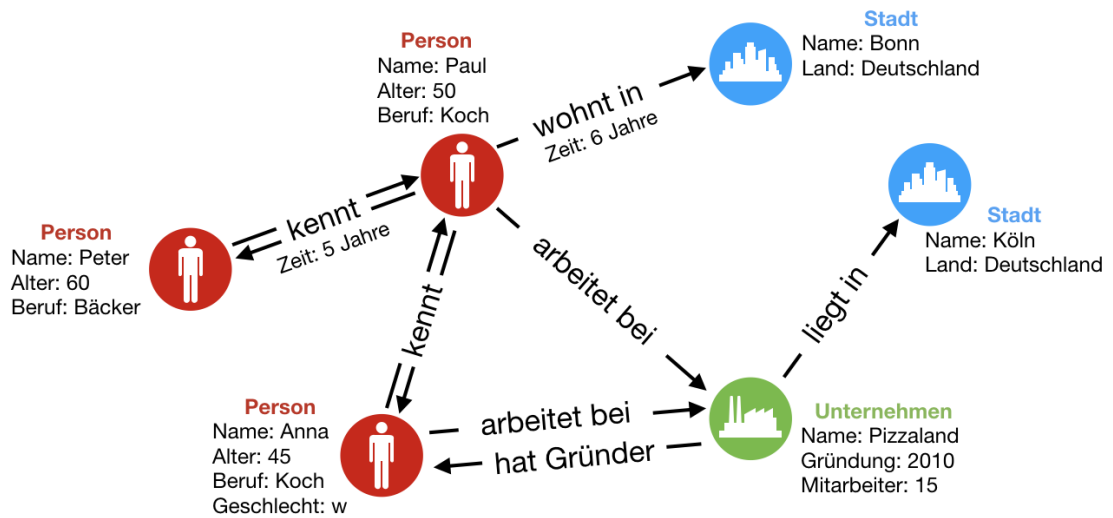
1.1.4 Property Graphen

Property Graphen sind gerichtete Graphen, die sich durch ihre den Kanten und Knoten zugewiesenen Eigenschaften (Properties) auszeichnen. Gespeichert werden diese Eigenschaften als Key-Value-Paare. Label ermöglichen die Unterteilung von Knoten und Kanten in verschiedene Knoten- und Kantentypen. Attribute, Label und die Richtung der Kanten erlauben eine sehr detaillierte modellierung von realen Sachverhalten. Somit sind Property Graphen von sehr großer Bedeutung für Graphdatenbanken.

Abbildung 1.1.4 zeigt einen Property Graphen. Die Knoten sind den drei Labeln Person, Unternehmen und Stadt zugeordnet. Die gerichteten Kanten stellen die Beziehungsverhältnisse zwischen den einzelnen Knoten her und können durch Attri-

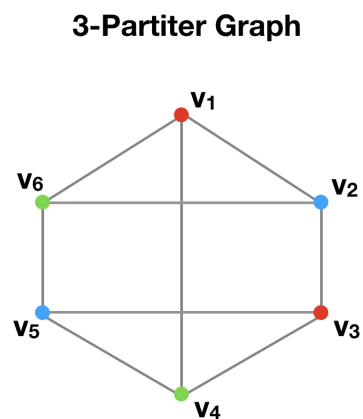
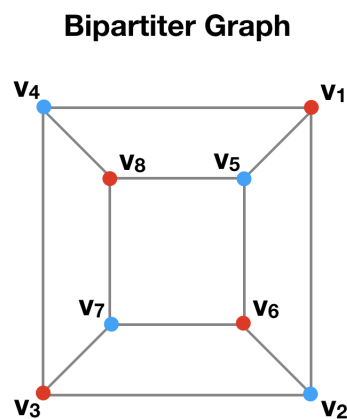
⁶[TT16]

bute, wie beispielsweise der Information über die Dauer der bisherigen Beziehung, genauer definiert werden.



1.1.5 k-Partite Graphen

Die Knoten können in k Partitionen unterteilt werden, sodass die Knoten in einer Gruppe keine direkten Nachbarn sind. Für $k = 2$ werden diese Graphen Bipartite Graphen genannt. Die folgende Abbildung 1.1.5 zeigt zwei k -Partite Graphen, wobei die Partitionen durch die unterschiedlichen Farben der Knoten gekennzeichnet sind. Die Darstellung verdeutlicht auch, dass die Anzahl der Knoten eines Graphs keinen direkten Einfluss auf die Anzahl der Partitionen hat.



1.1.6 Hypergraphen

Hypergraphen haben die Eigenschaft, dass Kanten im Gegensatz zu klassischen Graphen mehr als zwei Knoten miteinander verbinden können.

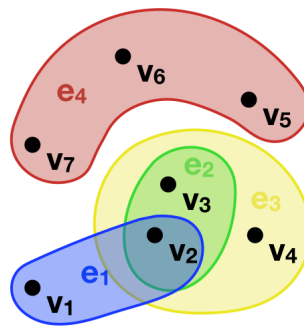
Mathematisch ist ein Hypergraph folgendermaßen definiert:

Definition. Let $X = \{v_1, v_2, \dots, v_n\}$ be a finite set, and let $E = \{e_1, e_2, \dots, e_m\}$ be a family of subsets of X such that

$$e_i \neq \emptyset (i = 1, 2, \dots, m) \cup_{i=1}^m e_i = X.$$

The pair $H = (X, E)$ is called a hypergraph with vertex set X and hyperedge set E . The elements v_1, v_2, \dots, v_n of X are vertices of hypergraph H , and the sets e_1, e_2, \dots, e_m are hyperedges of hypergraph H .⁷

Abbildung 1.1.6 zeigt einen Hypergraphen. Die Kante e_4 verbindet in diesem Graphen die Knoten v_5, v_6 und v_7 miteinander.



Im Vergleich zum normalen Graphen können die Kanten eines Hypergraphen eine beliebige Kardinalität haben (siehe Definition Graph Kapitel 1.2.1). Beim normalen Graphen können die Kanten nur die Kardinalität $1 \leq |X| \leq 2$ haben. Die Hyperedges in einem Hypergraphen sind somit eine beliebige Menge von Knoten. In einem normalen Graphen sind die Kanten, eine in einem Intervall festgelegte Menge von Knoten:

$$X = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\} \text{ Knoten}$$

$$E = \{e_1, e_2, e_3, e_4\} \text{ Kanten}$$

$$E = \{e_1, e_2, e_3, e_4\} = \{\{v_1, v_2, v_3\}, \{v_2, v_3\}, \{v_3, v_5, v_6\}, \{v_4\}\}$$

⁷Vgl. [ZSHZ18, Seite 2]

2 Graph-Datenbanken und -Frameworks - Ausgewählte Systeme

2.1 PostgreSQL

2.1.1 Visitenkarte des Systems

- Allgemein
 - Name: PostgreSQL, umgangssprachlich Postgres
 - Kategorie / Modell: PostgreSQL ist ein Relationales Datenbank System
 - Version: 11.1
 - Historie: PostgreSQL ist aus dem POSTGRES Projekt der University of California at Berkeley entstanden, welches unter der Leitung von Professor Michael Stonebraker im Jahre 1986 began.
 - Hersteller: PostgreSQL Global Development Group
 - Lizenz: Open-Source
 - Referenzen / Quellenangaben: [Fro18], [Pos18], [Groa], [Eis03]
- Architektur
 - Programmiersprache: C
 - Systemarchitektur: Objektrelationales Datenbankmanagementsystem
 - Betriebsart: Stand Alone, Cluster Betrieb für Replikation der Datenbank
 - API: u.A. libpq, psycopg, psqLODBC, pq, pgtnlg, Npgsql, node-postgres

- Datenmodell
 - Standardsprache: PL/pgSQL
 - Sichten (Views): Ja
 - Externe Dateien (BLOBs): Ja
 - Schlüssel: Ja
 - Semantische unterschiedliche Beziehungen: Ja
 - Constraints: Ja
- Indexe
 - Sekundärindexe: Ja
 - Gespeicherte Prozeduren: Ja
 - Triggermechanismen: Ja, Prozeduren , die als Trigger aufgerufen werden
 - Versionierung: Ja, Versionierung mit Hilfe von Transaktions-ID(XID)
- Abfragemethoden
 - Kommunikation, Protokoll(HTTP/REST): SQL über TCP/IP
 - CRUD-Operationen: Ja
 - Ad-hoc-Anfragen: Ja
- Konsistenz
 - ACID/BASE: ACID, besonders MVCC(Multiversion Concurrency Control Modell)
 - Transaktionen: Ja
 - Nebenläufigkeit (Synchronisation): Ja
- Administration
 - Werkzeuge: pgAdmin, dataGrip, diverse Erweiterungen
 - Massendatenimport: Ja
 - Datensicherung: Ja
 - Recovery: Ja

3 Graph-Datenbanken im praktischen Einsatz: OLTP

3.1 PostgreSQL: OLTP

3.1.1 Ausgewählte Use Cases

3.1.2 CSV-Import

Beim Import von (CSV)-Dateien wird zwischen import vom Clientsystem und Import vom Serversystem unterschieden. Für den Import vom Client wird das psql-Statement `\copy` verwendet. `\copy` liest Informationen aus einer Datei die vom psql-Client aus erreichbar sein muss. [Pos18]

Listing 3.1: CSV Input

```
\copy Beitraege
FROM './data/Beitraege.csv' DELIMITER ',' CSV HEADER;
```

Listing 3.2: Anlegen der Tabelle facebook-profiles

```
CREATE TABLE public."facebook-profiles"
(
    id serial PRIMARY KEY,
    first TEXT,
    last TEXT,
    gender TEXT,
    country TEXT,
    birth TEXT
);
```

Listing 3.3: Anlegen der Tabelle facebook

```
CREATE TABLE public.facebook  
(  
    src INT,  
    dst INT,  
    type TEXT,  
    date TEXT  
);
```

Listing 3.4: Hinzufügen von Fremdschlüsseln

```
ALTER TABLE public.facebook  
ADD CONSTRAINT "facebook_facebook-profiles_id_fk"  
FOREIGN KEY (src) REFERENCES public."facebook-profiles" (id)
```

3.1.3 Beurteilung

4 Graph-Datenbanken im praktischen Einsatz: OLAP

4.1 PostgreSQL: OLAP

4.1.1 Benchmark

Mit der Standardinstallation von PostgreSQL wird auch pgbench mitinstalliert. Bei pgbench handelt es sich um ein einfaches Tool zur Durchführung von Benchmark-Tests. Bei einem Benchmark-Test wird eine Menge von SQL-Statements beliebig oft wiederholt, dabei können auch mehrere parallele Sessions geöffnet werden. Beim durchführen des Tests berechnet pgbench die Anzahl der Transaktionen pro Sekunde.

Verwendung von pgbench

pgbench wird über die Kommandozeile gestartet. Dabei können eine Reihe von Parametern übergeben werden, mit denen das Verhalten von pgbench gesteuert werden kann.

- -c clients
Über das Flag -c wird die Anzahl der Clients bzw. die Anzahl der gleichzeitigen Datenbankverbindungen festgelegt. Wenn hier nichts angegeben ist wird nur ein Client verwendet.
- -t transactions
Über das Flag -t wird festgelegt wieviele Transaktionen jeder Client durchführt. Die Anzahl aller Transaktionen ergibt sich durch das Produkt von Clients und Transactions.

4.1.2 Standard SQL

4.1.3 Stored Procedures

4.1.4 PL/SQL-Recursion

4.1.5 Datenbankzugriffe

4.1.6 Zugriffsart Aggregation

4.1.7 Zugriffsart Traversierung

4.1.8 Interpretation der Ergebnisse

Literaturverzeichnis

- [Ang12] ANGLES, Renzo: A comparison of current graph database models. In: *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on IEEE*, 2012, S. 171–177
- [APPDSLP13] ANGLES, Renzo ; PRAT-PÉREZ, Arnau ; DOMINGUEZ-SAL, David ; LARRIBA-PEY, Josep-Lluís: Benchmarking database systems for social network applications. In: *First International Workshop on Graph Data Management Experiences and Systems* ACM, 2013, S. 15
- [AU95] AHO, Alfred V. ; ULLMAN, Jeffrey D.: *Foundations of computer science*. USA : Computer Science Press, 1995 <http://infolab.stanford.edu/~ullman/focs.html>
- [Bec18] BECKER, Peter: *Graphentheorie*. <http://www2.inf.h-brs.de/~pbecke2m/graphentheorie/einfuehrung.pdf>. Version: 2018
- [Eis03] EISENTRAUT, Peter: *PostgreSQL Das Offizielle Handbuch*. mitp-Verlag GmbH/Bonn, 2003
- [Fel03] FELSNER, Stefan: *Geometric graphs and arrangements: some chapters from combinatorial geometry*. Springer Science & Business Media, 2003
- [Fro18] FROEHLICH, Lutz: *PostgreSQL 10*. Carl Hanser Verlag München, 2018
- [Groat] GROUP, The PostgreSQL Global D.: *PostgreSQL 11.1 Documentation*. <https://www.postgresql.org/files/documentation/pdf/11/postgresql-11-A4.pdf>
- [Grob] GROUP, The PostgreSQL Global D.: *PostgreSQL 8.4 Documentation*. <https://www.postgresql.org/docs/8.4/datatype.html>
- [Gru17] GRUCIA, Jelena: *PostgreSQL and GraphQL*. <https://blog.cloudboost.io/postgresql-and-graphql-2da30c6cde26>. Version: 2017
- [KHA⁺16] KUCUK, Ahmet ; HAMDI, Shah M. ; AYDIN, Berkay ; SCHUH, Michael A. ; ANGRYK, Rafal A.: Pg-Trajectory: A PostgreSQL/PostGIS based data model for spatiotemporal trajectories. In: *2016*

IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom) IEEE, 2016, S. 81–88

- [KN12] KRUMKE, S.O. ; NOLTEMEIER, H.: *Graphentheoretische Konzepte und Algorithmen*. Vieweg+Teubner Verlag, 2012 (Leitfäden der Informatik). – ISBN 9783834822642
- [Kud15] KUDRASS, Thomas: *Taschenbuch Datenbanken*. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2015
- [Pos18] POSTGRESQL GLOBAL DEVELOPMENT GROUP: *PostgreSQL 11.1 Documentation*. <https://www.postgresql.org/docs/11>. Version: 2018
- [Rah17] RAHMAN, Saidur: *Basic Graph Theory*. Berlin, Heidelberg : Springer, 2017. – ISBN 978–3–319–49475–3
- [Red12] REDMOND, Eric: *Sieben Wochen, sieben Datenbanken*. O'Reilly Verlag, 2012
- [Sas18] SASAKI, Bryce M.: *Graph Databases for Beginners: The Basics of Data Modeling*. <https://neo4j.com/blog/data-modeling-basics/>. Version: 2018
- [SF05] STEFAN FELSNER, Gesine K. Christine Puhl P. Christine Puhl: *Graphentheorie*. <http://page.math.tu-berlin.de/~felsner/Lehre/GrTh05/Graphentheorie.pdf>. Version: 2005
- [TT16] In: THORSTEN THEOBALD, Sadik I.: *Graphen*. Springer Fachmedien Wiesbaden, 2016
- [ZSHZ18] ZHANG, Hongliang ; SONG, Lingyang ; HAN, Zhu ; ZHANG, Yingjun: *Hypergraph Theory in Wireless Communication Networks*. Springer, 2018

Eidesstattliche Erklärung

Ich versichere an Eides Statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Alle Stellen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Ich versichere außerdem, dass ich keine andere als die angegebene Literatur verwendet habe. Diese Versicherung bezieht sich auch auf alle in der Arbeit enthaltenen Zeichnungen, Skizzen, bildlichen Darstellungen und dergleichen.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Sankt Augustin,
den 9. Dezember 2018
Ort, Datum

Jennifer Wittling