

Wilhelm Büchner Hochschule

Darmstadt

Fachbereich Informatik

Entwicklung eines Konzepts und Handlungsanweisungen zur Optimierung der  
Massenzuordnung von Rechnungen und Zahlungen mittels Indizes

|               |  |
|---------------|--|
| vorgelegt bei | Herr Pat.-Ing. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Dipl.-Inform. Dipl.-<br>Wirtsch.-Inform. Jürgen R. Dietrich MBM, (Univ.) |
| von           | Marko Kovacevic  |
| Matr.-Nr.     | 882059   |
| Anschrift     | Kreuzherrenstr. 3, 53227 Bonn  |
| Abgabetermin  | 02.08.2017   |

# Danksagung

An dieser Stelle möchte ich allen danken, die mich bei der Erstellung dieser Bachelorarbeit unterstützt haben. Zuallererst möchte ich mich bei meinem Betreuer Herrn Pat.-Ing. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Dipl.-Inform. Dipl.-Wirtsch.-Inform. Jürgen R. Dietrich MBM, (Univ.) herzlich für die konstruktive Unterstützung sowie für die immer schnellen und informativen Rückmeldungen bedanken.

Meinem Fachmentor Stepan Rutz danke ich sowohl für die zahlreichen Diskussionen über Datenbanken und Indizes als auch für die eingeräumte Zeit, mich mit der Bachelorarbeit zu beschäftigen.

Meiner Freundin Nina Colak danke ich für die Aufmunterung und ihre hilfreichen Kommentare.

Außerdem gilt mein Dank meinem Arbeitskollegen Matthias Olschewski für seine Mühen und seine Zeit, die er für das Korrekturlesen aufgewendet hat.

Desweiteren möchte ich mich bei der Scopevisio AG sowie bei allen Mitarbeitern der Entwicklung für ihre Hilfe bedanken.

Ich widme diese Arbeit meinen Eltern Barica und Mijo Kovacevic, die mir das Informatik Studium und damit auch diese Bachelorarbeit ermöglicht haben.

# Abstract

Das Ziel der vorliegenden Bachelorarbeit ist die Optimierung der Massenzuordnung (Auszifferung) von Rechnungen und Zahlungen. Dabei gilt der Verwendungszweck als ausschließliches Zuordnungskriterium. Dazu werden Fragen bezüglich der Performance und Zuordnungsmöglichkeiten beantwortet.

Die Performance wird datenbankseitig durch Indizes optimiert, weshalb auf die Funktionsweise und Leistungsfähigkeit unterschiedlicher Indextypen eingegangen wird.

Im Hinblick auf die Zuordnungsmöglichkeiten erfolgt eine Untersuchung der SQL-Operatoren auf ihre Stringmatching Möglichkeiten in einer Testumgebung, wobei verschiedene Indizes zum Einsatz kommen. Die Ergebnisse werden zur besseren Vergleichbarkeit grafisch gegenübergestellt.

Das Ergebnis der Arbeit sind Handlungsanweisungen zur Optimierung der Zuordnung von Rechnungen und Zahlungen sowie Formatvorschläge für den Verwendungszweck, um eine möglichst effiziente und genaue Zuordnung zu ermöglichen. Abschließend werden die dabei gewonnenen Erkenntnisse über Indizes in Kombination mit den verwendeten Operatoren zusammengetragen.

# **Abstract**

The aim of this Bachelor thesis is the optimization of the mass assignment of invoices and payments based on reference data. Questions regarding the performance and assignment possibilities are also within the scope of this study.

The performance is optimized through indexing therefore several different index types are checked in terms of functionality and efficiency.

Regarding the assignment possibilities, SQL-operators are examined in respect of their stringmatching capabilities. Results are measured in a test environment where several different indexes are used. The outcome is structured into different diagrams and tables in order to gain an easy overview.

The conclusion of this thesis are instructions and manuals as well as format suggestions for the reference line to ensure an efficient and an exact assignment of invoices and payments.

Finally, the results which have been gained about indexes in combination with the used operators are summarized.

# Sperrvermerk

Scopevisio AG

26.06.2017

Wilhelm Büchner Hochschule

Postfach 10 01 64

64201 Darmstadt

Vertrag über einen Sperrvermerk zur Abschlussarbeit von Herrn Kovacevic (882059)

Die vorliegende Abschlussarbeit beinhaltet vertrauliche Informationen der Scopevisio AG. Veröffentlichungen und Vervielfältigungen sind ohne ausdrückliche vorherige schriftliche Genehmigung der Scopevisio AG nicht gestattet. Die Abschlussarbeit ist nur den Gutachtern und den Mitgliedern des Prüfungsausschusses zugänglich zu machen.

Unterschrift Firma

Unterschrift

Wilhelm Büchner Hochschule

Unterschrift

Hochschulbetreuer

# Inhaltsverzeichnis

|       |   |    |
|-------|---|----|
| 1     | Einleitung.....   | 1  |
| 1.1   | Aufgabenstellung und Motivation.....                        | 1  |
| 1.2   | Scopevisio AG.....  | 3  |
| 1.3   | Überblick über die Arbeit.....                              | 3  |
| 2     | Stand der Technik und Stand der Wissenschaft.....           | 5  |
| 2.1   | Grundlegendes zu Datenbanken.....                           | 5  |
| 2.1.1 | Datenbanksystem (DBS).....                                  | 5  |
| 2.1.2 | Relationale Datenbanken.....                                | 6  |
| 2.1.3 | SQL.....  | 8  |
| 2.2   | Klassifikation von Algorithmen.....                         | 10 |
| 2.3   | PostgreSQL.....   | 11 |
| 2.4   | Der Datenbankindex.....                                     | 11 |
| 2.5   | Erläuterung der verschiedenen Indextypen.....               | 13 |
| 2.5.1 | B-Baum-Index.....   | 13 |
| 2.5.2 | GIN-Index.....  | 18 |
| 2.5.3 | GiST-Index.....   | 19 |
| 2.5.4 | Trigramm-Index.....   | 19 |
| 2.5.5 | Die Volltextsuche und ihre Indizierungsmöglichkeiten.....   | 22 |
| 2.6   | Tabellen und Indizes bei PostgreSQL.....                    | 26 |
| 2.6.1 | Speicherorganisation in PostgreSQL.....                     | 26 |
| 2.6.2 | Abfragestatistik.....                                       | 29 |
| 2.7   | Möglichkeiten der Teilstringsuche mittels SQL-Abfragen..... | 32 |
| 2.7.1 | Equal (=).....  | 32 |
| 2.7.2 | LIKE.....   | 33 |
| 2.7.3 | ILIKE.....  | 33 |
| 2.7.4 | Reguläre Ausdrücke.....                                     | 34 |
| 2.7.5 | Volltextsuche.....  | 36 |
| 2.7.6 | Trigram-Similarity.....                                     | 37 |
| 3     | Testumgebung / -verfahren.....                              | 39 |
| 3.1   | Die Journal-Tabelle.....                                    | 40 |
| 3.2   | Testdatengenerierung.....                                   | 41 |
| 3.3   | Testfälle.....  | 42 |

|   |        |
|---|--------|
| 3.3.1 FixSizeQueryTest.....   | 42     |
| 3.3.2 ScalingTest.....  | 43     |
| 3.3.3 IndexSizeTest.....  | 44     |
| 3.3.4 Testverlauf.....  | 45     |
| 3.3.5 Hardware.....   | 45     |
| 3.4 Zusammenfassung Kapitel 3.....  | 46     |
| 4 Testdurchführung.....   | 47     |
| 4.1 Auswertung der Testergebnisse.....  | 49     |
| 4.2 Abfragen mit unverändertem Verwendungszweck.....                                  | 50     |
| 4.2.1 Abfragen auf exakte Übereinstimmung des Verwendungszwecks.....                  | 50     |
| 4.2.2 Abfragen unter Auslassung von beliebigen führenden Zeichen.....                 | 58     |
| 4.2.3 Abfragen unter Auslassung von beliebigen endenden Zeichen.....                  | 63     |
| 4.2.4 Abfragen unter Auslassung von beliebigen führenden und endenden<br>Zeichen..... | 69     |
| 4.3 Abfragen mit Teilen des Verwendungszwecks.....                                    | 74     |
| 4.4 Abfragen auf aufbereiteten Daten des Verwendungszwecks.....                       | 78     |
| 4.5 Volltextsuche.....  | 83     |
| 4.6 Ähnlichkeitsüberprüfung.....  | 87     |
| 4.7 Größenentwicklung der Indizes.....  | 90     |
| 4.8 Zusammenfassung Kapitel 4.....  | 91     |
| 5 Ergebnis mit Ausblick.....  | 92     |
| 5.1 Zuordnungsstrategien.....   | 92     |
| 5.2 Formatvorschläge für den Verwendungszweck.....                                    | 95     |
| 5.3 Erkenntnisse über Indizes.....  | 96     |
| 5.4 Ausblick.....   | 97     |
| 6 Quellenverzeichnis.....   | XCVIII |
| 7 Anhang.....   | CI     |

## Abbildungsverzeichnis

|  |    |
|--|----|
| Abbildung 1: Beispiel einer relationalen Datenbank.....  | 7  |
| Abbildung 2: Blattknoten mit Referenzen auf die Tabelle.....   | 13 |
| Abbildung 3: Wurzel-, Zweig- und Blattknoten.....  | 14 |
| Abbildung 4: Suche nach dem Wert „57“ im B-Baum.....   | 15 |
| Abbildung 5: Case-Szenarien für einen Binär Baum.....  | 16 |
| Abbildung 6: GIN-Index Blattknoten mit doppelten Werten.....   | 18 |
| Abbildung 7: Trigramm Erstellung mit der Funktion show_trgm().....                                       | 20 |
| Abbildung 8: Tabelle mit den Spalten ctid, firstname, lastname.....                                      | 20 |
| Abbildung 9: Trigramme der Spalte firstname.....   | 21 |
| Abbildung 10: Erzeugung eines tsvector.....  | 24 |
| Abbildung 11: Ausgabe der Funktion „ts_debug“.....   | 24 |
| Abbildung 12: Pfad der Tabelle „journal“.....  | 26 |
| Abbildung 13: Speicherung der Tabelle auf dem Datenträger.....   | 27 |
| Abbildung 14: Ermittlung der Tupel- und Pageanzahl für die Tabelle journal.....                          | 28 |
| Abbildung 15: Tupel mit dem Wert „RECHNUNG 6127675“ in der Spalte „documentext“<br>.....                 | 31 |
| Abbildung 16: Abfragestatistik ohne Index.....   | 31 |
| Abbildung 17: Abfragestatistik mit Index.....  | 32 |
| Abbildung 18: tsquery Erstellung.....  | 37 |
| Abbildung 19: „Entfernung“ der Wörter 'PostgreSQL' und 'Deutsche Post'.....                              | 38 |
| Abbildung 20: Auswertungsprogramm.....   | 49 |
| Abbildung 21: Laufzeit mit Operator Equal, Indizes: Ohne, B-Baum.....                                    | 51 |
| Abbildung 22: Laufzeit mit Operator Equal, Index: B-Baum.....  | 51 |
| Abbildung 23: Laufzeit mit Operator Equal (LOWER), Indizes: Ohne, FB-B-Baum.....                         | 52 |
| Abbildung 24: Laufzeit mit Operator Equal (LOWER), Index: FB-B-Baum.....                                 | 52 |
| Abbildung 25: Laufzeit mit Operator LIKE, Indizes: Ohne, B-Baum, Trigramm (GIN),<br>Trigramm (GiST)..... | 53 |
| Abbildung 26: Laufzeit mit Operator LIKE Index: B-Baum.....  | 53 |
| Abbildung 27: Laufzeit mit Operator LIKE (LOWER), Indizes: Ohne, FB-B-Baum.....                          | 54 |
| Abbildung 28: Laufzeit mit Operator LIKE (LOWER), Indizes: FB-B-Baum.....                                | 54 |
| Abbildung 29: Laufzeit mit Operator ILIKE, Indizes: Ohne, Trigramm (GIN), Trigramm<br>(GiST).....        | 55 |



|  |    |
|--|----|
| Abbildung 30: Laufzeit mit Operator ILIKE, Indizes: Trigramm (GIN), Trigramm (GiST)  | 55 |
| Abbildung 31: Laufzeit mit Operator Equal, Index: B-Baum / Operator Equal (LOWER),<br>Index: FB-B-Baum / Operator LIKE (LOWER), Index: FB-B-Baum / Operator LIKE,<br>Index: B-Baum / Operator ILIKE, Index: Trigramm (GIN) | 56 |
| Abbildung 32: Laufzeit mit Operator Equal, Index: B-Baum / Operator Equal (LOWER),<br>Index: FB-B-Baum / Operator LIKE (LOWER), Index: FB-B-Baum / Operator LIKE,<br>Index: B-Baum   | 56 |
| Abbildung 33: Maximale Laufzeit mit Operator Equal, Indizes: B-Baum, FB-B-Baum   | 57 |
| Abbildung 34: Laufzeit mit Operator %LIKE, Indizes: Ohne, Trigramm (GIN), Trigramm (GiST)  | 59 |
| Abbildung 35: Laufzeit mit Operator %LIKE, Indizes: Trigramm (GIN), Trigramm (GiST)  | 59 |
| Abbildung 36: Laufzeit mit Operator %ILIKE, Indizes: Ohne, Trigramm (GIN), Trigramm (GiST)   | 60 |
| Abbildung 37: Laufzeit mit Operator %ILIKE, Indizes: Trigramm (GIN), Trigramm (GiST)   | 60 |
| Abbildung 38: Maximale Laufzeit mit Operator %LIKE und %ILIKE, Indizes: Trigramm (GIN)   | 61 |
| Abbildung 39: Laufzeit mit Operator LIKE%, Indizes: Ohne, B-Baum, Trigramm (GIN),<br>Trigramm (GiST)   | 64 |
| Abbildung 40: Laufzeit mit Operator LIKE%, Index: B-Baum   | 64 |
| Abbildung 41: Laufzeit mit Operator LIKE (LOWER), Indizes: Ohne, FB-B-Baum   | 65 |
| Abbildung 42: Laufzeit mit Operator LIKE (LOWER), Index: FB-B-Baum   | 65 |
| Abbildung 43: Laufzeit mit Operator ILIKE%, Indizes: Ohne, Trigramm (GIN), Trigramm (GiST)   | 66 |
| Abbildung 44: Laufzeit mit Operator ILIKE%, Indizes: Trigramm (GIN), Trigramm (GiST)   | 66 |
| Abbildung 45: Laufzeit mit Operator LIKE%, Index: B-Baum / Operator: LIKE% (LOWER),<br>Index: FB-B-Baum / ILIKE%, Index: Trigramm (GIN)  | 67 |
| Abbildung 46: Laufzeit mit Operator LIKE%, Index: B-Baum / Operator: LIKE% (LOWER),<br>Index: B-Baum   | 67 |
| Abbildung 47: Maximale Laufzeit mit Operator LIKE%, Index: B-Baum / Operator: LIKE% (LOWER),<br>Index: B-Baum  | 67 |

|   |    |
|---|----|
| Abbildung 48: Laufzeit mit Operator %LIKE%, Indizes: Ohne, Trigramm (GIN),<br>Trigramm (GiST).....                        | 70 |
| Abbildung 49: Laufzeit mit Operator %LIKE%, Indizes: Trigramm (GIN), Trigramm<br>(GiST).....                              | 70 |
| Abbildung 50: Laufzeit mit Operator %LIKE%, Indizes: Ohne, Trigramm (GIN),<br>Trigramm (GiST).....                        | 71 |
| Abbildung 51: Laufzeit mit Operator %LIKE%, Indizes: Trigramm (GIN), Trigramm<br>(GiST).....                              | 71 |
| Abbildung 52: Maximale Laufzeit mit Operator %LIKE%, Indizes: Trigramm (GIN),<br>Trigramm (GiST).....                     | 72 |
| Abbildung 53: Laufzeit mit Operator RegExp., Indizes: Ohne, Trigramm (GIN),<br>Trigramm (GiST).....                       | 75 |
| Abbildung 54: Laufzeit mit Operator RegExp., Trigramm (GIN), Trigramm (GiST).....   | 75 |
| Abbildung 55: Maximale Laufzeit mit Operator RegExp., Index: Trigramm (GIN).....  | 76 |
| Abbildung 56: Laufzeit mit Operator Equal auf extra Spalte, Indizes: Ohne, B-Baum. .                                      | 79 |
| Abbildung 57: Laufzeit mit Operator Equal auf extra Spalte, Index: B-Baum.....  | 79 |
| Abbildung 58: Laufzeit mit Operator LIKE auf extra Spalte, Indizes: Ohne, B-Baum,<br>Trigramm (GIN), Trigramm (GiST)..... | 80 |
| Abbildung 59: Laufzeit mit Operator LIKE auf extra Spalte, Indizes: B-Baum-, Trigramm<br>(GIN), Trigramm (GiST).....      | 80 |
| Abbildung 60: Laufzeit mit Operator LIKE auf extra Spalte, Index: B-Baum.....   | 80 |
| Abbildung 61: Laufzeit auf extra Spalte mit Operator Equal, Index: B-Baum / Operator<br>LIKE, Index: B-Baum.....          | 81 |
| Abbildung 62: Maximale Laufzeit auf extra Spalte mit Operator Equal, Index: B-Baum /<br>Operator LIKE, Index: B-Baum..... | 81 |
| Abbildung 63: Laufzeit der Volltextsuche, Indizes: Ohne, GIN, GiST.....   | 83 |
| Abbildung 64: Laufzeit der Volltextsuche, Indizes: GIN, GiST.....   | 83 |
| Abbildung 65: Laufzeit der Volltextsuche, Indizes: GIN.....   | 84 |
| Abbildung 66: Maximale Laufzeit der Volltextsuche, Index: Trigramm (GIN).....   | 84 |
| Abbildung 67: tsquery-Erzeugung des Verwendungszwecks „RECHNUNG#3501404“  | 85 |
| Abbildung 68: to_tsvector-Erzeugung des Verwendungszwecks „Rechnung<br>RE\3501404“ .....                                  | 85 |
| Abbildung 69: Volltextsuche der Zahlung Rechnung „RE\3501404“ und Rechnung<br>„RECHNUNG#3501404“ .....                    | 85 |
| Abbildung 70: tsquery-Erzeugung des Verwendungszwecks „RECHNUNG.3501404“.   | 86 |

|   |    |
|---|----|
| Abbildung 71: ts_debug-Information des Verwendungszwecks „RECHNUNG.3501404“<br>.....                        | 86 |
| Abbildung 72: tsvector-Erzeugung des Verwendungszwecks „Rechnung\3501404“....                               | 86 |
| Abbildung 73: ts_debug-Information des Verwendungszwecks „Rechnung\3501404“.                                | 86 |
| Abbildung 74: Volltextsuche der Zahlung Rechnung „RECHNUNG.3501404“ und<br>Zahlung „RECHNUNG\3501404“ ..... | 87 |
| Abbildung 75: Laufzeit mit Similarity-Operator, Indizes: Ohne, Trigramm (GiST).....                         | 88 |
| Abbildung 76: Laufzeit mit Similarity-Operator, Indizes: Trigramm (GiST).....                               | 88 |
| Abbildung 77: Indexgrößenentwicklung bei skalierenden Daten.....  | 90 |

Abbildung ohne Angabe einer Quelle sind vom Autor erstellt.

# Abkürzungsverzeichnis

|        |   |
|--------|---|
| ANSI   | American National Standards Institute       |
| CRM    | Customer Relationship Management            |
| CSV    | Comma Separated Values                      |
| DBS    | Datenbanksystem                             |
| DBMS   | Datenbankmanagementsystem                   |
| DCL    | Data Control Language                       |
| DDL    | Data Description Language                   |
| DML    | Data Manipulation Language                  |
| DMS    | Dokumenten Management System                |
| FB(I)  | Funktionsbasierter (Index)                  |
| FTS    | Full Text Search                            |
| GIN    | Generalized Inverted Index                  |
| GiST   | Generalized Search Tree                     |
| I/O    | Input / Output                              |
| RegExp | Regular Expression / Regulärer Ausdruck     |
| SaaS   | Software as a Service                       |
| SPARC  | Standard Planning and Requirement Committee |
| SQL    | Structured Query Language                   |

## Tabellenverzeichnis

|   |    |
|---|----|
| Tabelle 1: Erläuterung der B-Baum Indexerstellung.....                              | 17 |
| Tabelle 2: Erläuterung der "ts_debug()" Funktion.....                               | 25 |
| Tabelle 3: System-Views „pg_stat_user_tables“, „pg_statio_user_tables“.....         | 29 |
| Tabelle 4: Spaltenerläuterung der View „table_stats“.....                           | 30 |
| Tabelle 5: Erläuterung der Platzhalter des LIKE-Befehls.....                        | 33 |
| Tabelle 6: Regulärer Ausdruck mit Bereichsangabe.....                               | 34 |
| Tabelle 7: Regulärer Ausdruck mit Bereichsangabe und Quantor * (Stern).....         | 35 |
| Tabelle 8: Regulärer Ausdruck mit Zeichenklasse und Quantor + (Plus).....           | 36 |
| Tabelle 9: Beispiele der Funktion „word_similarity“.....                            | 38 |
| Tabelle 10: Spaltenerläuterung der Tabelle "journal".....                           | 40 |
| Tabelle 11: Format des Verwendungszwecks.....                                       | 41 |
| Tabelle 12: Unterstützte Indizes für die Suche nach exakter Übereinstimmung.....    | 50 |
| Tabelle 13: Exakte Zuordnungen von Rechnungen und Zahlungen.....                    | 57 |
| Tabelle 14: Unterstützte Indizes der Abfrage LIKE / ILIKE %documenttext.....        | 58 |
| Tabelle 15: Erfolgversprechende Zuordnungen der %ILIKE Abfrage.....                 | 62 |
| Tabelle 16: Fehlzusordnungen der %ILIKE Abfrage.....                                | 62 |
| Tabelle 17: Zahlenformat mit fester Länge.....                                      | 62 |
| Tabelle 18: Unterstützte Indizes der Abfrage LIKE / ILIKE documenttext%.....        | 63 |
| Tabelle 19: Erfolgversprechende Zuordnungen der LIKE% / ILIKE% Abfrage.....         | 68 |
| Tabelle 20: Fehlzusordnungen der LIKE% / ILIKE% Abfrage.....                        | 68 |
| Tabelle 21: Zahlenformat mit fester Länge.....                                      | 68 |
| Tabelle 22: Unterstützte Indizes der Abfrage LIKE / ILIKE %documenttext%.....       | 69 |
| Tabelle 23: Erfolgversprechende Zuordnungen der Abfrage %LIKE% / %ILIKE%.....       | 73 |
| Tabelle 24: Fehlerhafte Zuordnung. Führende und Endende Zeichen umgehen.....        | 73 |
| Tabelle 25: Unterstützte Indizes bei regulären Ausdrücken (~).....                  | 74 |
| Tabelle 26: Erfolgversprechende Zuordnungen der Abfrage mit regulärem Ausdruck. .76 | 76 |
| Tabelle 27: Neue Spalte zur Speicherung aufbereiteter Daten.....                    | 78 |
| Tabelle 28: Unterstützte Indizes bei exakter Übereinstimmung (extra Spalte).....    | 78 |
| Tabelle 29: Erfolgversprechende Zuordnungen der Abfrage.....                        | 82 |
| Tabelle 30: Unterstützte Indizes bei der Volltextsuche.....                         | 83 |
| Tabelle 31: Erfolgversprechende Zuordnungen der Volltextsuche.....                  | 85 |
| Tabelle 32: Unterstützte Indizes für die Similarity-Überprüfung.....                | 87 |
| Tabelle 33: Erfolgversprechende Zuordnung durch die Abfrage auf Ähnlichkeit.....    | 89 |

Tabelle 34: Fehlzuordnungen von Rechnungen und Zahlungen. Operator: Similarity. .89

# 1 Einleitung

Im ersten Kapitel erfolgt die Erläuterung der Aufgabenstellung und Motivation. Bei der Massenzuordnung von Rechnungen und Zahlungen kommt es zu Performance- und Zuordnungsproblemen. Diese sollen mittels Indizes und Zuordnungsstrategien gelöst werden.

Zudem wird die Firma Scopevisio AG vorgestellt, bei der diese Problemstellung im Bereich der Anwendung „Finanzen“ auftritt. Für dieses Unternehmen soll ein Konzept mit Handlungsanweisungen erarbeitet werden. Anschließend wird ein Überblick über die Arbeit gegeben.

## 1.1 Aufgabenstellung und Motivation

In der Scopevisio-Anwendung „Finanzen“ ist es möglich, Rechnungen und Zahlungen zu verwalten. Darunter gehört auch das Ausziffern, bei dem offene Posten gegeneinander ausgeglichen werden. Dabei geht es darum, Rechnungen und Zahlungen einander zuzuordnen, um diese als bezahlt zu markieren. Hierbei ergeben sich folgende Möglichkeiten:

1. Eine Rechnung wird durch genau eine Zahlung ausgeziffert.
2. Eine Rechnung wird durch mehrere Zahlungen ausgeziffert.
3. Mehrere Rechnungen werden durch eine Zahlung ausgeziffert.

Von einer Teilauszifferung wird gesprochen, wenn Zahlungseingänge einer Rechnung zugeordnet wurden, aber noch ein offener Restbetrag besteht.<sup>1</sup>

Eine Auszifferung kann manuell oder automatisch erfolgen. Bei der manuellen Vorgehensweise werden Rechnungen und Zahlungen vom Benutzer ausgewählt und abgearbeitet, während bei der automatischen Methode Rechnungen und Zahlungen anhand bestimmter Kriterien wie z.B. Rechnungsbetrag, Datum oder Verwendungszweck zur Auszifferung vorgeschlagen werden.

Dabei ergeben sich folgende Problemstellungen:

- Es gibt eine große Anzahl an Rechnungen und Zahlungen, die miteinander verglichen werden müssen (Performanceproblem).

---

1 Vgl. Scopevisio, Ausziffern-Ausgleich offener Posten, URL, 15.05.2017

- Finanzdaten haben kein festes Zahlungsformat. Daher entsprechen die Verwendungszwecke der Zahlungen nicht genau den Rechnungsnummern. Zudem können Rechnungsnummern in anderen enthalten sein: RE-100 ist in RE-10000 enthalten. Des Weiteren kann es dazu kommen, dass Kunden die angeforderte Rechnungsnummer nicht korrekt wiedergeben, welches sich in fehlenden Präfixen, in zusätzlichem Text im Verwendungszweck und nicht zuletzt in fehlerhaften Eingaben manifestiert (Zuordnungsproblem).

Im Folgenden wird die Zuordnung von Rechnungen und Zahlungen über die Kriterien „Rechnungsnummer“ und „Verwendungszweck“ als Zuordnung über den Verwendungszweck definiert.

Für die automatische Auszifferung werden unterschiedliche Kriterien herangezogen, um Rechnungen und Zahlungen gegeneinander auszufizieren. Ziel der Arbeit ist es, zu überprüfen, wie die Zuordnung ausschließlich über den Verwendungszweck beschleunigt und durch Teilzeichenketten-Suchstrategien optimiert werden kann. Hierzu soll eine Testdatenbank erstellt werden, die den Umstand der Auszifferung realitätsnah abbildet. Auf diesen Testdaten sollen Auszifferungsstrategien implementiert werden, die als Grundlage für Performancemessungen dienen. Die Optimierung erfolgt mittels unterschiedlicher Indizes, deren Funktionsweise und Eigenschaften erläutert sowie ihre Performance gegenübergestellt werden. Hierbei wird auf die spezifischen Eigenschaften der Datenbank eingegangen, die die Basis für die Messungen darstellen. Die Untersuchungen erfolgen ausschließlich innerhalb einer PostgreSQL-Datenbank, um Abhängigkeiten zu anderen Systemen zu vermeiden. Für den besseren Überblick und Vergleichsmöglichkeiten wird ein Programm entwickelt, das die Ergebnisse der Messungen grafisch darstellt.

Die Ausarbeitung soll der Scopevisio AG sowie gegebenenfalls anderen Firmen mit gleicher Problemstellung dabei helfen, für ihr jeweiliges Format des Verwendungszwecks und einer bestimmten Datenmenge sowohl die richtige Zuordnungsstrategie als auch den richtigen Index für die Zuordnung zu finden.

Darüber hinaus können daraus Ergebnisse für ähnliche Aufgabenstellungen mit großen Datenmengen abgeleitet und zusätzlich auch Erfahrungswerte über die Performance von jeweiligen Abfrage-Typen und Indizes gewonnen werden.



## 1.2 Scopevisio AG

Die Scopevisio AG wurde im Jahr 2007 mit Standort in Bonn gegründet. Etwa 60 Mitarbeiter sind 2017 im Unternehmen tätig. Die entwickelten Anwendungen werden ausschließlich in der Cloud betrieben und als SaaS (Software as a Service) angeboten. Neben einer hohen Verfügbarkeit von Software, Plattform und Infrastruktur stellt das Unternehmen ebenso die Pflege, Wartung, Updates und Weiterentwicklung der Anwendungen sicher. Die Unternehmenssoftware vereint die Bereiche CRM (Customer Relationship Management), Projekte, Abrechnung, Finanzen und Teamwork-DMS (Dokumenten-Management-System). Die Entwicklung sowie auch das Hosting finden ausschließlich in Deutschland statt, wobei sichergestellt wird, dass Unbefugte keinen Zugriff auf die Kundendaten haben.

Die Kostengestaltung ist benutzerfreundlich und richtet sich nach dem Bedarf des Kunden, da Anwendungen und Benutzer flexibel hinzugebucht und wieder gekündigt werden können. Nur die genutzten Dienstleistungen werden tagesgenau berechnet und monatlich abgebucht. Hierbei kann Scopevisio mit einem Versorger (Stromversorger) gleichgestellt werden, bei dem nur die in Anspruch genommene Leistung gezahlt wird.<sup>2</sup>

## 1.3 Überblick über die Arbeit

Nach dem einleitenden ersten Kapitel werden in Kapitel 2 grundlegende Informationen zu Datenbanken vermittelt. Danach wird die bei der Scopevisio AG eingesetzte Datenbank „PostgreSQL“ vorgestellt. Nach der Vorstellung des Index und seiner Verwendung, werden unterschiedliche Indextypen behandelt, wobei auf den Aufbau sowie auf die Funktionsweise eingegangen wird. Anschließend wird die Speicherorganisation von Relationen in PostgreSQL dargestellt. Es wird gezeigt, wie und wo Relationen gespeichert werden. Die Auswirkung von Indizes auf SQL-Abfragen wird anhand von Statistiken veranschaulicht, wodurch der Performancegewinn durch Indizes besser nachvollzogen werden kann. Abschließend werden Operatoren auf ihre Möglichkeiten bezüglich der Teilstringsuche untersucht.

Kapitel 3 stellt die Testumgebung vor, in der der Umstand der Auszifferung nachgebildet wird. Hierfür werden Testdaten und Testfälle entwickelt, die Antworten auf die gestellten Fragen der Zuordnungs- und Performanceprobleme geben werden.

---

<sup>2</sup> Vgl. Scopevisio AG, Unternehmenssoftware aus der Cloud - Unternehmensprofil Scopevisio AG, <https://www.scopevisio.com/downloads/unternehmen/Unternehmensprofil.pdf>, 15.05.2017

In Kapitel 4 werden die Messungen innerhalb der Testfälle durchgeführt, wobei die Operatoren aus Kapitel 2.7 zum Einsatz kommen. Die Auswertung der Ergebnisse erfolgt mittels eines eigens entwickelten Programms, das die Werte grafisch darstellt. Nach jeder Suchstrategie erfolgt eine Analyse der Zuordnungsmöglichkeiten sowie die Überprüfung der maximalen Verarbeitung innerhalb einer bestimmten Zeitspanne. Zuletzt werden in Kapitel 5 das Ergebnis und der Ausblick präsentiert. Es werden Handlungsanweisungen und Zuordnungsstrategien in Abhängigkeit eines vorliegenden Verwendungszwecks und einer Datenmenge gegeben. Zudem wird aufgeführt, welche Operatoren mit welchem Index die beste Performance liefern.

## **2 Stand der Technik und Stand der Wissenschaft**

In diesem Kapitel werden grundlegende Themen für das Verständnis der darauffolgenden Kapitel behandelt. Nach allgemeinen Informationen zu Datenbanken erfolgt eine Klassifikation von Algorithmen anhand der O-Notation. Weiterhin wird eine kurze Vorstellung der eingesetzten Datenbank „PostgreSQL“, des Indexes und dessen Verwendung gegeben. Anschließend werden die unterschiedlichen Indextypen erklärt, wobei deren Aufbau und ihre Funktionsweise erläutert werden. Es folgt eine Einführung in Trigramme und ihre Verwendung in Kombination mit Indizes. Zudem wird die Volltextsuche mit ihren Indizierungsmöglichkeiten veranschaulicht. Neben der Speicherorganisation von Relationen wird der Effekt von Indizes auf SQL-Abfragen mithilfe von Statistiken dargestellt. Abschließend werden SQL-Operatoren auf ihre Möglichkeiten der Teilstringsuche untersucht.

### **2.1 Grundlegendes zu Datenbanken**

Im Folgenden werden für die Arbeit relevante Informationen zu Datenbanken vermittelt. Dabei wird auf das Datenbanksystem eingegangen und kurz die Datenbankarchitektur ANSI-SPARC erläutert. Weiterhin wird das relationale Datenmodell veranschaulicht, wobei die Normalisierung angerissen wird. Daraufhin werden die Datenbanksprache SQL und abschließend Views, Funktionen und Trigger vorgestellt.

#### **2.1.1 Datenbanksystem (DBS)**

Ein Datenbanksystem besteht aus einer oder mehreren Datenbanken und dem Datenbankmanagementsystem (DBMS). Das DBMS ist die Software, die den Zugriff auf die Daten in der Datenbank kontrolliert und steuert. Dazu gehört ein Sicherheitssystem, das die Zugriffe unberechtigter Benutzer verhindert. Zudem kümmert es sich um die Konsistenz (Integrität) der Daten sowie um die Verwaltung des gleichzeitigen konkurrierenden Zugriffs von mehreren Benutzern über sogenannte Transaktionen.

## **Datenbankarchitektur**

Unabhängig vom Datenbanksystem und dem Datenmodell werden nach dem Architekturvorschlag ANSI (American National Standards Institute) SPARC (Standard Planning and Requirement Committee) drei Ebenen unterschieden. Durch die Trennung der Ebenen können jeweils Änderungen an ihnen vorgenommen werden, ohne dass dies Auswirkungen auf die anderen Ebenen hat.

### **Externe Ebene**

In der externen Ebene werden unterschiedliche Sichten auf die Daten in der Datenbank definiert. Die Sichten können für Benutzer sowie auch für Anwendungsprogramme festgelegt werden. Die Ebene stellt sicher, dass nur die Daten des Gesamtmodells nach außen sichtbar sind, für die eine Berechtigung existiert.

### **Konzeptionelle Ebene**

In dieser Ebene erfolgt die logische Darstellung der gesamten Daten und ihre Beziehungen zueinander. Die Darstellung erfolgt in einem bestimmten Datenmodell, in dem ein Teil der Realität nachgebildet wird. Im folgenden Kapitel wird das relationale Datenmodell erläutert.

### **Interne Ebene**

Auf dieser Ebene erfolgt die physikalische Speicherung der in der konzeptuellen Ebene definierten logischen Darstellung.<sup>3</sup>

## **2.1.2 Relationale Datenbanken**

Es gibt unterschiedliche Datenbankmodelle, die festlegen, wie die Daten strukturiert sind und zueinander in Beziehung stehen. Es wird unter anderem zwischen dem hierarchischen, netzwerkartigen und dem relationalen Modell unterschieden. Die bei der Arbeit eingesetzte Datenbank entspricht dem relationalen Modell, weshalb dieses im Folgenden näher erläutert wird.

Das relationale Modell wurde 1970 von E.F. Codd vorgeschlagen und gilt als Meilenstein der Datenbanken.<sup>4</sup>

Bei der relationalen Datenbank erfolgt die Speicherung der Daten in Form von Tabellen (Relationen), die eindeutige Namen besitzen. Die Tabellen enthalten Zeilen (Tupel) die jeweils einem Datensatz entsprechen. Jede Zeile enthält mehrere Spalten (Attribute).

---

<sup>3</sup> Vgl. Connolly, Begg, Strachan, 2002, S. 80-83

<sup>4</sup> Vgl. ebda. S. 112

„Der Grad einer Relation ist die Anzahl der Attribute, die sie enthält“<sup>5</sup>. In Abbildung 1 wird ein Ausschnitt eines Datenbankschemas für eine klassische Bibliothek gezeigt.

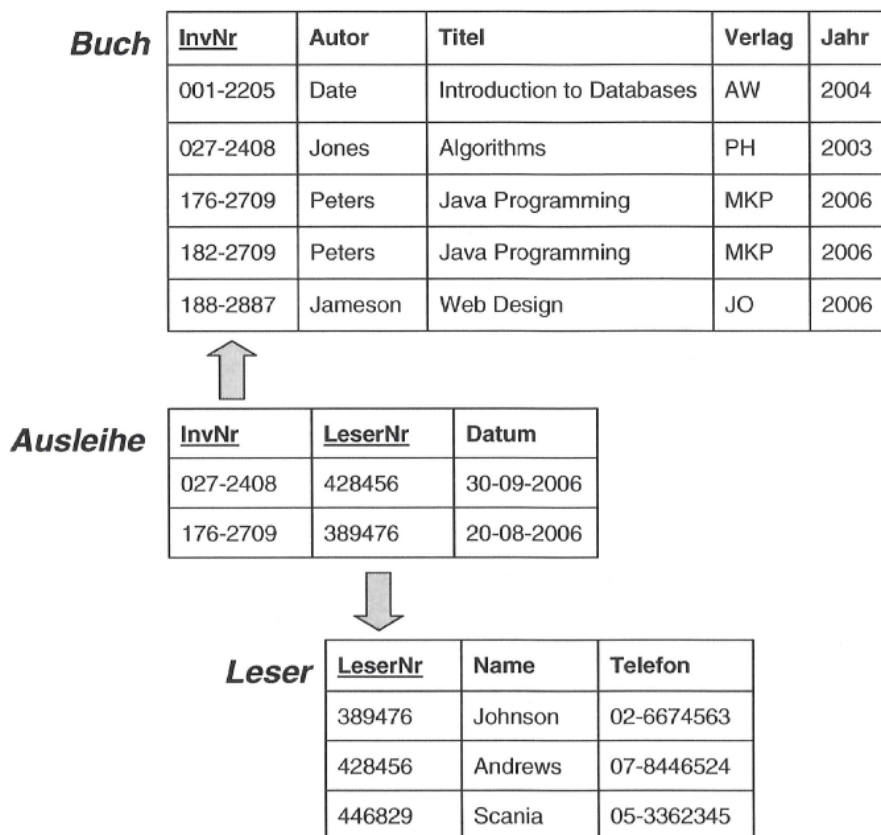


Abbildung 1: Beispiel einer relationalen Datenbank<sup>6</sup>

Es sind drei Tabellen mit den Namen „Buch“, „Ausleihe“ und „Leser“ abgebildet, die über eindeutige Schlüssel, sogenannte Primärschlüssel, in Beziehung stehen. Der Primärschlüssel ist in einer Tabelle einmalig. Er kennzeichnet einen Tupel eindeutig und ist in der Abbildung unterstrichen hervorgehoben.

Es ist zu erkennen, dass die Tabelle „Ausleihe“ zwei Spalten enthält, bei denen die Spaltennamen unterstrichen sind. Dabei handelt es sich um die Primärschlüssel der Tabelle Buch (InvNr) und der Tabelle Leser (LeserNr). Sie tauchen in der Tabelle „Ausleihe“ als Fremdschlüssel auf und werden mit einer zusätzlichen Information „Datum“ verknüpft. Die erste Spalte der Ausleihe ist wie folgt zu interpretieren:

Das Buch „Algorithms“ vom Autor "Jones" wurde von „Andrew“ am „30-09-2006“ ausgeliehen. Über die Primärschlüssel werden Tupel unterschiedlicher Tabellen in Verbindung gebracht.

5 Vgl. Connolly, Begg, Strachan, 2002, S.116

6 Vgl. Vossen, 2008, S. 96

## Normalisierung

Ziel beim Entwurf eines Datenbankschemas ist es, die einzelnen Tabellen so zu gestalten, dass Redundanzen und potentielle Inkonsistenzen vermieden werden. Hierfür gibt es Normalisierungsregeln, wobei bei der Umsetzung schrittweise eine Umstrukturierung der Relationen erfolgt. Es wird zwischen fünf Normalformen unterschieden, die aufeinander aufbauen. Ist z.B. die 3. Normalform gegeben, ist somit auch die 1. und 2. Normalform erfüllt.<sup>7</sup>

Durch die Normalformen wird zudem verhindert, dass bei dem Hinzufügen, Löschen, und Ändern von Tupel Probleme (Anomalien) entstehen.

### 2.1.3 SQL

SQL (Structured Query Language) ist eine Datenbanksprache relationaler Datenbanken.

Dabei erfolgt eine Unterscheidung zwischen drei Befehlskategorien:

- Data Description Language (DDL)  
Über die Datendefinitionssprache können Tabellen, Funktionen, Indizes, Views und Trigger etc. angelegt, bearbeitet und gelöscht werden.  
In dieser Arbeit wird auf die Befehle der Indexerstellung sowie in diesem Kapitel auf die Definition einer View und Funktion eingegangen.
- Data Control Language (DCL)  
Die Datenkontrollsprache dient zur Vergabe von Berechtigungen (Datensicherheit, Datenschutz). Die Befehle sind nicht Bestandteil dieser Arbeit.
- Data Manipulation Language (DML)  
Unter die Datenbearbeitungssprache fallen folgende Befehle: SELECT, INSERT, UPDATE, DELETE. Sie werden verwendet, um Daten zu lesen, zu schreiben, zu ändern und zu löschen.

Da sich die Ausarbeitung auf das Lesen von Daten konzentriert, wird im Folgenden der SELECT-Befehl näher erläutert. Zudem sind für den weiteren Verlauf der Arbeit Kenntnisse über Views, Funktionen und Trigger notwendig, auf die ebenfalls eingegangen wird.

---

<sup>7</sup> Vgl. Connolly, Begg, Strachan, 2002, S.224 -230

## **SELECT – Befehl**

Der wichtigste DML Befehl ist der SELECT-Befehl. Er dient dazu, Daten aus ein oder mehreren Tabellen zu lesen. Die Grundstruktur einer SELECT-Anweisung ist folgende:

`SELECT Attribute FROM Relation(en) WHERE Bedingungen`

Sie gibt an, welche Spalten (Attribute) aus einer oder mehreren Tabellen (Relationen), die einer oder mehrerer Bedingungen entsprechen, geladen werden sollen.<sup>8</sup>

Dabei wird das "Was" vom "Wie" getrennt. "Eine SQL-Anweisung beschreibt, was benötigt wird, ohne Anweisungen zu geben, wie das Ergebnis erreicht wird"<sup>9</sup>

Bei der Formulierung der SQL-Anweisung ist kein Wissen über die internen Abläufe der Datenbank notwendig. Auf den ersten Blick erscheint es, dass demnach kein Einfluss auf die Performance der SQL-Abfragen genommen werden kann. Im weiteren Verlauf der Arbeit wird jedoch gezeigt, wie durch Indizes die Performance von Abfragen verbessert wird.

## **Views (Sichten)**

Views können als logische oder virtuelle Tabellen verstanden werden. Sie werden über eine Datenbankabfrage definiert und können wie eine normale Tabelle abgefragt werden.<sup>10</sup> Die Daten bezieht die View über eine SELECT-Anweisung aus einer oder mehreren Tabellen.

## **Funktionen**

Innerhalb eines Datenbanksystems stehen zahlreiche Funktionen zur Verfügung. Darunter fallen mathematische Funktionen wie auch Funktionen zur Bearbeitung von Zeichenketten.

Im Verlauf der Arbeit wird die LOWER-Funktion verwendet. Sie nimmt eine Zeichenkette als Parameter entgegen und gibt diese in Kleinbuchstaben zurück. Es besteht auch die Möglichkeit, benutzerdefinierte Funktionen zu definieren.

## **Trigger**

Ein Trigger wird für eine Tabelle oder View erstellt und führt eine Funktion aus, wenn ein bestimmtes Ereignis eintritt. Dieses könnte zum Beispiel der DML-Befehl INSERT sein. Somit kann vor dem Einfügen eines Datensatzes eine Funktion ausgeführt werden, die Änderungen an den Daten vornimmt.

---

<sup>8</sup> Vgl. Vossen, 2008, S. 130

<sup>9</sup> Vgl. Winand, 2012, S. IV

<sup>10</sup> Vgl. W3schools, SQL Views, [https://www.w3schools.com/sql/sql\\_view.asp](https://www.w3schools.com/sql/sql_view.asp), 15.06.2017

## 2.2 Klassifikation von Algorithmen

Die einzelnen Abfragen auf den unterschiedlichen Indizes, die im Rahmen der Arbeit durchgeführt werden, werden nach ihrer Laufzeit klassifiziert. Dabei wird die O-Notation verwendet, die die Komplexität eines Algorithmus angibt. Die Laufzeit wird in Abhängigkeit einer Problemgröße  $N$  angegeben. Die Problemgröße ist in unserem Fall die Anzahl der Rechnungen und Zahlungen.

Es wird zwischen folgenden Laufzeiten unterschieden:

### **Konstante Laufzeiten** $O(1)$

Bei Programmen mit konstanter Laufzeit hängt die Laufzeit nicht von  $N$  ab.

### **Logarithmisch** $O(\log N)$

Anwendungen mit logarithmischer Laufzeit sind kaum langsamer als Programme konstanter Laufzeit.<sup>11</sup>

### **Lineare Laufzeit** $O(N)$

Wenn sich  $N$  verdoppelt, verdoppelt sich auch die Laufzeit.

### **Leicht überlinear** $O(N \log N)$

Wenn sich  $N$  verdoppelt, wird die Laufzeit mehr als doppelt so groß (aber nicht wesentlich mehr).

### **Quadratisch** $O(N^2)$

Wenn sich  $N$  verdoppelt, vervierfacht sich die Laufzeit.

### **Kubische Laufzeitanalyse** $O(N^3)$

Wenn sich  $N$  verdoppelt, erhöht sich die Laufzeit um das Achtfache.<sup>12</sup>

---

<sup>11</sup> Vel. Sedgewick Sedgewick, Wayne, 2014, S.206

<sup>12</sup> Vgl. Sedgewick, 1992, S. 97



## 2.3 PostgreSQL

Die Scopevisio-Anwendung verwendet die Datenbank PostgreSQL. PostgreSQL ist ein freies Open-Source-Programm und kann unter den Bedingungen der PostgreSQL License genutzt werden. Die SQL-Implementierung unterstützt den ANSI-SQL:2008 Standard und bietet darüber hinaus eigene Erweiterungen an. „PostgreSQL bietet ~ 90% der Funktionen / Leistungen, die in führenden proprietären Datenbanken zu finden sind.“<sup>13</sup> Darunter fallen unter anderem die Möglichkeit der Erzeugung eigener Datentypen sowie erweiterbare Abfragemethoden, um nach ihnen zu suchen. Des Weiteren steht eine Volltextsuche zur Verfügung. Es gibt unterschiedliche Indizierungsmöglichkeiten, worunter der GIN- (Generalized Inverted) und GiST- (Generalized Search Tree)-Index fallen.

Zudem können zahlreiche Erweiterungen nachinstalliert werden. Dazu gehört auch das Trigramm-Modul. Das Trigramm-Modul sowie die oben erwähnten Indizes werden in dieser Arbeit genauer untersucht.

## 2.4 Der Datenbankindex

Bevor auf die verschiedenen Typen von Indizes eingegangen wird, wird zunächst der Index und seine Verwendung erklärt.

Ein Index kann für mindestens eine Spalte einer Datenbanktabelle mit dem Ziel angelegt werden, Suchen in den Spalten zu beschleunigen. Er besitzt eine eigenständige Datenstruktur, die in einem eigenen Speicherbereich getrennt von der Tabelle abgelegt ist. Beim Anlegen des Indexes werden die Werte der Spalte nicht geändert, sie werden lediglich für die Indexerstellung verwendet. Der Index beinhaltet redundante Informationen, die auf die Tabellendaten referenzieren. Die Datenstruktur des Index hält die Informationen so vor, dass sie schnell durchsucht werden können. Dabei gleicht die Suche in einem Index der Suche in dem Stichwortverzeichnis eines Buches: Das Stichwortverzeichnis ist im hinteren Teil des Buches auf einigen Seiten zu finden (eigener Speicherplatz), besteht aus redundanten Wörtern (Information) mit der Seitenzahl (Referenz) auf die eigentlichen Seiten (Tabellendaten).<sup>14</sup> Würde das Stichwortverzeichnis nicht existieren, müsste das komplette Buch durchblättert werden, um alle Vorkommnisse eines bestimmten Wortes zu finden. Genauso müsste die

---

<sup>13</sup> Vgl. PostgreSQL Usergroup Germany, Warum PostgreSQL benutzen?,  
<http://www.pgug.de/de/informationen/warum-postgresql-benutzen.html>, 05.05.2017

<sup>14</sup> Vgl. Connolly, Begg, Strachan, 2002, S.1277

Datenbank nach der Suche eines bestimmten Wertes in einer Spalte ohne Index die kompletten Tupel durchlaufen, um zu prüfen, ob sie dem Suchkriterium entsprechen. Dieser Vorgang wird als „Sequential-Scan“ bezeichnet. Er wird durch den Einsatz von Indizes vermieden, stattdessen wird ein „Index-Scan“ die gewünschten Informationen schnell finden. Ein Unterschied zum Stichwortverzeichnis ist, dass der Index ständigen Änderungen unterliegt. Bei den Befehlen der Datenmanipulation werden neue Tupel hinzugefügt (INSERT), geändert (UPDATE) und gelöscht (DELETE). In diesen Fällen muss der Index ebenfalls aktualisiert werden. Eine Optimierung der Lesezugriffe kann sich daher negativ auf Schreibzugriffe (oder Änderungen) auswirken, weshalb der Einsatz der Indizes nicht willkürlich erfolgen sollte.<sup>15</sup>

---

15 Vgl. Winand, 2012, S.1-2

## 2.5 Erläuterung der verschiedenen Indextypen

Im folgenden Abschnitt werden die Indizes vorgestellt, die im Rahmen der Performanceuntersuchung zum Einsatz kommen. Beginnend mit dem Standard-Index „B-Baum“ wird danach auf die Index-Frameworks „GIN“ und „GiST“ eingegangen, die Abwandlungen des B-Baumes sind. Sie werden in Kombination mit Trigrammen und der Volltextsuche verwendet, die ebenfalls in diesem Kapitel erläutert werden.

### 2.5.1 B-Baum-Index

Ausgangspunkt ist eine Tabelle mit vier Spalten, bei der die zweite Spalte Ganzzahlen enthält. Diese Spalte wird indiziert.

#### Die Blattknoten

Bei der Erläuterung wird bei den „Index Blattknoten“ begonnen, die in Abbildung 2 links zu sehen sind.

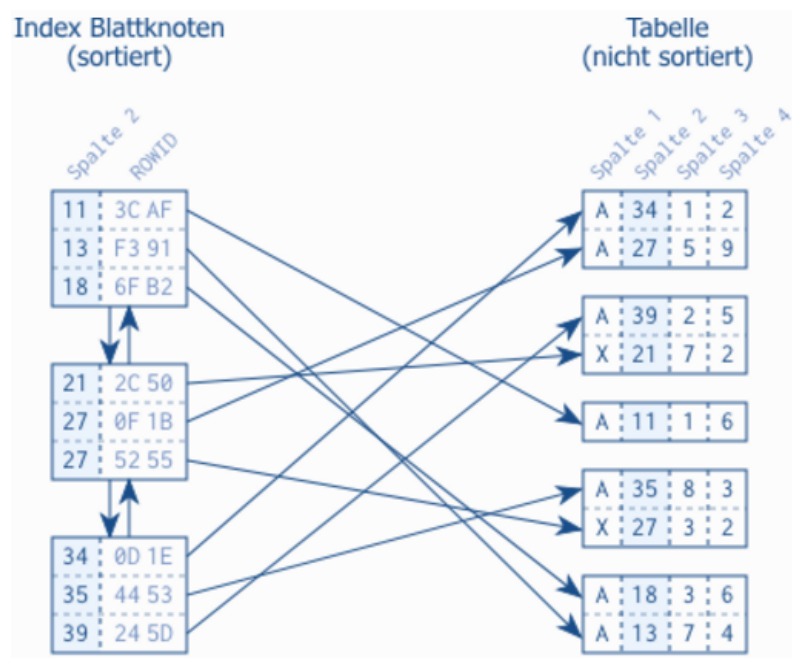


Abbildung 2: Blattknoten mit Referenzen auf die Tabelle<sup>16</sup>

Sie bestehen aus redundanten Werten der indizierten Spalte („Spalte 2“) und einer Referenz (ROWID) auf die Tupel der Tabelle (rechts) in der der entsprechende Wert zu finden ist. Es ist zu erkennen, dass die Werte der Blattknoten im Gegensatz zur Tabelle

<sup>16</sup> Winand, Use the Index, Luke, [http://use-the-index-luke.com/static/fig01\\_01\\_index\\_leaf\\_nodes.de.sURxz1Bl.png](http://use-the-index-luke.com/static/fig01_01_index_leaf_nodes.de.sURxz1Bl.png), 05.05.2017

sortiert sind. Mit den Pfeilen zwischen den einzelnen Blattknoten wird eine verkettete Liste dargestellt, auf die später eingegangen wird. Ziel ist es nun, den gesuchten Eintrag in einem Blattknoten zu finden, um anschließend mithilfe der dazugehörigen ROWID auf den entsprechenden Tupel in der Tabelle zuzugreifen.

### Der Aufbau des B-Baumes

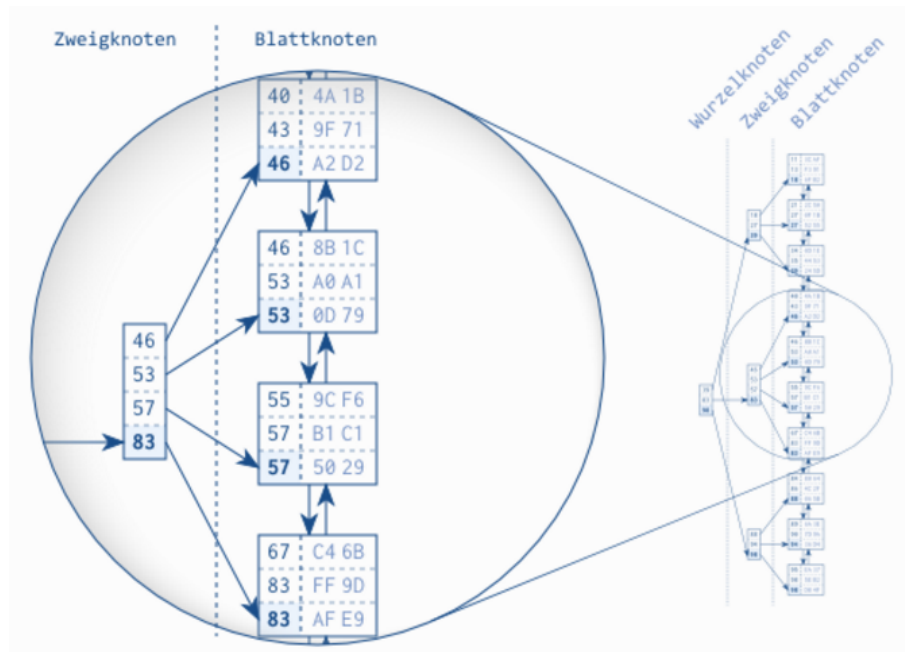


Abbildung 3: Wurzel-, Zweig- und Blattknoten<sup>17</sup>

Bei dem B-Baum gibt es drei Arten von Knoten: den Wurzel-, die Zweig- und die eben erwähnten Blattknoten. Zwischen Wurzel- und Blattknoten kann es mehrere Ebenen von Zweigknoten geben.

Diese Knoten haben eine feste maximale Anzahl an Einträgen. Somit kann ein B-Baum bei einer gegebenen Ebenenanzahl nur eine bestimmte Menge an Einträgen aufnehmen. Bei dem Einfügen neuer Einträge kann es dazu kommen, dass eine neue Ebene von Zweigknoten angelegt wird. Bei dieser Umstrukturierung bleibt der Baum balanciert.

In Abbildung 3 ist hervorgehoben, dass der Zweigknoten die Werte 46, 53, 57 und 83 enthält. Dies sind jeweils die größten Werte der darunterliegenden Knoten. Nach diesem Prinzip ist der Pfad innerhalb des Baumes von der Wurzel bis zum Blattknoten eindeutig festgelegt.

<sup>17</sup> Winand, Use the Index, Luke, [http://use-the-index-luke.com/static/fig01\\_02\\_tree\\_structure.en.LAuPsMu2.png](http://use-the-index-luke.com/static/fig01_02_tree_structure.en.LAuPsMu2.png), 05.05.2017

## Suche innerhalb eines B-Baumes

Der eben erklärte Aufbau ermöglicht eine schnelle Suche innerhalb des B-Baumes. In Abbildung 4 wird anhand der lilafarbenen Linie dargestellt, wie von der Wurzel bis an den Blattknoten gewandert wird, um an den gesuchten Wert 57 zu gelangen.

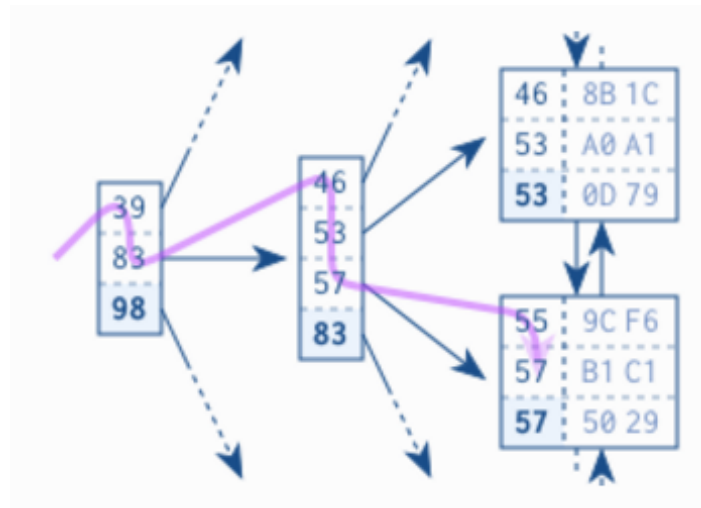


Abbildung 4: Suche nach dem Wert „57“ im B-Baum<sup>18</sup>

Die Einträge eines Knotens werden von oben nach unten mit dem Suchwert verglichen. Es wird der Referenz auf den darunterliegenden Knoten gefolgt, wenn die Bedingung „größer gleich“ Suchwert zutrifft. Beginnend bei dem Wurzelknoten wird im ersten Schritt der Suchwert 57 mit dem Wert des Knotens 39 verglichen. Dieser ist nicht „größer gleich“ dem Suchwert, somit wird der Referenz nicht gefolgt. Daraufhin erfolgt der Vergleich mit dem Wert 83. Dieser Wert erfüllt die Bedingung, da 83 größer als 57 ist, somit wird der Referenz auf den darunterliegenden Knoten gefolgt. Dies wird bis zum Blattknoten weitergeführt.

Beim Blattknoten angelangt, werden dessen Werte ebenfalls auf den Suchwert untersucht. Dabei kann es vorkommen, dass der gesuchte Wert in mehr als einem Blattknoten vorkommt, was auf Duplikate in der indizierten Spalte zurückzuführen ist. Um schnell zu einem benachbarten Blattknoten zu springen, wird die verkettete Liste verwendet. Sie ermöglicht das Durchwandern der Blattknoten in beide Richtungen ohne Nutzung des darüberliegenden Baumes.<sup>19</sup>

<sup>18</sup> Winand, Use the Index, Luke, [http://use-the-index-luke.com/static/fig01\\_03\\_tree\\_traversal.en.RJRAetvl.png](http://use-the-index-luke.com/static/fig01_03_tree_traversal.en.RJRAetvl.png), 05.05.2017

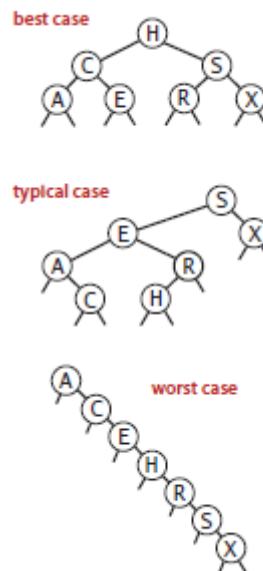
<sup>19</sup> Vgl. Bayer, B-Bäume, [www.bayer.in.tum.de/lehre/WS2001/HSEM-bayer/BTreesAusarbeitung.pdf](http://www.bayer.in.tum.de/lehre/WS2001/HSEM-bayer/BTreesAusarbeitung.pdf), 07.05.2017

Der B-Baum kann neben der hier vorgestellten Abfrage auf Gleichheit auch bei Bereichsabfragen genutzt werden, somit werden SQL-Abfragen mit folgenden Operatoren unterstützt:  $<$ ,  $\leq$ ,  $=$ ,  $\geq$ ,  $>$ .<sup>20</sup>

### Der balancierte Baum und die Auswirkung auf die Suche

Der B-Baum ist ein perfekt balancierter Baum. Robert Sedgewick und Kevin Wayne definieren ihn folgendermaßen: „Ein perfekt balancierter [...] Baum ist ein Baum, dessen null-Referenzen alle die gleiche Entfernung zur Wurzel haben“<sup>21</sup>. Bezogen auf unser Beispiel, handelt es sich bei den null-Referenzen um die Blattknoten.

Um die Vorteile eines balancierten Baumes (u.a. B-Baum) zu verdeutlichen, wird im folgenden Beispiel ein nicht balancierter Baum (Binär Baum) betrachtet und auf die Unterschiede eingegangen:



In Abbildung 5 sind drei verschiedene Binär Bäume zu sehen, die die gleichen Einträge besitzen. Der Unterschied ist, dass diese Einträge in unterschiedlicher Reihenfolge dem Baum hinzugefügt wurden. Es ist zu

Abbildung 5: Case-Szenarien für einen Binär Baum (Sedgewick, Wayne, 2014, S. 432)

sehen, dass im „Worst Case“-Szenario der Binär-Baum zu einer Liste entartet. Nur im „Best Case“ ist er balanciert. Bei einer Suche müssten im „Worst Case“-Szenario alle Knoten durchwandert werden, bis der gesuchte Wert gefunden wird.<sup>22</sup> Dies hat Auswirkung auf die Performance, da die Knotenzugriffe mit Speicher- und Festplattenzugriffen verbunden sind.

Um eine Entartung zu verhindern, wird der B-Baum nach jeder Einfüge-/Aktualisierungs- bzw. Löschoperation in Balance gehalten. Dadurch hat die Reihenfolge des Einfügens der Werte keine Auswirkung auf die Form des Baumes.

Der B-Baum ermöglicht es, mit wenig Knotenzugriffen an die Blattknoten zu gelangen. Somit ist das Durchsuchen einer großen Datenmenge möglich, ohne sie komplett im Hauptspeicher zu halten, die ggf. nicht in diesen passen würde. Beim Suchen müssen

<sup>20</sup> Vgl. Smith, 2010, S.225

<sup>21</sup> Sedgewick, Wayne, 2014, S.453

<sup>22</sup> Vgl. ebda. S.453

jeweils nur die Knoten auf dem Weg zum Blattknoten geladen werden. Such-, Löscho- und Einfügeoperationen werden in logarithmischer Zeit durchgeführt.<sup>23</sup>

### B-Baum-Index in PostgreSQL erstellen

Der B-Baum Index wird mit folgendem Befehl angelegt:

```
CREATE INDEX indexName ON tableName USING btree(column);
```

In Tabelle 1 erfolgt die Erklärung der einzelnen Elemente des Befehls:

| Wert      | Erläuterung   |
|-----------|---|
| indexName | Der Name des Index. Kann frei gewählt werden. Innerhalb der Relationen muss dieser jedoch eindeutig sein. |
| tableName | Der Name der Tabelle, für die der Index angelegt werden soll.   |
| column    | Die Spalte der angegebenen Tabelle, für die der Index angelegt werden soll.                               |

Tabelle 1: Erläuterung der B-Baum Indexerstellung

### Funktionsbasierter Index (FBI)

Wird eine Spalte mit einem B-Baum-Index indiziert, die Zeichenketten speichert, können nur Abfragen unter Beachtung der Groß- und Kleinschreibung, sogenannte „Case Sensitive“-Abfragen, ausgeführt werden. Der Grund liegt darin, dass der B-Baum nach den Anfangsbuchstaben der Wörter aufgebaut ist. Wird ein Wort mit Kleinbuchstaben in den B-Baum eingetragen, kann dieser nicht mit Großbuchstaben gefunden werden.

Oft ist es jedoch gewünscht, die Groß- und Kleinschreibung zu ignorieren und stattdessen eine „Case Insensitive“-Abfrage auszuführen.

Eine Möglichkeit dies zu realisieren ist, bei der Erstellung des Indexes alle Einträge vor dem Einfügen in den B-Baum in Kleinbuchstaben umzuwandeln. Dies kann mit der LOWER-Funktion realisiert werden. Da bei der Erstellung des Index eine Funktion verwendet wird, wird dieser Index „funktionsbasierter Index“ (im Folgendem: FBI) genannt.<sup>24</sup> Der Befehl für die Indexerstellung ist folgender:

```
CREATE INDEX indexName ON tableName(LOWER(column));
```

<sup>23</sup> Vgl. ebda. S.915

<sup>24</sup> Vgl. FH Köln, Funktionsbasierter Index, [http://wikis.gm.fh-koeln.de/wiki\\_db/Datenbanken/Funktionsbasierter-Index](http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Funktionsbasierter-Index), 20.05.2017

Bei der Definition der Abfrage auf die indizierte Spalte muss die gesuchte Zeichenkette ebenfalls mit der LOWER-Funktion in Kleinbuchstaben umgewandelt werden. Derartige Abfragen werden u.a. im Kapitel 4.2.1 behandelt.

## 2.5.2 GIN-Index

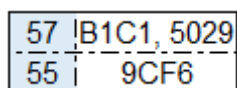
Der GIN (Generalized Inverted Index) erweitert den B-Baum in zwei Richtungen. Er ist ein Framework, mit dem Indizes für benutzerdefinierte Datentypen mit passenden Abfragetypen erstellt werden können. Über zur Verfügung stehende Schnittstellen müssen hierfür, entsprechend des jeweiligen Datentyps, Methoden implementiert werden, über die festgelegt wird, wie mit dem Datentyp z.B. bei einer Suche umgegangen wird. Der B-Baum hingegen arbeitet mit vordefinierten Vergleichsoperatoren. Weiterhin wird der Umgang mit mehrfach vorkommenden Schlüsseln mit dem GIN-Index effizienter umgesetzt.<sup>25</sup>

### Abgrenzung zum B-Baum

Im eben erklärten B-Baum wird ein Beispiel gezeigt, in dem ein Schlüssel mehrfach vorkommt. Um alle Schlüssel zu erhalten, ist es notwendig, die Suche nach dem Finden des ersten Schlüssels weiter durchzuführen. Sie erstreckt sich auch auf die nachfolgenden Schlüssel und eventuell auch auf die folgenden Blattknoten, bis ein Schlüssel mit einem anderen Wert gefunden wird. Hier besteht ein Unterschied zum GIN-Index, der ein invertierter Index ist.

### Der invertierte Index

„Inverted“ bezieht sich auf die Art, wie die Adressen gespeichert werden. Wird ein Schlüssel das erste Mal dem Index hinzugefügt, erfolgt die Speicherung wie gewohnt in einem Blattknoten. Der Unterschied bei einem „Inverted Index“ ist, dass in einem Blattknoten jetzt nicht mehr nur die Adresse eines Tupels steht, sondern eine Liste von



|    |            |
|----|------------|
| 57 | B1C1, 5029 |
| 55 | 9CF6       |

Abbildung 6: GIN-Index  
Blattknoten mit doppelten  
Werten

Adressen, die sogenannte Posting List.<sup>26</sup> Kommt es nun dazu, dass ein bereits vorhandener Schlüssel hinzugefügt wird, wird lediglich die Posting List um die Adresse des entsprechenden Tupels erweitert. Dabei wird kein neuer Schlüssel im Blattknoten eingetragen. Es kann vorkommen, dass eine Adresse in mehreren Posting Lists vorkommt. In Abbildung 4 kommt der

<sup>25</sup> Vgl. PostgreSQL, PostgreSQL 9.6.3 Documentation / 63.1 Introduction, <https://www.postgresql.org/docs/9.6/static/gin-intro.html>, 15.05.2017

<sup>26</sup> Vgl. Sigaev, Bartunov, GIN, <http://www.sigaev.ru/gin/Gin.pdf>, 15.05.2017



Wert 57 zweimal im Blattknoten des B-Baumes vor, wie dies im GIN-Index aussieht, ist in Abbildung 6 dargestellt.

Durch diese Art der Speicherung wird Speicherplatz gespart. Der GIN-Index eignet sich somit für die Speicherung von Daten, in denen eine hohe Anzahl an doppelten Werten vorkommt.

### 2.5.3 GiST-Index

Der GiST (Generalized Search Tree) ist wie der B-Baum eine balancierte Baum-Struktur und wie der GIN-Index ein Framework, mit dem maßgeschneiderte Indizes mit passenden Abfragen implementiert werden können.<sup>27</sup>

Die Knoten des Indexes enthalten (p, ptr)-Paare. Dabei entspricht „p“ dem Element, mit dem der Index aufgebaut wird. Die Elemente sind als benutzerdefinierte Klasse implementiert und repräsentieren eine Eigenschaft, die für alle Datenelemente der darunterliegenden Knoten gelten.<sup>28</sup> Eigenschaften könnten dabei sein: „Alle Daten unter diesem Knoten sind größer 15 und kleiner 25“ oder „Alle Daten unter diesem Knoten liegen in Bonn“.

Der „ptr“ zeigt dabei auf die Datensätze in der Tabelle oder auf Zweigknoten.

Der GiST-Index kann für Abfragen genutzt werden, die mit dem B-Baum-Index nicht realisiert werden können. Darunter fallen Abfragen wie:

- den kürzesten Weg zwischen zwei Punkten finden
- Überprüfen, welche Datums-Zeiträume sich überlappen
- Überprüfen, welches Wort einem bestimmten Wort am ähnlichsten ist.

### 2.5.4 Trigramm-Index

Der Trigramm-Index kombiniert die Zerlegung von Zeichenketten in sogenannte Trigramme mit den eben vorgestellten GIN- und GiST-Indizes. Dabei werden die Indizes mit den Trigrammen befüllt, wodurch sich neue Abfragemöglichkeiten ergeben. Beispielhaft wird die Indexerstellung mit dem GIN-Index demonstriert.

---

27 Vgl. UC BERKELEY DATABASE GROUP, Generalized Search Trees for Database Systems, <http://db.cs.berkeley.edu/papers/vldb95-gist.pdf>, 04.06.2017

28 Vgl. Berkeley University of California, GiST: A Generalized Search Tree for Secondary Storage, <http://gist.cs.berkeley.edu/gist1.html>, 04.06.2017

## Der Trigramm

Bei Trigrammen erfolgt die Zerteilung in Sequenzen dreier aufeinanderfolgender Zeichen einer Zeichenkette.

Abbildung 7 zeigt die Erstellung von Trigrammen mit der Funktion `show_trgm` anhand des Worts „PostgreSQL“. Am Ergebnis der Funktion ist neben der Ausgabe in Kleinbuchstaben zu erkennen, dass vor dem Bilden der Trigramme der übergebenen Zeichenkette zwei Leerzeichen am Anfang sowie eines am Ende hinzugefügt werden.

```
testdb=# SELECT show_trgm('PostgreSQL');
          show_trgm
-----
{"  p"," po","esq,gre,ost,pos,"ql ","res,sql,stg,tgr}
(1 Zeile)
```

Abbildung 7: Trigramm Erstellung mit der Funktion `show_trgm()`

Die Funktion `show_trgm` ist eine von mehreren des Trigramm-Moduls, die als Erweiterung mit dem Befehl `create extension "pg_trgm";` hinzugefügt werden kann.

## Funktionsweise von Trigramm-Indizes

Im Folgenden wird die Funktionsweise der Trigramme in Kombination mit dem GIN-Index erläutert.

Der Unterschied zu dem schon vorgestellten B-Baum sind die anders implementierten Blattknoten, auf die jetzt genauer eingegangen wird.

Als Beispiel dient eine Tabelle (Abbildung 8), die aus den Spalten `ctid`, `firstname` (Vorname) und `lastname` (Nachname) besteht. Die Spalte „`ctid`“ kann als eindeutige Adresse der Tupel aufgefasst werden, die im Kapitel 2.6.1 näher erläutert wird. Für die Spalte „`firstname`“ wird ein Trigramm (GIN)-Index erstellt.

## Indexaufbau

Für jedes Tupel der Tabelle werden aus den Werten der Spalte „`firstname`“ Trigramme gebildet. Zu jedem Trigramm werden die Adressen der Tupel gespeichert, in denen es vorkommt. Daraus ergeben sich die in Abbildung 9 dargestellten Zuordnungen. Als Beispiel

| ctid       | firstname | lastname |
|------------|-----------|----------|
| (0,1)      | Andreas   | Kirchner |
| (0,2)      | Andrea    | Colak    |
| (0,3)      | Anemarie  | Poellath |
| (3 Zeilen) |           |          |

Abbildung 8: Tabelle mit den Spalten `ctid`, `firstname`, `lastname`

wird das Trigramm „and“ genommen. Dieses Trigramm kommt in den Vornamen „Andreas“ und „Andrea“ vor, entsprechend enthält die Posting List die Adresse (0,1) und (0,2).

### Suche innerhalb des Indexes

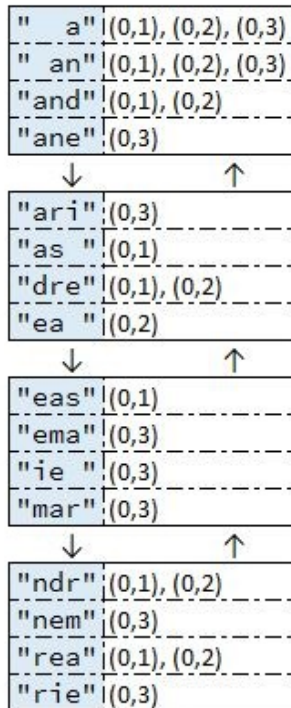


Abbildung 9: Trigramme der Spalte firstname

In der Spalte „firstname“ soll nun, mithilfe des gerade erstellten Indizes, nach „Andrea“ gesucht werden. Hierzu werden zunächst aus dem Suchwort ebenfalls die Trigramme gebildet:

" a", " an", "and", "dre", "ea ", "ndr", "rea"

Im darauffolgenden Schritt wird über diese Trigramme iteriert und für jedes Trigramm wird im Index nachgeschaut, in welchen Adressen dieses vorkommt. Diese Adressen werden sich gemerkt. In jedem weiteren Iterationsschritt bleiben nur die Zeilennummer übrig, die eine gemeinsame Schnittmenge mit den gemerkten Adressen haben. Bei jedem Iterationsschritt wird jetzt die Schnittmenge zwischen den gemerkten Adressen und den an dem Trigramm angehängten Adressen gebildet. Diese Schnittmenge ist die neue gemerkte Menge für den nächsten Iterationsschritt.

Für die ersten beiden Iterationsschritte würden für die Trigramme " a" und " an" die Adressen (0,1), (0,2) und (0,3) gemerkt werden.

Im dritten Iterationsschritt fällt die Adresse (0,3) weg, da der Trigramm "and" nur in den Adressen (0,1) und (0,2) enthalten ist. Dieses wird bis zum letzten Trigramm durchgeführt. Die am Ende übrig gebliebenen Adressen enthalten alle Trigramme des Suchwortes. In unserem Beispiel ist es die Adresse (0,1).

### Trigramm Indexerstellung in PostgreSQL

Befehl für die Trigramm GIN-Index Erstellung:

```
CREATE INDEX indexName ON tableName USING gin(column gin_trgm_ops);
```

Befehl für die GiST-Index Erstellung:

```
CREATE INDEX indexName ON tableName USING gist(column  
gist_trgm_ops);
```

Mit dem übergebenen Parameter „gist\_trgm\_ops“ wird dem GIN- und GiST-Framework die Handhabung für Trigramme übermittelt.

## 2.5.5 Die Volltextsuche und ihre Indizierungsmöglichkeiten

Die Volltextsuche (Full Text Search oder Text Search) bietet die Möglichkeit, innerhalb von Dokumenten linguistische Suchvorgänge auszuführen. Darunter wird die Normalisierung unterschiedlicher Formen eines Wortes verstanden, wodurch auch grammatikalische Abwandlungen des Wortes gefunden werden können.

Ein Dokument bildet dabei die Menge bzw. die Datenbasis, in der gesucht wird. Ein Dokument kann z.B. aus einer oder mehreren Spalten einer oder mehreren Tabellen gebildet werden.

Für die Suche werden die Dokumente aufbereitet und können daraufhin indiziert werden. Bei der Aufbereitung des Dokumentes werden folgende Punkte abgearbeitet:

- **Entfernung von Stoppwörtern**

Hierbei geht es darum, die Wörter aus dem Dokument zu entfernen, die für die Suche keine Relevanz haben. Sie kommen oft vor und können somit nicht zur Eingrenzung beitragen.

Darunter fallen Wörter wie: der, die, das, und, oder.

Ein Nebeneffekt, der durch die Entfernung der Stoppwörter entsteht, ist die Minimierung der Indexgröße.

- **Stemming**

Beim Stemming werden die Wörter bis in die kleinste morphologische Einheit, in sogenannte „Lexeme“ gebracht, wodurch die Normalisierung erfolgt. Werden auch die Suchwörter normalisiert, können Wörter gefunden werden, die in einer anderen Form vorkommen. Die Lexeme sind im Gegensatz zu den Stoppwörtern relevant für die Suche.

- **Ignorierung von Groß- und Kleinschreibung**

Damit bei der Suche die Wörter "Index" und "index" einen Treffer ergeben, wird die Groß- und Kleinschreibung ignoriert. Hierfür werden die Wörter in Kleinbuchstaben gespeichert.<sup>29</sup>

Für die Realisierung der Volltextsuche gibt es in PostgreSQL zwei Datentypen: `tsvector` und `tsquery`, wobei „ts“ für Text Search steht.

In diesem Kapitel wird der `tsvector` behandelt. Der Datentyp `tsquery` wird im Kapitel 2.7.5 erklärt, in dem es um die Definition von Abfragen geht. Mit den zuvor erklärten Indextypen GIN und GiST ist es möglich, den Datentyp `tsvector` zu indizieren.

### **Der Datentyp „TSVECTOR“**

Der Datentyp `tsvector` repräsentiert ein aufbereitetes Dokument und kann mit der Funktion `to_tsvector` erzeugt werden. Dabei wird das Dokument mithilfe von Parsern in sogenannte Token zerlegt. Die erzeugten Token werden jeweils einem Typ zugeordnet, z.B. Wort oder Zahl. In Abhängigkeit der gefundenen Typen wird auf Wörterbücher zugegriffen, um die Token zu normalisieren oder als Stoppwort zu verwerfen.<sup>30</sup>

Nach einer erfolgreichen Normalisierung entsteht ein Lexem. Der `tsvector` ist eine Liste, die diese Lexeme sortiert speichert. Dabei können Wörter, aber auch z.B. URL's normalisiert werden.

### **Konfigurationen**

Wenn es darum geht, Stoppwörter zu entfernen und Lexeme zu bilden, sind Kenntnisse über die Sprache notwendig, in der die Daten vorliegen. Dafür stehen unterschiedliche Konfigurationen zur Verfügung, die unterschiedliche Parser sowie Wörterbücher enthalten, die bei der Erstellung vom `tsvector` verwendet werden können. Zudem besteht die Möglichkeit, eigene Konfigurationen zu erstellen. Dabei können bestehende Parser und Wörterbücher unterschiedlich kombiniert und auch selbst erstellt werden.<sup>31</sup>

---

29 Vgl. PostgreSQL, PostgreSQL 9.6.3 Documentation / 12.1 Introduction,  
<https://www.postgresql.org/docs/9.6/static/textsearch-intro.html>, 13.05.2017

30 Vgl. PostgreSQL, PostgreSQL 9.6.3 Documentation / 12.5. Parsers,  
<https://www.postgresql.org/docs/9.6/static/textsearch-parsers.html>, 13.05.2017

31 Vgl. PostgreSQL, PostgreSQL 9.6.3 Documentation / 12.1.3 Configurations,  
<https://www.postgresql.org/docs/9.6/static/textsearch-intro.html>, 13.05.2017

## Erstellung eines tsvector

In Abbildung 10 wird die Erstellung eines tsvector gezeigt. Dabei wird die Konfiguration für die deutsche Sprache benutzt („german“) und als erstes Argument der Funktion `to_tsvector` übergeben. Als Dokument dient die Zeichenkette „Die Indizierung von Spalten macht Spaß“:

```
testdb=# SELECT to_tsvector('german','Die Indizierung von Spalten macht Spaß');
           to_tsvector
-----
'indizier':2 'macht':5 'spalt':4 'spaß':6
(1 Zeile)
```

Abbildung 10: Erzeugung eines tsvector

An der Ausgabe ist die Aufbereitung des übergebenen Dokuments zu erkennen. Hinter den ausgegebenen Wörtern sind Zahlen aufgeführt. Sie geben an, an welcher Stelle des Dokuments sie vorkommen. Dabei fällt auf, dass die Zahlenreihe lückenhaft ist. Dies liegt an den entfernten Stoppwörtern. Weiterhin fällt auf, dass die Sortierung innerhalb des tsvector nicht nach der Position im Dokument erfolgt, sondern alphabetisch. Die Groß- und Kleinschreibung wird ignoriert, die Ausgabe erfolgt in Kleinbuchstaben. Die Auswirkung des Stemming ist an den Wörtern „Indizierung“ und „Spalten“ zu erkennen, sie wurden zu „indizier“ und „spalt“ normalisiert. Bei den Token des übergebenen Dokuments handelte es sich um den Typ: „Word, all ASCII letters“.

## ts\_debug – Funktion

Um Details über die Arbeit des Parsers zu erfahren, gibt die Funktion `ts_debug` hierzu Auskunft.<sup>32</sup> In Abbildung 11 ist die Ausgabe der Funktion zu sehen.

```
testdb=# SELECT * FROM ts_debug('german','Die Indizierung von Spalten macht Spaß');
 alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
asciiword | Word, all ASCII | Die | {german_stem} | german_stem | {}
blank | Space symbols |  | {} |  | 
asciiword | Word, all ASCII | Indizierung | {german_stem} | german_stem | {indizier}
blank | Space symbols |  | {} |  | 
asciiword | Word, all ASCII | von | {german_stem} | german_stem | {}
blank | Space symbols |  | {} |  | 
asciiword | Word, all ASCII | Spalten | {german_stem} | german_stem | {spalt}
blank | Space symbols |  | {} |  | 
asciiword | Word, all ASCII | macht | {german_stem} | german_stem | {macht}
blank | Space symbols |  | {} |  | 
word | word, all letters | Spaß | {german_stem} | german_stem | {spaß}
(11 Zeilen)
```

Abbildung 11: Ausgabe der Funktion „ts\_debug“

32 Vgl. PostgreSQL, PostgreSQL 9.6.3 Documentation / 12.8. Testing and Debugging Text Search, <https://www.postgresql.org/docs/9.6/static/textsearch-debugging.html>, 13.05.2017

In Tabelle 2 erfolgt die Erläuterung der einzelnen Spalten der Ausgabe:

| Spalte       | Beschreibung   |
|--------------|--|
| alias        | Kurzname des Token-Typs  |
| description  | Beschreibung des Token-Typs  |
| token        | Der gefundene Token  |
| dictionaries | Wörterbücher, die für die Bearbeitung des Typs herangezogen wurden.              |
| dictionary   | Wörterbuch, das den Token zuordnen konnte.                                       |
| lexemes      | Das Lexem, das aus dem Wörterbuch erstellt wurde, das den Token zuordnen konnte. |

Tabelle 2: Erläuterung der "ts\_debug()" Funktion

## Indexerstellung für die Volltextsuche in PostgreSQL

Es existieren GIN und GiST Implementierungen für den „tsvector“.<sup>33</sup>

Befehl für die GIN-Index Erstellung:

```
CREATE      INDEX      indexName      ON      tableName      USING  
gin(to_tsvector('german', column));
```

Befehl für die GiST-Index Erstellung:

```
CREATE      INDEX      indexName      ON      tableName      USING  
gist(to_tsvector('german', column));
```

---

<sup>33</sup> Vgl. Bartunov, Sigaev, Full-Text Search in PostgreSQL, <https://www.pgcon.org/2007/schedule/attachments/12-fts.pdf>, 13.05.2017

## 2.6 Tabellen und Indizes bei PostgreSQL

In diesem Kapitel wird auf grundlegende Informationen zu PostgreSQL eingegangen. Darunter fällt, wie und wo Tabellen gespeichert werden sowie die Berechnung der Tabellengröße. Zudem wird die Auswirkung von Indizes auf Abfragen im Hinblick auf die Tabellenzugriffe analysiert.

Dies erfolgt mithilfe von Statistiken, die von PostgreSQL gesammelt und über System-Tabellen und Views zur Verfügung gestellt werden.

System-Kataloge sind gewöhnliche Tabellen, die auch editiert werden können. Davon ist abzuraten, da das System dadurch beschädigt werden kann. Für Änderungen an den Tabellen sollte auf Funktionen zurückgegriffen werden, die ebenfalls von PostgreSQL zur Verfügung gestellt werden.

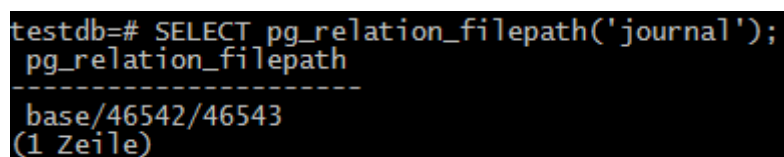
Die Abfragen und Statistiken beziehen sich auf die Testtabelle „journal“. Sie bildet die Grundlage der Performancemessungen und wird in Kapitel 3.1 näher erläutert.

### 2.6.1 Speicherorganisation in PostgreSQL

Jede Relation (Tabelle, Index) wird in einer eigenen Datei auf der Festplatte gespeichert. Um herauszufinden, wo diese Datei auf dem Datenträger zu finden ist, steht folgende Funktion zur Verfügung:

```
SELECT pg_relation_filepath('journal');
```

Als Parameter dient der Tabellenname (journal). In Abbildung 12 ist das Ergebnis der Funktion zu sehen.



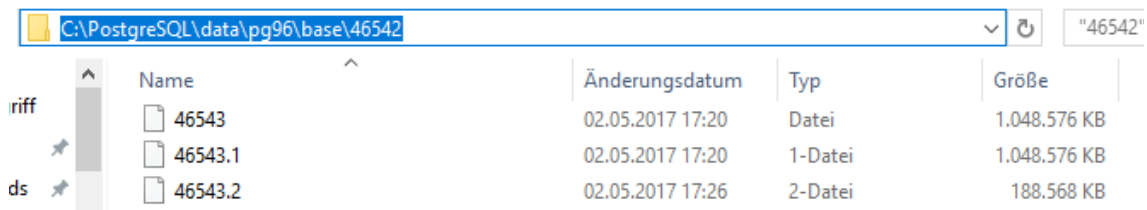
```
testdb=# SELECT pg_relation_filepath('journal');
pg_relation_filepath
-----
base/46542/46543
(1 Zeile)
```

Abbildung 12: Pfad der Tabelle „journal“

Der angegebene Pfad „base/12401/43547“ befindet sich unterhalb des Installationsverzeichnis von PostgreSQL.

In Abbildung 13 sind sowohl der vollständige Pfad als auch die Dateien der Tabelle „journal“ im Explorer zu sehen.





| Name    | Änderungsdatum   | Typ     | Größe        |
|---------|------------------|---------|--------------|
| 46543   | 02.05.2017 17:20 | Datei   | 1.048.576 KB |
| 46543.1 | 02.05.2017 17:20 | 1-Datei | 1.048.576 KB |
| 46543.2 | 02.05.2017 17:26 | 2-Datei | 188.568 KB   |

Abbildung 13: Speicherung der Tabelle auf dem Datenträger

Um herauszufinden, wo ein Index gespeichert ist, muss der Funktion `pg_relation_filepath` der Name des Index übergeben werden.

### Speicherorganisation

In den Dateien erfolgt die Speicherung der Tupel der Tabelle. Die Speicherung erfolgt in Segmenten, die Pages genannt werden und jeweils 8 KB groß sind. Nach der Erstellung der Tabelle beträgt die Größe der Datei 0 KB. Sobald ein Tupel hinzugefügt wird, wächst die Datei auf 8 KB an. Die Datei wächst um eine weitere Page, wenn die vorherige keine Tupel mehr aufnehmen kann. Eine Tabelle wächst demnach in 8 KB Schritten.

Eine Datei kann maximal 1 GB groß werden. Wird diese Größe überschritten, wird eine neue Datei erzeugt. Aus diesem Grund sind in Abbildung 13 drei Dateien mit dem Namen 43547 enthalten, die sich nur im Suffix unterscheiden. Dieses wird bei jedem weiteren GB inkrementiert.

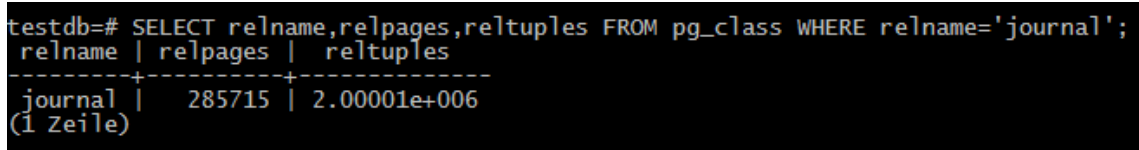
Zum Indexaufbau sei an dieser Stelle erwähnt, dass die einzelnen Knoten des B-Baumes jeweils eine Page groß sind.<sup>34</sup>

<sup>34</sup> Vgl. Belaid, Introduction to PostgreSQL physical storage, <http://rachbelaid.com/introduction-to-postgres-physical-storage/>, 13.05.2017

## Anzahl der Pages und Speichergröße einer Tabelle ermitteln

Über den System-Katalog `pg_class` kann herausgefunden werden, aus wie vielen Pages die „journal“- Tabelle aktuell besteht.<sup>35</sup>

```
SELECT relname, relpages, reltuples FROM pg_class WHERE  
relname='journal';
```



```
testdb=# SELECT relname,relpages,reltuples FROM pg_class WHERE relname='journal';  
 relname | relpages | reltuples  
-----+-----+-----  
 journal |    285715 | 2.00001e+006  
(1 Zeile)
```

Abbildung 14: Ermittlung der Tupel- und Pageanzahl für die Tabelle *journal*

An der Spalte „relpages“ in Abbildung 14 ist abzulesen, dass die Tabelle *journal* aus 285715 Pages besteht. Insgesamt sind in diesen Pages 2 Million Tupel gespeichert. Dies entspricht ca. 6 Tupel je Page.

Da die Größe einer Page bekannt ist, kann mit den Angaben auch die Größe der Tabelle berechnet werden:

285715 Pages \* 8 KB = 2.285.720 KB

Dies entspricht genau der Summe der Dateigrößen aus Abbildung 13.

## CTID Spalte

Es kann nun die Spalte „ctid“ aus Abbildung 8 näher erläutert werden. Der erste Tupel enthält den Wert (0,1). Der erste Wert (0) sagt aus, in welcher Page innerhalb der Datei der Tupel zu finden ist. Der zweite gibt an, an welcher Position innerhalb der Page er sich befindet, wobei die Aufzählung der Page mit Null und die Position innerhalb der Page mit Eins beginnt. Somit ist der erste Tupel innerhalb der Datei in der ersten Page zu finden.

Die ctid eines Tupels ist tabellenweit eindeutig und kann für jede Tabelle ermittelt werden. Dafür muss sie in der SELECT-Anweisung explizit angegeben werden. Bsp.:

```
SELECT ctid,* FROM journal;
```

<sup>35</sup> Vgl. Smith, 2010, S. 212

## 2.6.2 Abfragestatistik

Im Folgenden wird auf Statistiken von Abfragen eingegangen. An ihnen kann unter anderem abgelesen werden, wie viele Pages geladen werden mussten, um eine Abfrage zu beantworten. An diesen Werten wird die Auswirkung von Indizes veranschaulicht.

Hierfür wird die View „table\_stats“ erstellt:

```
CREATE OR REPLACE VIEW table_stats AS
SELECT
    stat.relname AS relname,
    seq_scan, seq_tup_read, idx_scan, idx_tup_fetch,
    heap_blks_read, heap_blks_hit, idx_blks_read, idx_blks_hit
FROM
    pg_stat_user_tables stat
    RIGHT JOIN pg_statio_user_tables statio
        ON stat.relid=statio.relid;36
```

Die erzeugte View greift auf zwei System-Views zu, die in Tabelle 3 erläutert werden:

| View-Name             | Erläuterung   |
|-----------------------|---|
| pg_stat_user_tables   | Die View bietet Statistiken über Zugriffe auf vom Benutzer angelegten Tabellen. Systemtabellen werden hierbei ausgeschlossen. |
| pg_statio_user_tables | Die View bietet Statistiken über I/O Zugriffe auf vom Benutzer angelegten Tabellen. Systemtabellen werden ausgeschlossen.     |

Tabelle 3: System-Views „pg\_stat\_user\_tables“, „pg\_statio\_user\_tables“<sup>37</sup>

<sup>36</sup> Vgl. Smith, 2010, S.210

<sup>37</sup> Vgl. PostgreSQL, PostgreSQL 9.6.3 Documentation / 28.2 The Statistics Collector, <https://www.postgresql.org/docs/9.6/static/monitoring-stats.html>, 13.05.2017

In Tabelle 4 erfolgt die Erläuterung der selektierten Spalten:

| Spaltenname    | Erläuterung   |
|----------------|---|
| relname        | Name der Relation, auf die sich die Statistiken beziehen.                   |
| seq_scan       | Anzahl der Sequential-Scans, die auf der Tabelle ausgeführt wurden.         |
| seq_tup_read   | Anzahl der Tupel, die bei Sequential-Scans geladen wurden.                  |
| idx_scan       | Anzahl der Index-Scans, die auf der Tabelle ausgeführt wurden.              |
| idx_tup_fetch  | Anzahl der Tupel, die nach einem Index-Scan aus der Tabelle geladen wurden. |
| heap_blks_read | Anzahl der Tabellen-Pages, die von der Festplatte geladen wurden.           |
| heap_blks_hit  | Anzahl der Tabellen-Pages, die aus dem Cache geladen wurden.                |
| idx_blks_read  | Anzahl der Index-Pages, die von der Festplatte geladen wurden.              |
| idx_blks_hit   | Anzahl der Index-Pages, die aus dem Cache geladen wurden.                   |

Tabelle 4: Spaltenerläuterung der View „table\_stats“<sup>38</sup>

Die Werte beziehen sich auf alle Abfragen einer Tabelle. Um die Werte nur auf die zuletzt ausgeführte Abfrage zu erhalten, müssen diese vorher zurückgesetzt werden. Dies erfolgt mit folgender Funktion `SELECT pg_stat_reset();`

---

38 Vgl. ebda., 13.05.2017

## Abfrage ohne Index

Mit einer Abfrage wird nun nach den Tupel in der Tabelle „journal“ gesucht, die in der Spalte „documenttext“ den Wert „RECHNUNG 6127675“ haben:

```
SELECT * FROM journal WHERE documenttext = 'RECHNUNG 6127675';
```

Das Ergebnis der Abfrage ist in Abbildung 15 zu sehen.

| id<br>bigint | personalaccountname<br>character varying(1000) | documentnumber<br>character varying(1000) | documenttext<br>character varying(1000) | date<br>bigint | type<br>character varying(1000) | amount<br>numeric | openamount<br>numeric | junk<br>character(1000) |
|--------------|--|---|---|----------------|---------------------------------|-------------------|-----------------------|-------------------------|
| 11172        | Volkswagen                                     | 2017-011172                               | RECHNUNG 6127675                        | 413420400588   | RECHNUNG                        | 263.82            | 263.82                | junk                    |
| 388653       | SAP  | 2017-0388653                              | RECHNUNG 6127675                        | 32285005199412 | ZAHLUNG                         | -67.80            | -67.80                | junk                    |

Abbildung 15: Tupel mit dem Wert „RECHNUNG 6127675“ in der Spalte „documenttext“

Nach dem Ausführen der Abfrage wird mithilfe der zuvor erstellten View ein Blick auf die Statistiken der Tabelle geworfen:

```
testdb=# SELECT * FROM table_stats WHERE relname='journal';
```

| relname | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | heap_blks_read | heap_blks_hit | idx_blks_read | idx_blks_hit |
|---------|----------|--------------|----------|---------------|----------------|---------------|---------------|--------------|
| journal | 1        | 2000000      |          |               | 283632         | 2083          |               |              |

(1 Zeile)

Abbildung 16: Abfragestatistik ohne Index

Die Werte können wie folgt interpretiert werden:

Auf der Tabelle „journal“ wurde ein Sequential-Scan ausgeführt (seq\_scan). Bei dem Sequential-Scan wurden 2000000 Tupel aus der Tabelle gelesen (seq\_tup\_read). Dabei handelt es sich um alle Tupel der Tabelle, wie aus Abbildung 14 entnommen werden kann. Für das Lesen der Tupel wurden 283632 Pages von der Festplatte (heap\_blks\_read) und 2083 Pages aus dem Cache (heap\_blks\_hit) geladen. Insgesamt wurden alle 285715 Pages der Tabelle geladen (vgl. Abbildung 14).

## Abfrage mit Index

Nun wird die durchsuchte Spalte (documenttext) mit einem B-Baum-Index indiziert:

```
CREATE INDEX idx_documenttext_btree ON journal USING btree  
(documenttext varchar_pattern_ops);
```

Nach dem Zurücksetzen der Statistiken wird die gleiche Abfrage erneut ausgeführt und anschließend auf die View „table\_stats“ zugegriffen (Abbildung 17).

```
testdb=# SELECT * FROM table_stats WHERE relname='journal';
```

| relname   | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | heap_blks_read | heap_blks_hit | idx_blks_read | idx_blks_hit |
|-----------|----------|--------------|----------|---------------|----------------|---------------|---------------|--------------|
| journal   | 0        | 0            | 1        | 2             | 2              | 0             | 4             | 0            |
| (1 Zeile) |          |              |          |               |                |               |               |              |

Abbildung 17: Abfragestatistik mit Index

Aus den Statistiken ist zu entnehmen, dass auf der Tabelle „journal“ ein Index-Scan (idx\_scan) ausgeführt wurde. Für den Index-Scan wurden 4 Index-Pages von der Festplatte (idx\_blks\_read) und keine aus dem Cache (idx\_blks\_hit) geladen. Nach dem Index-Scan wurden 2 Tupel aus der Tabelle (idx\_tup\_fetch) geholt. Für diesen Tabellenzugriff wurden 2 Pages von der Festplatte (heap\_blks\_read) und keine aus dem Cache (heap\_blks\_hit) geladen.

### Vergleich der Statistiken

Ohne Index musste auf alle 2 Millionen Tupel, demnach auf alle 285715 Pages, der Tabelle zugegriffen werden, um die Abfrage zu beantworten.

Mit einem Index konnten nach dem Indexzugriff, für den lediglich 4 Pages geladen werden mussten, die 2 Treffer gezielt aus der Tabelle geholt werden.

Durch die Statistiken wird ersichtlich, dass durch den Index die Anzahl der zu ladenden Pages deutlich reduziert wird. Dadurch wird die Abfrage enorm beschleunigt.

## 2.7 Möglichkeiten der Teilstringsuche mittels SQL-Abfragen

In diesem Kapitel wird ein Überblick über die Stringmatching Möglichkeiten von SQL-Operatoren gegeben, die in PostgreSQL zur Verfügung stehen. Im weiteren Verlauf werden diese Operatoren auf ihre Eignung für die Zuordnung der Verwendungszwecke von Rechnungen und Zahlungen sowie auf ihre Performance untersucht.

### 2.7.1 Equal (=)

```
SELECT * FROM kontakt WHERE firstname = 'Robert';
```

Das Gleichheitszeichen (=) dient zum Auffinden einer exakt übereinstimmenden Zeichenkette innerhalb einer Spalte. Es ist nicht möglich, Muster zu verwenden. Diese Case Sensitive-Suche bietet die geringste Flexibilität. Im oben aufgeführten Beispiel werden alle Kontakte gefunden, die den Wert „Robert“ in der Spalte „firstname“ haben.

## 2.7.2 LIKE

```
SELECT * FROM kontakt WHERE lastname LIKE 'Mar_us'
```

Mit dem LIKE-Operator ist es möglich, nach einfachen Mustern innerhalb einer Spalte zu suchen. Für die Definition des Musters stehen das Prozentzeichen (%) und der Unterstrich (\_) zur Verfügung. Das Prozentzeichen wird als Platzhalter für kein, ein oder mehrere beliebige Zeichen verwendet. Dagegen steht der Unterstrich für genau ein beliebiges Zeichen. Wird der LIKE-Operator ohne Platzhalter verwendet, verhält er sich wie die Suche mit dem zuvor erklärten Gleichheitszeichen. Der LIKE-Operator ist Case Sensitive.<sup>39</sup>

Beispiele:

| SQL-Anweisung                        | Ergebnis | Erläuterung  |
|--------------------------------------|----------|--|
| SELECT 'Indizierung'<br>LIKE '%ung'  | true     | Findet die Werte, die mit der Zeichenkette ,ung' enden.  |
| SELECT 'Indizierung'<br>LIKE 'Indi%' | true     | Findet Werte, die mit der Zeichenkette ,Indi' beginnen und keine oder beliebig viele weitere Zeichen besitzen. |
| SELECT 'Indizierung'<br>LIKE '%izi%' | true     | Findet Zeichenketten, die die Zeichenkette „izi“ an beliebiger Stelle besitzen.                                |
| SELECT 'Indizierung'<br>LIKE '_e%'   | false    | Findet einen String mit einem e an zweiter Position und ggf. beliebig folgende Zeichen.                        |

Tabelle 5: Erläuterung der Platzhalter des LIKE-Befehls

## 2.7.3 ILIKE

```
SELECT * FROM kontakt WHERE lastname ILIKE 'Mar_us'
```

ILIKE bietet die gleichen Muster wie der LIKE-Operator, mit dem Unterschied, dass bei diesem Operator die Groß- und Kleinschreibung ignoriert wird (Case Insensitive).

<sup>39</sup> Vgl. w3schools, SQL LIKE Operator, [https://www.w3schools.com/sql/sql\\_like.asp](https://www.w3schools.com/sql/sql_like.asp), 21.06.2017

## 2.7.4 Reguläre Ausdrücke

Reguläre Ausdrücke ermöglichen eine sehr flexible Art, um eine Zeichenkette auf ein bestimmtes Muster zu prüfen. Für die Definition des Musters stehen zahlreiche Metazeichen zur Verfügung, die unterschiedlich kombiniert werden können.

Eine Abfrage, die einen regulären Ausdruck enthält, wird mit dem Tilde-Operator (~) ausgeführt. Die Abfrage ist Case Sensitive. Für eine Case Insensitive-Abfrage wird der Operator ~\* verwendet.

Im Folgenden werden einige Muster anhand von Beispielen erklärt.

Muster: '[1-5]'

Das Minuszeichen in der eckigen Klammer definiert einen Bereich von Zeichen. In diesem Fall steht es für genau eine Ziffer zwischen 1 und 5.

Beispiele:

| SQL-Anweisung                            | Ergebnis | Erläuterung  |
|--|----------|--|
| SELECT 'test4test' ~<br>'test[1-5]test'  | true     | Liefert true, da die Zeichenkette dem definierten Muster entspricht. Die Zeichenkette entspricht einer Ziffer zwischen 1 und 5, die von den Wörtern „test“ umgeben ist.  |
| SELECT 'test42test'<br>~ 'test[1-5]test' | false    | Liefert false, da die Zeichenkette nicht dem definierten Muster entspricht. Die Zeichenkette besteht zwar aus Ziffern zwischen 1 und 5, die von den Wörtern „test“ umgeben sind. Im Muster wurde jedoch nur eine einstellige Zahl definiert. |

Tabelle 6: Regulärer Ausdruck mit Bereichsangabe



Der Ausdruck wird nun dahingehend geändert, dass auch mehrere Ziffern zwischen dem Text „test“ vorkommen können:

'[1-5]\*'

Der Ausdruck wird um einen Stern (\*), einem sogenannten „Quantor“, erweitert. Der Stern nach einem Zeichen oder in diesem Fall nach einem Bereich (eckige Klammern) gibt an, dass dieser mehrfach, einmal oder keinmal vorkommen kann.

Beispiele:

| SQL-Anweisung                          | Ergebnis | Erläuterung  |
|--|----------|--|
| SELECT 'test42test' ~ 'test[1-5]*test' | true     | Liefert true, da die Zeichenkette dem definierten Muster entspricht. Es wurde definiert, dass zwischen den Wörtern „test“ mindestens eine oder keine der Ziffern im Bereich zwischen 1 und 5 vorkommen können. |
| SELECT 'testtest' ~ 'test[1-5]*test'   | true     | Liefert true, da die Zeichenkette dem definierten Muster entspricht. Wie eben beschrieben, muss keine Ziffer zwischen 1 und 5 vorkommen.   |

Tabelle 7: Regulärer Ausdruck mit Bereichsangabe und Quantor \* (Stern)

Der Ausdruck wird nun dahingehend geändert, dass eine beliebige Zahl zwischen den Wörtern „test“ vorhanden sein muss:

'[\d]+'

In diesem Ausdruck sind die Ziffern aus der Klammer durch „\d“ ersetzt worden. Bei dem „\d“ handelt es sich um eine vordefinierte Zeichenklasse, die für eine beliebige Ziffer zwischen 0 und 9 steht. Somit werden jetzt nicht nur Ziffern zwischen 1 und 5 gefunden. Zudem wird der Quantor „+“ verwendet, der definiert, dass mindestens eine Ziffer vorkommen muss.

Beispiele:

| SQL-Anweisung                             | Ergebnis | Erläuterung   |
|---|----------|---|
| SELECT 'testtest' ~<br>'test[\\d]+test'   | false    | Liefert false, da die Zeichenkette dem definierten Muster nicht entspricht. Zwischen den Zeichenketten "test" muss mindestens eine Zahl stehen. |
| SELECT 'test99test'<br>~ 'test[\\d]+test' | true     | Liefert true, da die Zeichenkette dem definierten Muster entspricht. Zwischen der Zeichenkette kommt mindestens eine Zahl vor.                  |

Tabelle 8: Regulärer Ausdruck mit Zeichenklasse und Quantor + (Plus)

Die Beispiele geben einen kleinen Einblick in die komplexen Möglichkeiten der regulären Ausdrücke.<sup>40</sup>

## 2.7.5 Volltextsuche

Bei der Volltextsuche wird ein `tsvector` mit einer `tsquery` verglichen. Der `tsvector` wurde im Kapitel 2.5.5 vorgestellt. Nun folgt die Erklärung des `tsquery` Datentyps.

### Der Datentyp „tsquery“

Der Datentyp `tsquery` speichert Lexeme, nach denen in einem `tsvector` gesucht wird. Mit der Funktion `to_tsquery` wird ein `tsquery` Datenobjekt erzeugt. Der Funktion werden Token übergeben, die mit logischen Operatoren verknüpft werden können, um die Suche zu verfeinern. Es stehen folgende Operatoren zur Verfügung: `&` (und), `|` (oder), und `!` (nicht). Bevor die Token im `tsquery` gespeichert werden, erfolgt die Normalisierung, die auch bei der Erstellung des `tsvector`s vorgenommen wurde. Nur durch die Normalisierung auf beiden Seiten (`tsquery` und `tsvector`) können auch Abwandlungen der Token gefunden werden.<sup>41</sup>

<sup>40</sup> Vgl. Sedgewick, Wayne, 2014, S.829-833

<sup>41</sup> Vgl. PostgreSQL, PostgreSQL 9.6.3 Documentation / 8.11 Text Search Types,  
<https://www.postgresql.org/docs/9.6/static/datatype-textsearch.html>, 13.05.2017

## Erstellung einer tsquery

In Abbildung 18 ist die Erstellung eines tsquery zu sehen. Die tsquery definiert eine Suche nach Dokumenten, in denen die Token „Spaß“ und (&) „Spalte“ enthalten sind.

```
testdb=# SELECT to_tsquery('german', 'Spaß&Spalte');
to_tsquery
-----
'spaß' & 'spalt'
(1 Zeile)
```

Abbildung 18: tsquery Erstellung

## Ausführung der Suche

Eine Volltextsuche wird mit dem Operator @@ durchgeführt. Dabei werden die Token der tsquery mit denen eines tsquery verglichen. Ist die Suche erfolgreich, wird true ansonsten false zurückgegeben.

Hierzu ein Beispiel:

```
SELECT to_tsvector('german', 'Die Indizierung von Spalten macht Spaß') @@ to_tsquery('german', 'Spaß&Spalte');
```

Mit der Abfrage wird geprüft, ob die Token „Spaß“ und „Spalte“ in dem Dokument „Die Indizierung der Spalten macht Spaß“ enthalten sind. Die Abfrage liefert „true“ zurück, da beide Wörter im Dokument vorkommen. Obwohl nach dem Wort „Spalte“ gesucht wird, ergibt die Suche einen Treffer. Dies liegt daran, dass die Wörter „Spalten“ und „Spalte“ auf das Lexem „spalt“ normalisiert werden.

## 2.7.6 Trigram-Similarity

Im Kapitel 2.5.4 wird die Indexerstellung anhand von Trigrammen erläutert. Mithilfe des Trigramm-Moduls können auch Abfragen definiert werden, die Zeichenketten auf ihre Ähnlichkeit überprüfen. Die Suche basiert auf der Funktion `word_similarity(text, text)`.

Die Funktion nimmt zwei Zeichenketten entgegen, die miteinander verglichen werden. Der Rückgabewert ist eine Zahl, die angibt, wie ähnlich die im ersten Parameter angegebene Zeichenkette mit dem ähnlichsten Wort der zweiten Zeichenkette ist. Der Rückgabewert ist 0, wenn beide Zeichenketten vollständig unterschiedlich sind und 1,

wenn die erste Zeichenkette mit einem der Wörter der zweiten Zeichenkette identisch ist.<sup>42</sup>

Beispiele:

| SQL-Befehl   | Ergebnis | Erläuterung  |
|--|----------|--|
| SELECT<br>word_similarity('PostgreSQL',<br>'PostgreSQL');        | 1        | Die beiden Zeichenketten sind identisch, somit liefert die Funktion den Wert 1.  |
| SELECT<br>word_similarity('PostgreSQL', 'Datenbank postgresql'); | 1        | Im zweiten Parameter kommt die Zeichenkette des ersten Parameters vor. Trotz Unterschieden in der Groß- und Kleinschreibung liefert die Funktion den Wert 1.   |
| SELECT<br>word_similarity('PostgreSQL', 'Deutsche Post');        | 0.363636 | Die Gleichheit der Zeichenketten „PostgreSQL“ und „Deutsche Post“ wird mit 0.363636 bewertet. Dies entspricht der "word_similarity" von PostgreSQL und dem zweiten Wort der zweiten Zeichenkette „Post“. |

Tabelle 9: Beispiele der Funktion „word\_similarity“

Mit dem Operator <<-> ist es möglich, die „Entfernung“ zweier Zeichenketten zu ermitteln. Diese ist wie folgt definiert:

$1 - \text{word\_similarity}()$

Beispiel:

Die Zeichenkette „PostgreSQL“ und „Deutsche Post“ haben eine word\_similarity von 0.363636. Die Berechnung der „Entfernung“ der beiden Zeichenketten ist in Abbildung 19 zu sehen.

```
testdb=# SELECT 'PostgreSQL' <<-> 'Deutsche Post';
?column?
-----
0.636364
(1 Zeile)
```

Abbildung 19: „Entfernung“ der Wörter 'PostgreSQL' und 'Deutsche Post'

42 Vgl. PostgreSQL, PostgreSQL 9.6.3 Documentation / F.31. pg\_trgm, <https://www.postgresql.org/docs/9.6/static/pgtrgm.html>, 13.05.2017

### 3 Testumgebung / -verfahren

Mit den in Kapitel 2.7 vorgestellten Operatoren werden Abfragen definiert, die auf ihre Zuordnungsmöglichkeiten und Performance untersucht werden.

Bei der Untersuchung der Zuordnungsmöglichkeit geht es darum, passende Abfragen zu formulieren, die die Zuordnung von Rechnungen und Zahlungen über den Verwendungszweck ermöglichen. Die Festlegung der dabei eingesetzten Suchmuster und verwendeten Indizes wird im nachfolgenden Kapitel behandelt.

Bei der Performanceuntersuchung werden die einzelnen Abfragen mit unterschiedlichen Indizes getestet. Es wird untersucht, welche Abfrage mit welchem Index die beste Performance liefert. Dies soll auch bei skalierenden Daten überprüft werden.

Zusätzlich wird die Indexgröße betrachtet. Hier ist zu überprüfen, wie sich die Indexgröße der unterschiedlichen Indextypen bei skalierenden Daten entwickelt.

In diesem Kapitel geht es darum, Testdaten und Testfälle zu erstellen, die den Umstand der Auszifferung realitätsnah abbilden und Antworten auf die Anforderungen der oben genannten Kriterien liefern.

## 3.1 Die Journal-Tabelle

In der Scopevisio-Anwendung werden Zahlungsein- und Ausgänge in einer Tabelle gespeichert. Für die Tests wird die Tabelle „journal“ erstellt, die diese nachbildet. Dabei werden nur die Spalten übernommen, die für das Beispiel der Auszifferung relevant sind. Die Tabelle dient als Grundlage für die Performancemessungen.

### Tabellenaufbau

| Spalte              | Datentyp      | Beschreibung   |
|---------------------|---------------|--|
| id                  | BIGINT        | Primärschlüssel der Tabelle.<br>(Eindeutige Nummer)  |
| type                | VARCHAR(1000) | Typ des Belegs. Über die Spalte wird definiert, ob es sich um eine Rechnung oder eine Zahlung handelt. (Mögliche Werte: „RECHNUNG“ oder „ZAHLUNG“)     |
| personalAccountName | VARCHAR(1000) | Name des Kunden.   |
| documentNumber      | VARCHAR(1000) | Unternehmensinterne Nummer des Dokuments.  |
| documentText        | BIGINT        | Belegtext.   |
| date                | VARCHAR(1000) | Erstellungsdatum des Belegs.   |
| amount              | numeric       | Der Betrag des Belegs.   |
| openAmount          | numeric       | Beim Typ RECHNUNG: offener Betrag der Rechnung<br>Beim Typ ZAHLUNG: noch nicht ausgezifferter Betrag einer Rechnung.                                   |
| junk                | CHAR(1000)    | Die Spalte dient dazu, der Tabelle eine realistische Größe zu geben. Der Datentyp CHAR nimmt im Gegensatz zu VARCHAR den gesamt geforderten Platz ein. |

Tabelle 10: Spaltenerläuterung der Tabelle "journal"

Über die Spalte „documentText“ erfolgt die Zuordnung von Rechnungen und Zahlungen. Bei Tupel vom Typ „RECHNUNG“ ist in der Spalte „documentText“ die

Rechnungsnummer hinterlegt, während bei der Zahlung der Verwendungszweck der Überweisung aufgeführt ist.

## 3.2 Testdatengenerierung

Für die Erstellung der Testdaten wurde ein Programm geschrieben, das diese im CSV-Format ausgibt. Dadurch ist es möglich, die Daten in PostgreSQL zu importieren.

Die Spalten werden mit Zufallswerten gefüllt.

### Format des Verwendungszwecks

Besonderes Augenmerk bei der Testdatengenerierung liegt bei der Erstellung des Verwendungszwecks, da über diesen die Zuordnung von Rechnungen und Zahlungen erfolgt.

Die Erstellung wird nach folgendem Format durchgeführt:

Verwendungszweck der Rechnungen: [Prefix] [Konnektor] [Zahl]

Verwendungszweck der Zahlungen: [Text] [Prefix] [Konnektor] [Zahl] [Text]

| Platzhalter | Beschreibung   |
|-------------|--|
| Text        | 20 willkürliche Buchstaben.  |
| Prefix      | Mögliche Präfixe: "", "RE", "re", "Rechnung", "RECHNUNG", "R", "r", "Rechnung RE", "Rechnung re", "Überweisung", "r", "Zahlung der Rechnung" |
| Konnektor   | Mögliche Konnektoren: "", " ", "-", "_", "/", "\"\", "x", ".", "#"   |
| Zahl        | Zufallszahl zwischen 0 und 10.000.000  |

Tabelle 11: Format des Verwendungszwecks

Der führende und endende Text bei dem Verwendungszweck der Zahlungen stellt willkürliche Eingaben des Benutzers dar, die mit dem eigentlich geforderten Verwendungszweck nichts zu tun haben.

Führende und endende Leerzeichen werden aus dem Verwendungszweck entfernt.

## 3.3 Testfälle

Die „journal“-Tabelle und die durch das Programm generierten Testdaten bilden die Grundlage für folgende Testfälle:

### 3.3.1 FixSizeQueryTest

Beim „FixSizeQueryTest“ werden Abfragen auf einer Tabelle mit fester Größe ausgeführt. Dabei werden nacheinander unterschiedliche Indextypen verwendet. Da bestimmte Abfragen sehr lange dauern, kann die maximale Laufzeit begrenzt werden.

#### Ziel

Ziel des Tests ist es, zu überprüfen, welche Zuordnungen mit einem konkreten Format des Verwendungszwecks durch eine Abfrage erzielt wird.

Anhand der Zuordnungsergebnisse kann überprüft werden, inwieweit sich eine Abfrage für das Format der erstellten Testdaten eignet und für welche Formate es gegebenenfalls eingesetzt werden kann. Der Test gibt Antworten auf die Zuordnungsprobleme.

#### Testverlauf

Im ersten Schritt werden aus der Tabelle „journal“ alle Tupel vom Typ „Rechnung“ geladen. Über diese Tupel wird iteriert. Innerhalb jeder Iteration wird mithilfe des Verwendungszwecks der Rechnung eine Abfrage konstruiert, die in der Menge der Zahlungen nach einem übereinstimmenden Verwendungszweck sucht. Dabei werden unterschiedliche Operatoren und Suchstrategien verwendet sowie unterschiedliche Indizes für die Spalte „documentText“ erzeugt. Jede Iteration mit einer Abfrage wird als „comparisonCount“ gezählt. Falls die Abfrage einen Treffer zurückliefert, wird dies als Match gewertet.

#### Messungen

Der Test generiert zwei Dateien:

- **Zeitmessung**

Nach jeder 10. Iteration wird eine Zeile in der CSV-Datei mit folgendem Format hinzugefügt: <comparisonCount>,<runTime>

Der <comparisonCount> gibt an, wie viele Rechnungen auf der Menge der Zahlungen verglichen wurden. Die <runtime> gibt die Zeit an, die für die im



<comparisonCount> angegebenen Vergleiche gebraucht wurde. Die runtime gibt die Zeit seit dem Beginn der Messung an und wird in Mikrosekunden gemessen.

- **Loggen der Übereinstimmung**

In dieser Datei werden alle Matches geloggt, die gefunden wurden.

Das Logging erfolgt im Format:

<ID-Rechnung>, <ID-Zahlung>, <Rechnung.DocumentText>, <Zahlung.DocumentText>

### **Testdaten / Testbedingungen**

Für den Test wurde mit dem Testdatengenerator eine CSV-Datei mit 2 Millionen Tupel erstellt. Davon sind 999.881 vom Typ Rechnung und 1.000.119 vom Typ Zahlungen. Die generierte Datei dient als Basis für alle in dem Testfall ausgeführten Abfragen. Die maximale Laufzeit wurde auf 2 Stunden festgelegt.

### **3.3.2 ScalingTest**

Beim „ScalingTest“ werden Abfragen auf Tabellen mit wachsender Tabellengröße (Skalierenden Daten) ausgeführt. Dabei werden nacheinander unterschiedliche Indextypen verwendet.

#### **Ziel**

Bei dem Test erfolgt eine Laufzeituntersuchung der Abfragen mit einem bestimmten Index bei skalierenden Daten. Dabei soll eine Klassifizierung der Laufzeiten in der in Kapitel 2.2 vorgestellten O-Notation erfolgen. Anhand der Ergebnisse kann der performanteste Index für einen Abfragetyp ermittelt werden. Zudem kann eine Aussage darüber getroffen werden, bis zu welchem Datenvolumen eine Abfrage noch in einer akzeptablen Zeit ausgeführt wird.

Der Test gibt Antworten auf die Performanceprobleme.

#### **Testverlauf**

Der Testverlauf ist mit dem des „FixSizeQueryTest“ identisch. Der Unterschied liegt darin, dass nicht über eine einzelne journal-Tabelle, sondern über mehrere Tabellen mit wachsender Größe, iteriert wird.

## **Messungen**

Für jede Tabelle einer bestimmten Größe wird die Gesamtzeit, die für die Iteration über die Rechnungen mit den jeweiligen Abfragen auf die Zahlungen benötigt wird, gemessen. Das Resultat wird in einer CSV-Datei gespeichert. Format:

<Anzahl der Datensätze>,<Zeit in Mikrosekunden>

## **Testdaten / Testbedingungen**

Für jeden Testdurchlauf werden dreißig Tabellen erzeugt, wobei die Größe von 1.000 bis 120.000 Daten reicht.

Die genaue Skalierung ist folgende:

Bei der Datengröße von 1.000 bis 20.000 wurde in Tausender-Schritten erhöht (20 Tabellen). Im Bereich zwischen 20.000 und 120.000 Datensätzen erfolgte die Erhöhung in Zehntausender-Schritten (10 Tabellen).

Die Daten bestehen jeweils zur Hälfte aus Rechnungen und zur Hälfte aus Zahlungen.

Für Abfragen, die sehr schnell ausgeführt werden, wird der Datenbestand nochmals erhöht, um die Grenze der Abfrage zu ermitteln.

### **3.3.3 IndexSizeTest**

Beim „IndexSizeTest“ wird die Indexgröße der unterschiedlichen Indextypen bei skalierenden Daten untersucht.

#### **Ziel**

Neben der Performance eines Index ist auch seine Größe auf der Festplatte ein wichtiges Kriterium. Kleine Indizes sind vorteilhaft, da sie aus wenig Pages bestehen und dadurch schneller durchsucht werden können. Zudem kann ein kleiner Index gegebenenfalls komplett in den Arbeitsspeicher geladen werden. Vorteilhaft ist auch, dass kleine Relationen zu kleinen Backups führen.

Es wird untersucht, wie sich steigende Daten in der Tabelle auf die Indexgröße auswirken.

#### **Testverlauf**

Es werden Tabellen unterschiedlicher Größe indiziert. Daraufhin wird die Größe vom Index anhand der Funktion in Abbildung 14 untersucht.

## **Messungen**

Bei der Ermittlung der Indexgröße wird die Pageanzahl sowie die Größe in KB angegeben.

## **Testdaten / Testbedingungen**

Anhand der Tabellen, die für den „ScalingTest“ angelegt wurden, wird der Verlauf der Indexgröße bei steigender Datenzahl in der Tabelle für die jeweiligen Indextypen gemessen.

### **3.3.4 Testverlauf**

Um einen möglichst einheitlichen Verlauf der Tests zu ermöglichen und den Einfluss eines möglichen Cachings zu minimieren, wird folgende Vorgehensweise bei allen Tests eingehalten:

- 1) Datenbank erstellen
- 2) Tabelle(n) erstellen
- 3) Daten importieren
- 4) ggf. Index erstellen
- 5) Testfall ausführen

Die Schritte werden manuell nacheinander ausgeführt.

### **3.3.5 Hardware**

Die Tests werden auf einem „hp – EliteBook“ mit folgenden Eigenschaften ausgeführt:

Betriebssystem: Windows 10, 64 Bit

Arbeitsspeicher: 8GB

Prozessor: Intel i5-2540M 2.60GHz

Festplatte: Samsung SSD 830 Series / Speicherkapazität 237 GB

### 3.4 Zusammenfassung Kapitel 3

In diesem Kapitel wurden sowohl die Testumgebung als auch die Generierung der Testdaten vorgestellt. Im Mittelpunkt stand die Journal-Tabelle, die Tupel vom Typ Rechnung und Zahlung enthält. Da über die Spalte „documentText“ die Auszifferung erfolgt, lag besonderes Augenmerk auf der Erläuterung des Formats.

Um eine Aussage über die Zuordnungsmöglichkeit bei einem konkreten Format zu erhalten, dient der „FixSizeQueryTest“. Auf einer Tabelle mit fester Größe erfolgt die Zuordnung von Rechnungen und Zahlungen über das festgelegte Format des Verwendungszwecks.

Um Antworten bezüglich der Performance bei skalierenden Daten zu bekommen, wurde der „ScalingTest“ erstellt, bei dem Abfragen auf Tabellen mit steigender Größe ausgeführt werden.

Bei den Tests werden nacheinander unterschiedliche Indextypen verwendet. Die Testfälle generieren Daten im CSV-Format, die die Grundlage für die Auswertung bilden. Es wurde der Testverlauf erläutert, der möglichst gleiche Bedingungen der Tests sicherstellen soll.

Im folgenden Kapitel werden Abfragen definiert, die innerhalb der vorgestellten Testfälle und Hardware ausgeführt werden.

## 4 Testdurchführung

Nach der Festlegung der Testfälle und der dabei zu testenden Indizes werden in diesem Kapitel Abfragen formuliert, mit denen die Zuordnung von Rechnungen und Zahlungen über den Verwendungszweck umgesetzt werden. Dabei wird zwischen fünf Abfragegruppen unterschieden:

### 1) Abfragen mit unverändertem Verwendungszweck

In der ersten Gruppe der Abfragen wird nach dem exakten Vorkommen des Verwendungszwecks (documenttext) gesucht. Es werden die zur Verfügung stehenden Platzhalter der Abfragetypen in unterschiedlichen Kombinationen verwendet, um beliebigen Text vor oder nach dem Verwendungszweck zu übergehen.

### 2) Abfragen mit Teilen des Verwendungszwecks

Es werden Abfragen definiert, die Teile des Verwendungszwecks für die Suche verwenden. Beispielsweise werden nur die Zahlen genutzt.

### 3) Abfragen auf aufbereiteten Daten des Verwendungszwecks

Bei der Aufbereitung der Daten werden bestimmte Teile des Verwendungszwecks extrahiert und in einer eigenen Spalte gespeichert. Die Abfragen werden auf den extrahierten Daten ausgeführt.

### 4) Volltextsuche

Es wird überprüft, inwieweit sich die Volltextsuche für die Zuordnung von Rechnungen und Zahlungen eignet. Die Volltextsuche wurde in Kapitel 2.7.5 vorgestellt.

### 5) Ähnlichkeitsüberprüfung

Es wird geprüft, inwieweit sich die Ähnlichkeitsüberprüfung mithilfe der im Kapitel 2.7.6 vorgestellten Trigramm-Similarity für die Zuordnung von Rechnungen und Zahlungen eignet.

Für die einzelnen Abfragen der Abfragegruppen wird überprüft, auf welchen Indizes sie ausgeführt werden können. Jede Abfrage wird zusätzlich zu den unterstützten Indizes auch immer ohne Index ausgeführt. Die dabei definierten Abfragen werden auf den zuvor erläuterten Testfällen getestet. Die Ergebnisse werden anhand von Graphen

gegenübergestellt. Nach jeder Testgruppe erfolgt eine Gegenüberstellung der Laufzeiten der jeweils schnellsten Indizes der einzelnen Abfragen, wobei der schnellste Index der Abfragegruppe ermittelt wird. Für den schnellsten Index wird die maximale Verarbeitung innerhalb einer Zeitspanne von 30 Minuten überprüft. Für jede Abfragegruppe erfolgt zudem eine Analyse der Zuordnungsmöglichkeiten.

## 4.1 Auswertung der Testergebnisse

Für die Auswertung der Testfälle wurde ein eigenes Programm geschrieben. Um eine bessere Vergleichbarkeit der Ergebnisse zu erhalten, werden sie auch visuell präsentiert (Abbildung 20).

Hier werden in Form von Graphen und Tabellen die Ergebnisse dargestellt. Aus den oben genannten Abfragegruppen können mit dieser Anwendung auch gruppenübergreifend Werte verglichen werden. Die vom Programm erzeugten Diagramme bilden die Grundlage bei den Vergleichen der Daten der Testfälle.

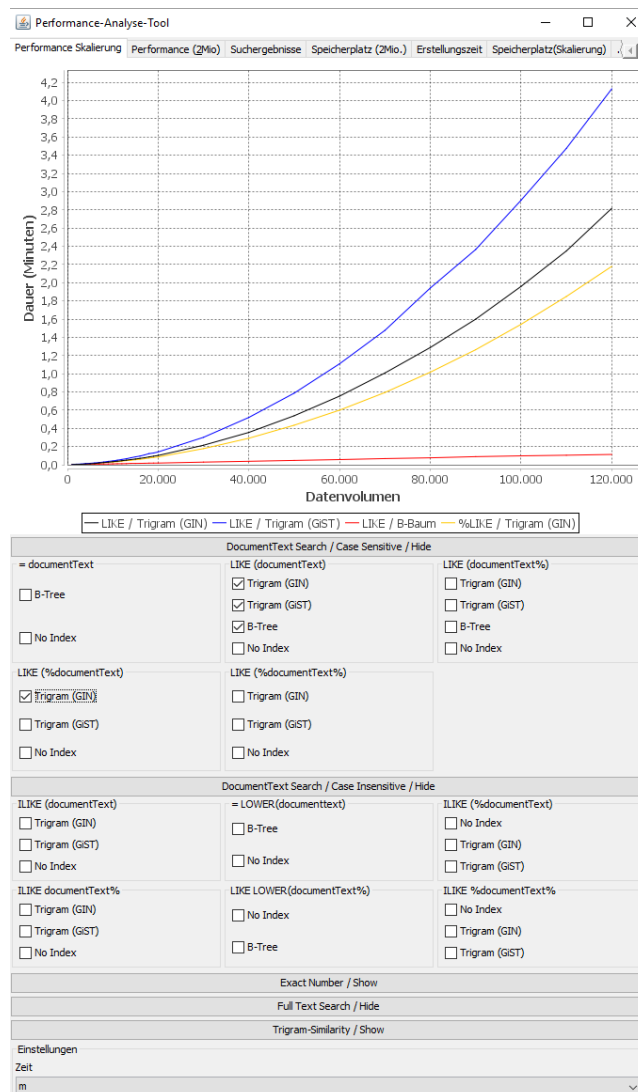


Abbildung 20: Auswertungsprogramm

## 4.2 Abfragen mit unverändertem Verwendungszweck

Im folgenden Kapitel werden Abfragen mit unverändertem Verwendungszweck der Rechnung auf der Menge der Zahlungen durchgeführt. Der Verwendungszweck wird dabei unverändert in die Abfrage übernommen. Je nach Operator werden die zur Verfügung stehenden Platzhalter so eingesetzt, um Text vor oder nach dem Verwendungszweck bei den Zahlungen zu umgehen.

### 4.2.1 Abfragen auf exakte Übereinstimmung des Verwendungszwecks

Die Überprüfung auf exakte Übereinstimmung des Verwendungszwecks erfolgt ohne Verwendung von Platzhaltern und wird mit den Operatoren = (Equal), LIKE, ILIKE durchgeführt.

Die Abfrage erfolgt bei = (Equal) und LIKE Case Sensitive, bei ILIKE Case Insensitive. Um mit den Operatoren Equal und LIKE auch Case Insensitive-Abfragen zu formulieren, wird der FB-B-Baum-Index verwendet.

#### Unterstützte Indizes

Für die Abfragen werden folgende Indizes unterstützt:

| Operator | B-Baum | FB-B-Baum | Trigramm (GIN) | Trigramm (GiST) |
|----------|--------|-----------|----------------|-----------------|
| =        | ✓      | ✓         | ✗              | ✗               |
| LIKE     | ✓      | ✓         | ✓              | ✓               |
| ILIKE    | ✗      | ✗         | ✓              | ✓               |

Tabelle 12: Unterstützte Indizes für die Suche nach exakter Übereinstimmung

Die im Folgenden aufgeführten Diagramme sind jeweils im größeren Maßstab im Anhang zu finden.



## Operator

=

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext =  
'Rechnung-541548185'
```

## Laufzeitanalyse

Der Equal-Operator wird mit dem B-Baum-Index ausgeführt:

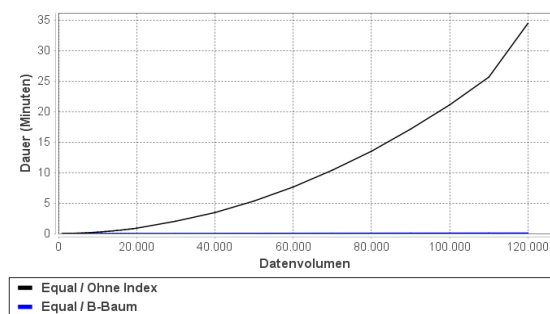


Abbildung 21: Laufzeit mit Operator Equal, Indizes: Ohne, B-Baum

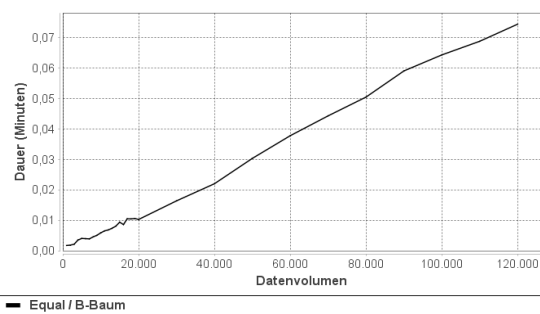


Abbildung 22: Laufzeit mit Operator Equal, Index: B-Baum

In Abbildung 21 sind die Laufzeiten der Abfragen bei der Verwendung der verschiedenen Indizes gegenübergestellt. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 34,46 Minuten benötigt. Mit dem B-Baum-Index wird eine Zeit von 0,08 Minuten erreicht. Da der Anstieg der Kurve mit dem B-Baum-Index aufgrund der Skalierung schlecht sichtbar ist, wird diese nochmal in Abbildung 22 aufgeführt. Im Vergleich zur Verarbeitung ohne Index wird mit dem B-Baum-Index eine Beschleunigung um den Faktor 431 erreicht.

## Laufzeit

Ohne Index: Quadratisch  $O(N^2)$

B-Baum-Index: leicht überlinear  $O(N \log N)$

## Operator

= (unter Nutzung der LOWER-Funktion)

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND  
LOWER(documenttext) = LOWER('Rechnung-541548185')
```

## Laufzeitanalyse

Der Equal-Operator wird mit dem FB-B-Baum-Index ausgeführt:

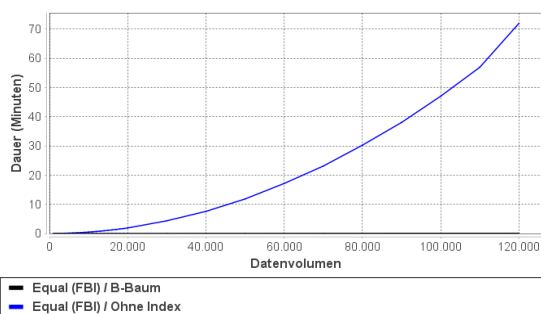


Abbildung 23: Laufzeit mit Operator Equal (LOWER),  
Indizes: Ohne, FB-B-Baum

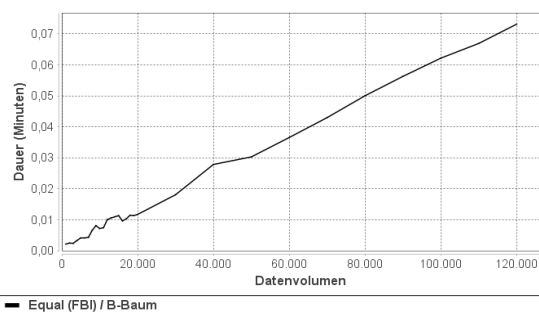


Abbildung 24: Laufzeit mit Operator Equal (LOWER),  
Index: FB-B-Baum

In Abbildung 23 ist die Gegenüberstellung der Laufzeiten zu sehen. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 71,94 Minuten (1,2 Stunden) benötigt, wohingegen der FB-B-Baum-Index 0,07 Minuten für die Bearbeitung braucht. Um den Anstieg des FB-Index besser zu erkennen, ist dieser nochmal in Abbildung 24 aufgeführt. Im Vergleich zur Verarbeitung ohne Index wird mit dem FB-B-Baum-Index eine Beschleunigung um den Faktor 1028 erreicht.

## Laufzeit

Ohne Index: Quadratisch  $O(N^2)$

Mit FB-B-Baum-Index: leicht überlinear  $O(N \log N)$

## Operator

LIKE

### Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext  
LIKE 'Rechnung-541548185'
```

### Laufzeitanalyse

Der LIKE-Operator wird mit dem B-Baum-, Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

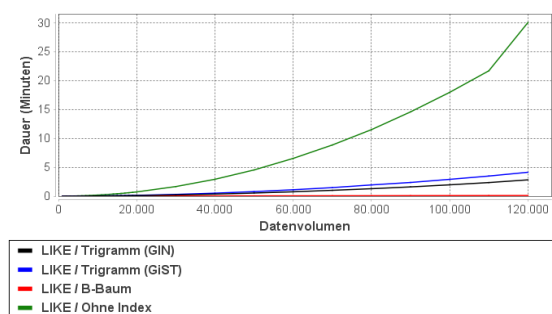


Abbildung 25: Laufzeit mit Operator LIKE, Indizes: Ohne, B-Baum, Trigramm (GIN), Trigramm (GiST)

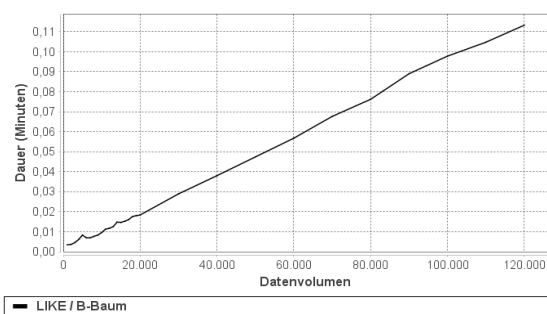


Abbildung 26: Laufzeit mit Operator LIKE Index: B-Baum

In Abbildung 25 erfolgt die Gegenüberstellung der Laufzeiten. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 30,03 Minuten benötigt. Hingegen braucht der Trigramm (GiST)-Index 4,13 Minuten. Der Trigramm (GIN)-Index kommt auf 2,82 Minuten. Die beste Performance ist mit dem B-Baum zu erzielen, der 0,11 Minuten benötigt. Zur Verdeutlichung wird sein Verlauf in Abbildung 26 nochmals dargestellt. Im Vergleich zur Verarbeitung ohne Index wird mit dem B-Baum-Index eine Beschleunigung um den Faktor 273 erreicht.

### Laufzeit

Ohne- / Trigramm (GIN)- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

B-Baum-Index: leicht überlinear  $O(N \log N)$

## Operator

LIKE (unter Nutzung der LOWER-Funktion)

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND  
LOWER(documenttext) LIKE LOWER('Rechnung-541548185')
```

## Laufzeitanalyse

Der LIKE-Operator wird mit dem FB-B-Baum-Index ausgeführt:

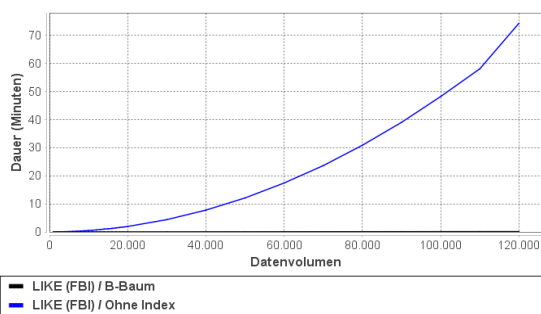


Abbildung 27: Laufzeit mit Operator LIKE (LOWER),  
Indizes: Ohne, FB-B-Baum

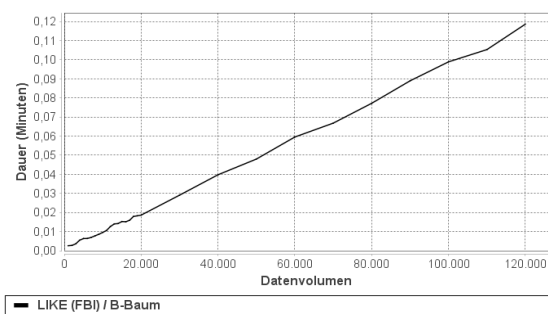


Abbildung 28: Laufzeit mit Operator LIKE (LOWER),  
Indizes: FB-B-Baum

In Abbildung 27 erfolgt die Gegenüberstellung der Laufzeiten. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 74,25 Minuten (1,24 Stunden) benötigt. Der FB-B-Baum-Index hingegen braucht 0,12 Minuten. Um den Anstieg des FB-B-Baum-Index besser zu erkennen, ist dieser nochmal in Abbildung 28 aufgeführt. Im Vergleich zur Verarbeitung ohne Index wird mit dem FB-B-Baum-Index eine Beschleunigung um den Faktor 619 erreicht.

## Laufzeit

Ohne Index: Quadratisch  $O(N^2)$

FB-Baum-Index: leicht überlinear  $O(N \log N)$

## Operator

ILIKE

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext  
ILIKE 'Rechnung-541548185'
```

## Laufzeitanalyse

Der ILIKE-Operator wird mit dem Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

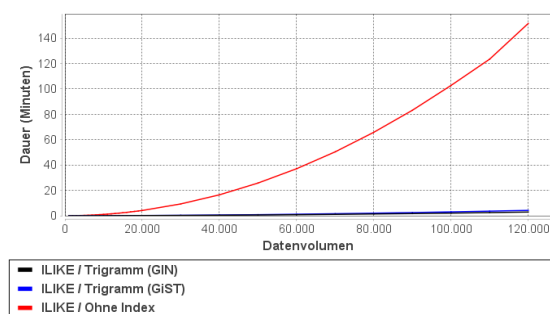


Abbildung 29: Laufzeit mit Operator ILIKE, Indizes: Ohne, Trigramm (GIN), Trigramm (GiST)

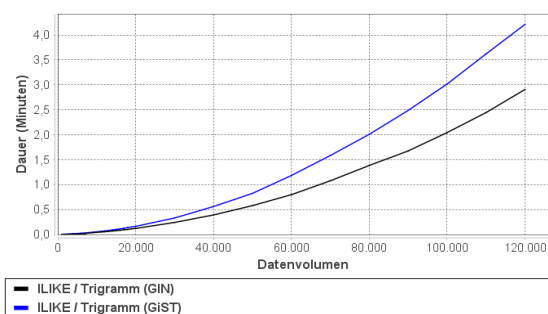


Abbildung 30: Laufzeit mit Operator ILIKE, Indizes: Trigramm (GIN), Trigramm (GiST)

In Abbildung 29 sind die Laufzeiten der Abfragen bei der Verwendung der verschiedenen Indizes gegenübergestellt. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 151,67 Minuten (2,53 Stunden) benötigt. Hingegen braucht der Trigramm (GiST)-Index 4,22 Minuten. Die beste Performance ist mit dem Trigramm (GIN)-Index erreichbar, der 2,91 Minuten benötigt. Zur Verdeutlichung ist sein Verlauf in Abbildung 30 nochmals mit dem Trigramm (GiST)-Index zu sehen.

Im Vergleich zur Verarbeitung ohne Index wird mit dem Trigramm (GIN)-Index eine Beschleunigung um den Faktor 52 erreicht.

## Laufzeit

Ohne- / Trigramm (GIN)- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

## Gegenüberstellung der Laufzeiten

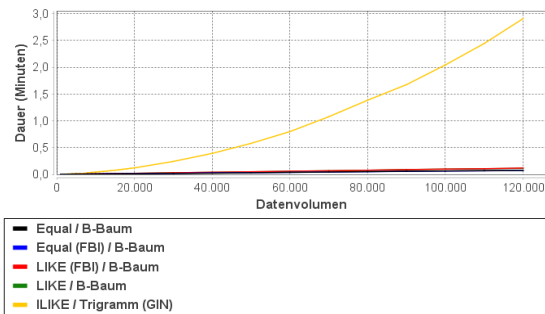


Abbildung 31: Laufzeit mit Operator Equal, Index: B-Baum / Operator Equal (LOWER), Index: FB-B-Baum / Operator LIKE (LOWER), Index: FB-B-Baum / Operator LIKE, Index: B-Baum / Operator ILIKE, Index: Trigramm (GIN)

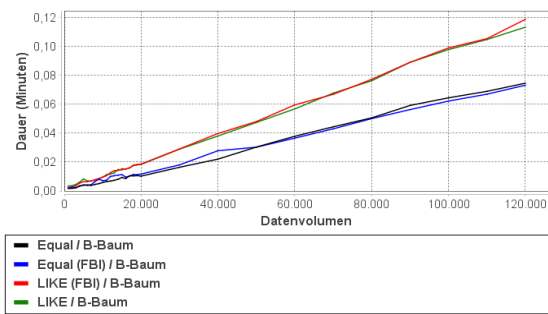


Abbildung 32: Laufzeit mit Operator Equal, Index: B-Baum / Operator Equal (LOWER), Index: FB-B-Baum / Operator LIKE (LOWER), Index: FB-B-Baum / Operator LIKE, Index: B-Baum

In Abbildung 31 erfolgt die Gegenüberstellung der Laufzeit der jeweils schnellsten Indizes der einzelnen Operatoren.

Da die Laufzeiten der schnellsten Indizes nicht deutlich zu erkennen sind, werden diese nochmals in Abbildung 32 ohne den ILIKE Operator mit dem Trigramm (GIN)-Index aufgeführt.

Es ist zu entnehmen, dass der LIKE-Befehl mit dem FB-B-Baum- und dem B-Baum Index nahezu die gleiche Performance hat.

Der Equal-Operator hat auch nahezu die gleiche Performance mit dem FB-B-Baum- und dem B-Baum-Index und liefert eine bessere Performance als der LIKE-Operator.

Eine Case Insensitive-Suche kann bei beiden Operatoren ohne Performanceeinbuße durchgeführt werden.

## Maximale Verarbeitung innerhalb 30 Minuten

Da die beste Performance mit dem Equal-Operator in Verbindung mit dem B-Baum- und dem FB-B-Baum-Index zu erzielen ist, werden sie auf ihre maximale Verarbeitung untersucht.

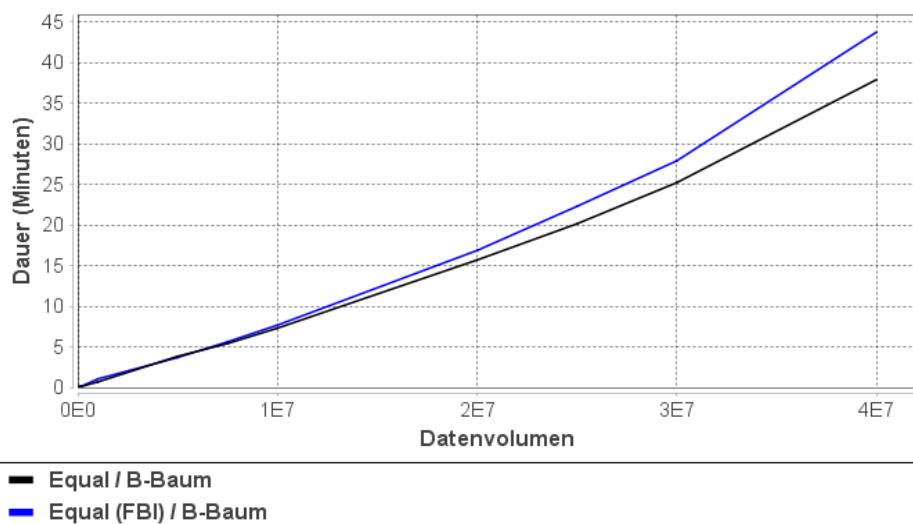


Abbildung 33: Maximale Laufzeit mit Operator Equal, Indizes: B-Baum, FB-B-Baum

Aus Abbildung 33 ist zu entnehmen, dass innerhalb einer halben Stunde ca. 32.000.000 Datensätze bearbeitet werden können.

## Analyse der Zuordnungsmöglichkeiten

Bei der Suche nach dem exakten Vorkommen des Verwendungszwecks können nur Rechnungen und Zahlungen zugeordnet werden, deren Verwendungszweck fehlerfrei angegeben wurde. Der einzige Unterschied, der abgefangen werden kann, ist die Abweichung der Groß- und Kleinschreibung.

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung |
|-------------------------------|------------------------------|
| Überweisung.1488879           | Überweisung.1488879          |
| Rechnung RE-9832053           | Rechnung re-9832053          |
| Rechnung re.5325606           | rechnung re.5325606          |

Tabelle 13: Exakte Zuordnungen von Rechnungen und Zahlungen

## 4.2.2 Abfragen unter Auslassung von beliebigen führenden Zeichen

Abfragen unter Auslassung von beliebigen führenden Zeichen werden mit den Operatoren LIKE und ILIKE ausgeführt. Die Abfragen erfolgen unter Verwendung des Platzhalters %, der dem Verwendungszweck der einzelnen Rechnungen vorangestellt wird (%documenttext).

### Unterstützte Indizes

Für die Abfragen werden folgende Indizes unterstützt:

| Operator | B-Baum | Trigramm (GIN) | Trigramm (GiST) |
|----------|--------|----------------|-----------------|
| LIKE     | x      | ✓              | ✓               |
| ILIKE    | x      | ✓              | ✓               |

*Tabelle 14: Unterstützte Indizes der Abfrage LIKE / ILIKE %documenttext*

Die im Folgenden aufgeführten Diagramme sind jeweils im größeren Maßstab im Anhang zu finden.



## Operator

LIKE

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext  
LIKE '%RE-12255'
```

## Laufzeitanalyse

Der LIKE-Operator mit führendem Platzhalter (%) wird mit dem Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

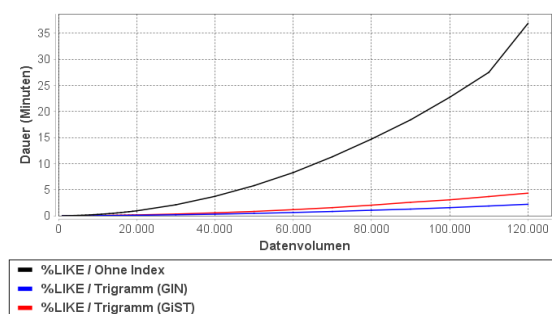


Abbildung 34: Laufzeit mit Operator %LIKE, Indizes:  
Ohne, Trigramm (GIN), Trigramm (GiST)

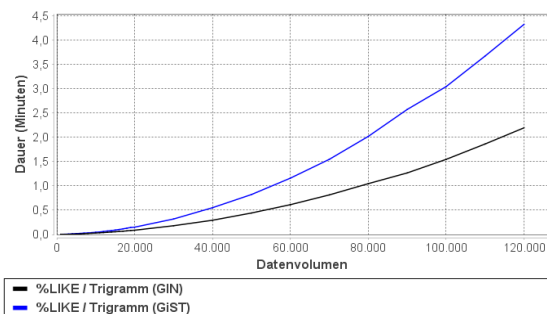


Abbildung 35: Laufzeit mit Operator %LIKE, Indizes:  
Trigramm (GIN), Trigramm (GiST)

In Abbildung 34 sind die Laufzeiten der Abfragen auf den einzelnen Indizes gegenübergestellt. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 36,88 Minuten benötigt. Hingegen braucht der Trigramm (GiST)-Index 4,33 Minuten. Die beste Performance ist mit dem Trigramm (GIN)-Index zu erreichen, der 2,20 Minuten benötigt. Zur Verdeutlichung wird sein Verlauf in Abbildung 35 nochmals zusammen mit dem Trigramm (GiST)-Index dargestellt.

Im Vergleich zur Verarbeitung ohne Index wird mit dem Trigramm (GIN)-Index eine Beschleunigung um den Faktor 17 erreicht.

## Laufzeit

Ohne- / Trigramm (GIN)- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

## Operator

ILIKE

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext  
ILIKE '%RE-12255557'
```

## Laufzeitanalyse

Der ILIKE-Operator mit führendem Platzhalter (%) wird mit dem Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

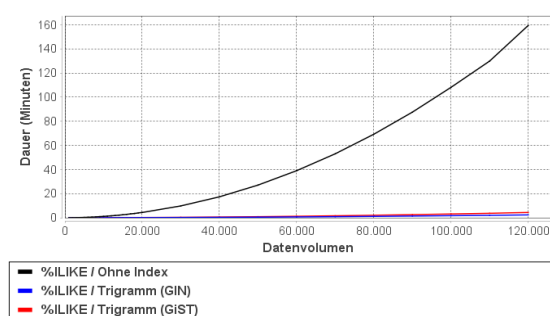


Abbildung 36: Laufzeit mit Operator %ILIKE, Indizes:  
Ohne, Trigramm (GIN), Trigramm (GiST)

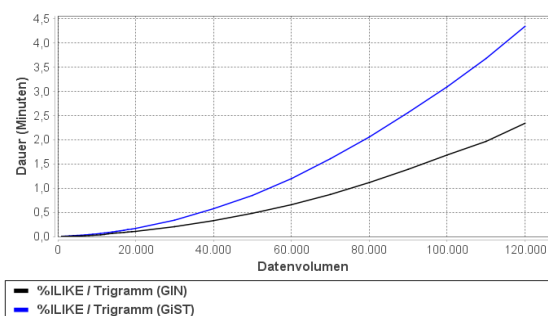


Abbildung 37: Laufzeit mit Operator %ILIKE, Indizes:  
Trigramm (GIN), Trigramm (GiST)

In Abbildung 36 sind die Laufzeiten der Abfragen auf den einzelnen Indizes gegenübergestellt. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 159,43 Minuten (2,66 Stunden) benötigt. Hingegen braucht der Trigramm (GiST)-Index 4,34 Minuten. Die beste Performance ist mit dem Trigramm (GIN)-Index zu erreichen, der 2,34 Minuten benötigt. Zur Verdeutlichung des Verlaufs werden in Abbildung 37 die Laufzeiten des Trigramm (GIN)- und Trigramm (GiST)-Index nochmals aufgeführt.

Im Vergleich zur Verarbeitung ohne Index wird mit dem Trigramm (GIN)-Index eine Beschleunigung um den Faktor 68 erreicht.

## Laufzeit

Ohne- / Trigramm (GIN)- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

## Gegenüberstellung der Laufzeiten

In Abbildung 38 erfolgt die Gegenüberstellung der Laufzeit der jeweils schnellsten Indizes der jeweiligen Operatoren:

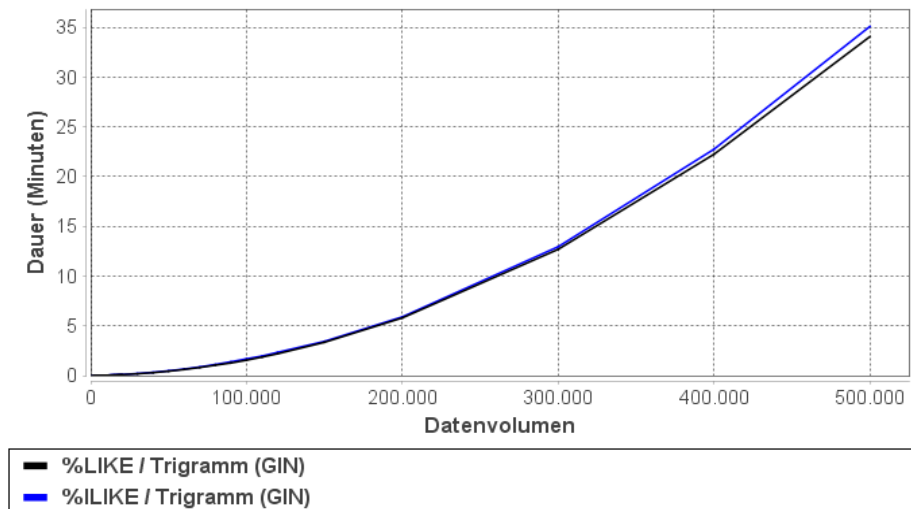


Abbildung 38: Maximale Laufzeit mit Operator %LIKE und %ILIKE, Indizes: Trigramm (GIN)

Es ist zu erkennen, dass der LIKE- und ILIKE-Operator in Verbindung mit dem Trigramm (GIN)-Index nahezu die gleiche Performance haben. Eine Case Insensitive-Suche kann demnach ohne Performanceeinbuße durchgeführt werden.

## Maximale Verarbeitung innerhalb 30 Minuten

Mit den Abfragen können in einer Zeitspanne von 30 Minuten ca. 450.000 Datensätze bearbeitet werden (Abbildung 38).

## Analyse der Zuordnungsmöglichkeiten

Abfragen unter Auslassung von beliebigen führenden Zeichen prüfen, ob ein Verwendungszweck einer Zahlung mit einer bestimmten Zeichenkette endet. Somit können auch Rechnungen und Zahlungen zugeordnet werden, wenn vor dem geforderten Verwendungszweck beliebige Zeichen stehen.

Ist z.B. der Verwendungszweck der Rechnung „RE-6983761“, kann durch die Abfrage eine Zuordnung zu der Zahlung „Vielen Dank. RE-6983761“ vorgenommen werden.

Hierzu sind in Tabelle 15 weitere Beispiele aus den Testdaten zu sehen:

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung               |
|-------------------------------|--|
| Rechnung.6983761              | Zahlung der Rechnung.6983761               |
| re5466556                     | Rechnung re5466556                         |
| \2598089                      | NIBUJVAUGWDCDBVQGRQ<br>Überweisung\2598089 |

Tabelle 15: Erfolgsversprechende Zuordnungen der %LIKE Abfrage

Die Zuordnungsstrategie funktioniert solange das Format des Verwendungszwecks sicherstellt, dass ein Verwendungszweck nicht als Teil eines anderen Verwendungszwecks vorkommt.

Dies ist besonders bei Formaten der Fall, die nur aus Zahlen bestehen bzw. mit Zahlen enden:

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung |
|-------------------------------|------------------------------|
| 825842                        | r.3825842                    |
| 977609                        | R#1977609                    |

Tabelle 16: Fehlzusordnungen der %LIKE Abfrage

Dem kann entgegengewirkt werden, indem die Zahlen mit konstanter Länge angegeben werden:

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung |
|-------------------------------|------------------------------|
| 0000000825842                 | r.0000000825842              |

Tabelle 17: Zahlenformat mit fester Länge

Durch die Festlegung einer festen Länge kann ein Verwendungszweck nicht mehr in einem anderen enthalten sein. Somit kann es nicht mehr zu einer Zuordnung der Zeichenketten „825842“ und „r.3825842“ kommen („000000825842“ und „r.000003825842“).

### 4.2.3 Abfragen unter Auslassung von beliebigen endenden Zeichen

Abfragen unter Auslassung von beliebigen endenden Zeichen werden mit den Operatoren LIKE und ILIKE ausgeführt. Die Abfragen erfolgen unter Verwendung des Platzhalters %, der dem Verwendungszweck der einzelnen Rechnungen nachgestellt wird (documenttext%).

Der LIKE-Operator führt die Suche Case Sensitive aus. Um mit dem Operator auch Case Insensitive Abfragen zu formulieren, wird der FB-B-Baum Index verwendet.

#### Unterstützte Indizes

Für die Abfragen werden folgende Indizes unterstützt:

| Operator | B-Baum | FB-B-Baum | Trigramm (GIN) | Trigramm (GiST) |
|----------|--------|-----------|----------------|-----------------|
| LIKE     | ✓      | ✓         | ✓              | ✓               |
| ILIKE    | ✗      | ✗         | ✓              | ✓               |

Tabelle 18: Unterstützte Indizes der Abfrage LIKE / ILIKE documenttext%

Die im Folgenden aufgeführten Diagramme sind jeweils im größeren Maßstab im Anhang zu finden.

## Operator

LIKE

### Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext  
LIKE 'RE-12255557%'
```

### Laufzeitanalyse

Der LIKE-Operator mit endendem Platzhalter (%) wird mit dem B-Baum-, Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

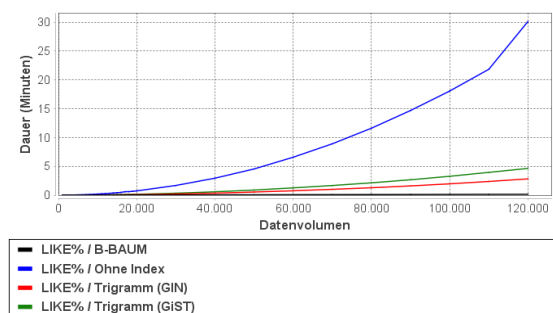


Abbildung 39: Laufzeit mit Operator LIKE%, Indizes: Ohne, B-Baum, Trigram (GIN), Trigram (GiST)

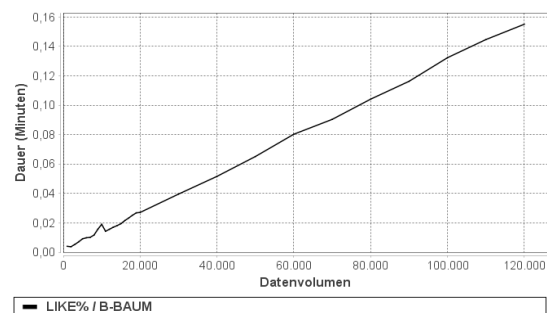


Abbildung 40: Laufzeit mit Operator LIKE%, Index: B-Baum

In Abbildung 39 sind die Laufzeiten der Abfragen gegenübergestellt. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 30,13 Minuten benötigt. Hingegen braucht der Trigramm (GiST)-Index 4,64 Minuten. Der Trigramm (GIN)-Index kommt auf 2,82 Minuten. Die beste Performance ist mit dem B-Baum-Index zu erzielen, der 0,16 Minuten benötigt. Zur Verdeutlichung wird sein Verlauf in Abbildung 40 nochmals dargestellt.

Im Vergleich zur Verarbeitung ohne Index wird mit dem B-Baum-Index eine Beschleunigung um den Faktor 188 erreicht.

### Laufzeit

Ohne- / Trigramm (GIN)- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

B-Baum-Index: leicht überlinear  $O(N \log N)$

## Operator

LIKE% (unter Nutzung der LOWER-Funktion)

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND  
LOWER(documenttext) LIKE LOWER('RE-12255557%')
```

## Laufzeitanalyse

Der LIKE-Operator mit endendem Platzhalter (%) wird mit dem FB-B-Baum-Index ausgeführt:

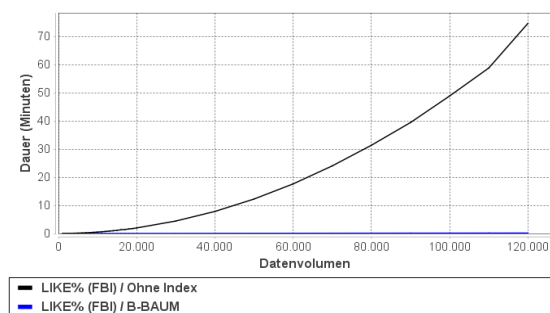


Abbildung 41: Laufzeit mit Operator LIKE (LOWER),  
Indizes: Ohne, FB-B-Baum

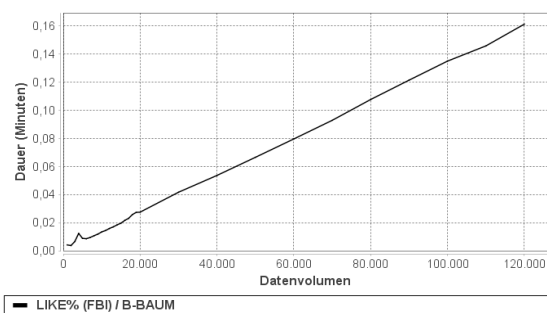


Abbildung 42: Laufzeit mit Operator LIKE (LOWER),  
Index: FB-B-Baum

Die Gegenüberstellung der Laufzeiten erfolgt in Abbildung 41. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 74,63 Minuten (1,24 Stunden) benötigt. Mit dem FB-B-Baum-Index ist dies in 0,16 Minuten erreichbar. Zur Verdeutlichung wird sein Verlauf in Abbildung 42 nochmals dargestellt.

Im Vergleich zur Verarbeitung ohne Index entspricht dies einer Beschleunigung um den Faktor 466.

## Laufzeit

Ohne Index: Quadratisch  $O(N^2)$

FB-B-Baum-Index: leicht überlinear  $O(N \log N)$

## Operator

ILIKE

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext  
ILIKE 'RE-12255557%'
```

## Laufzeitanalyse

Der ILIKE-Operator mit endendem Platzhalter (%) wird mit dem Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

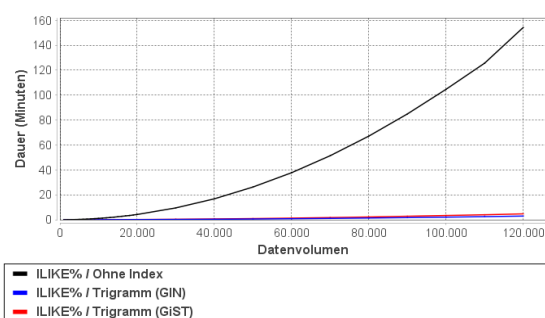


Abbildung 43: Laufzeit mit Operator ILIKE%, Indizes: Ohne, Trigramm (GIN), Trigramm (GiST)

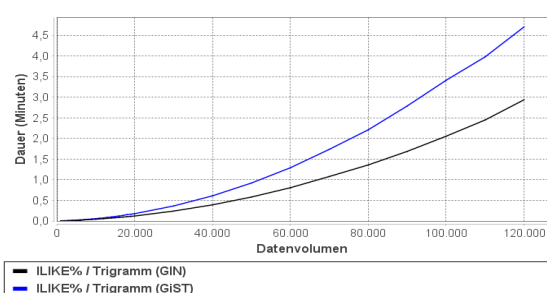


Abbildung 44: Laufzeit mit Operator ILIKE%, Indizes: Trigramm (GIN), Trigramm (GiST)

In Abbildung 43 sind die Laufzeiten der Abfragen gegenübergestellt. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 154,37 Minuten (2,57 Stunden) benötigt, wobei der Trigramm (GiST)-Index 4,71 Minuten braucht. Die beste Performance ist mit dem Trigramm (GIN)-Index zu erreichen, der 2,94 Minuten benötigt. Zur Verdeutlichung des Verlaufs werden in Abbildung 44 die Laufzeiten des Trigramm (GIN)- und Trigramm (GiST)-Index nochmals aufgeführt.

Im Vergleich zur Verarbeitung ohne Index wird mit dem Trigramm (GIN)-Index eine Beschleunigung um den Faktor 53 erreicht.

## Laufzeit

Ohne- / Trigramm (GIN)- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$



## Gegenüberstellung der Laufzeiten

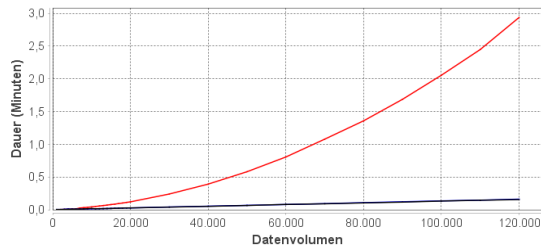


Abbildung 45: Laufzeit mit Operator LIKE%, Index: B-Baum / Operator: LIKE%(LOWER), Index: FB-B-Baum / ILIKE%, Index: Trigramm (GIN)

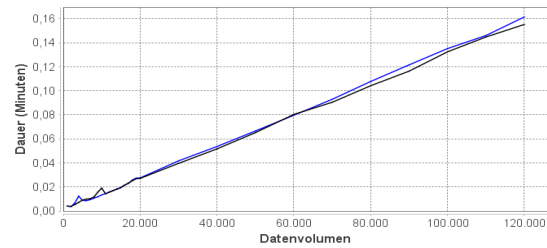


Abbildung 46: Laufzeit mit Operator LIKE%, Index: B-Baum / Operator: LIKE% (LOWER), Index: B-Baum

In Abbildung 45 erfolgt die Gegenüberstellung der Laufzeiten der jeweils schnellsten Indizes der jeweiligen Operatoren.

Um den Verlauf der LIKE-Abfragen in Verbindung mit dem FB-B-Baum- und dem B-Baum-Index besser zu erkennen, sind diese nochmals in Abbildung 46 aufgeführt. Es ist zu sehen, dass beide Abfragen nahezu die gleiche Performance haben. Eine Case Insensitive-Abfrage kann demnach ohne Performanceeinbuße durchgeführt werden.

## Maximale Verarbeitung innerhalb 30 Minuten

Mit den Abfragen mit dem FB-B-Baum- und dem B-Baum-Index können innerhalb von 30 Minuten ca. 15.000.000 Daten bearbeitet werden (Abbildung 47).

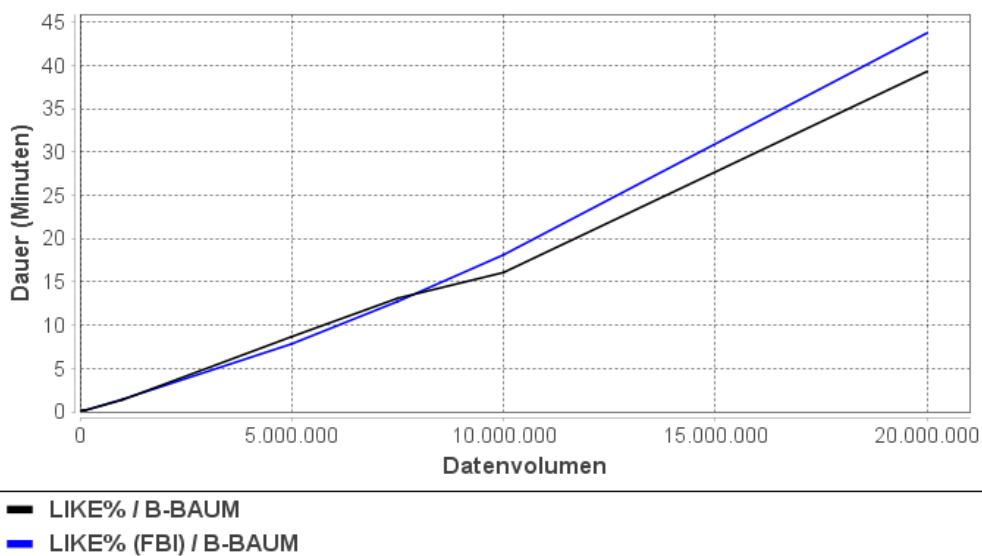


Abbildung 47: Maximale Laufzeit mit Operator LIKE%, Index: B-Baum / Operator: LIKE% (LOWER), Index: B-Baum

## Analyse der Zuordnungsmöglichkeiten

Abfragen unter Auslassung von beliebigen endenden Zeichen prüfen, ob ein Verwendungszweck einer Zahlung mit einer bestimmten Zeichenkette beginnt. Somit können auch Rechnungen und Zahlungen zugeordnet werden, wenn nach dem geforderten Verwendungszweck beliebige Zeichen vorkommen.

Ist z.B. der Verwendungszweck „RE-6983761“ gefordert, kann durch die Abfrage eine Zuordnung zu der Zahlung „RE-6983761 Danke!“ erfolgen.

Hierzu weitere Beispiele aus den Testdaten.

| Verwendungszweck der Zahlung | Verwendungszweck der Rechnung           |
|------------------------------|---|
| Überweisung41405             | Überweisung41405<br>WBDWGFAYEFLUMVTTYRD |
| r#58578                      | r#58578 JTISDVIIYQLKBYTOSUPFC           |

Tabelle 19: Erfolgversprechende Zuordnungen der LIKE% / ILIKE% Abfrage

Die Zuordnungsstrategie funktioniert solange das Format des Verwendungszwecks sicherstellt, dass ein Verwendungszweck nicht als Teil eines anderen Verwendungszwecks vorkommt.

Dies ist besonders bei Formaten der Fall, die nur aus Zahlen bestehen bzw. mit einer Zahl enden.

| Verwendungszweck der Zahlung | Verwendungszweck der Rechnung |
|------------------------------|-------------------------------|
| Zahlung der Rechnung-6181    | Zahlung der Rechnung-6181301  |
| r 580                        | r 5801784                     |
| 94376                        | 943769                        |

Tabelle 20: Fehlzusordnungen der LIKE% / ILIKE% Abfrage

Dem kann entgegengewirkt werden, indem die Zahlen in konstanter Länge angegeben werden:

| Verwendungszweck der Zahlung    | Verwendungszweck der Rechnung                  |
|---------------------------------|--|
| Zahlung der Rechnung-0000006181 | Zahlung der Rechnung-0000006181<br>Vielen Dank |

Tabelle 21: Zahlenformat mit fester Länge

#### 4.2.4 Abfragen unter Auslassung von beliebigen führenden und endenden Zeichen

Abfragen unter Auslassung von führenden und endenden Zeichen werden mit den Operatoren LIKE und ILIKE durchgeführt. Die Abfragen erfolgen unter Verwendung des Platzhalters %, der dem Verwendungszweck der einzelnen Rechnungen voran- und nachgestellt wird (%documenttext%).

##### Unterstützte Indizes

Für die Abfragen werden folgende Indizes unterstützt:

| Operator | B-Baum | Trigramm (GIN) | Trigramm (GiST) |
|----------|--------|----------------|-----------------|
| LIKE     | x      | ✓              | ✓               |
| ILIKE    | x      | ✓              | ✓               |

Tabelle 22: Unterstützte Indizes der Abfrage LIKE / ILIKE %documenttext%

Die im Folgenden aufgeführten Diagramme sind jeweils im größeren Maßstab im Anhang zu finden.

## Operator

LIKE

### Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext  
LIKE '%RE-12255%'
```

### Laufzeitanalyse

Der LIKE-Operator mit führendem und endendem Platzhalter (%) wird mit dem Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

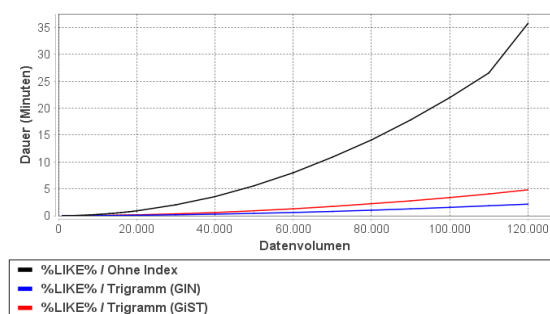


Abbildung 48: Laufzeit mit Operator %LIKE%,  
Indizes: Ohne, Trigramm (GIN), Trigramm (GiST)

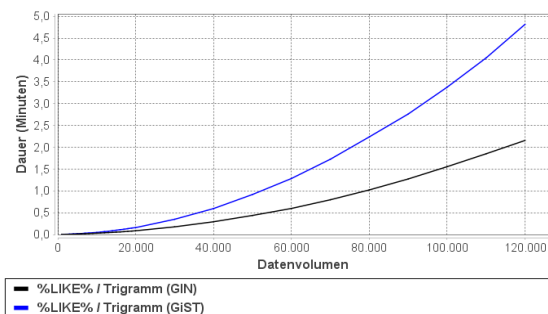


Abbildung 49: Laufzeit mit Operator %LIKE%, Indizes:  
Trigramm (GIN), Trigramm (GiST)

Die Gegenüberstellung der Laufzeiten erfolgt in Abbildung 48. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 35,78 Minuten benötigt. Hingegen braucht der Trigramm (GiST)-Index 4,82 Minuten. Die beste Performance ist mit dem Trigramm (GIN)-Index zu erzielen, der 2,16 Minuten benötigt. Zur Verdeutlichung des Verlaufs werden in Abbildung 49 die Laufzeiten des Trigramm (GIN)- und Trigramm (GiST)-Index nochmals aufgeführt.

Im Vergleich zur Verarbeitung ohne Index wird mit dem Trigramm (GIN)-Index eine Beschleunigung um den Faktor 17 erreicht.

### Laufzeit

Ohne- / Trigramm (GIN)- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

## Operator

ILIKE

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext  
ILIKE '%RE-12255%'
```

## Laufzeitanalyse

Der ILIKE-Operator mit führendem und endendem Platzhalter (%) wird mit dem Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

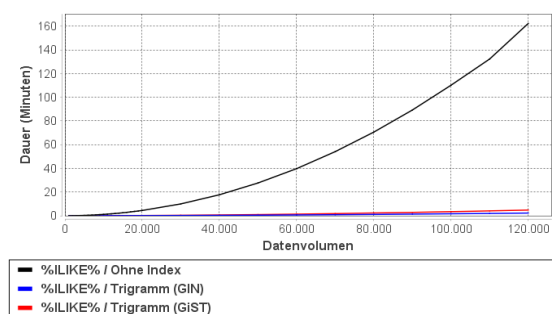


Abbildung 50: Laufzeit mit Operator %ILIKE%, Indizes: Ohne, Trigramm (GIN), Trigramm (GiST)

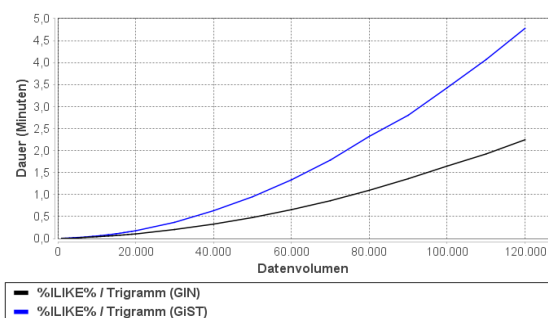


Abbildung 51: Laufzeit mit Operator %ILIKE%, Indizes: Trigramm (GIN), Trigramm (GiST)

In Abbildung 50 erfolgt die Gegenüberstellung der Laufzeiten. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 162,38 Minuten (2,71 Stunden) benötigt. Hingegen braucht der Trigramm (GiST)-Index 4,78 Minuten. Die beste Performance ist mit dem Trigramm (GIN)-Index zu erzielen, der 2,25 Minuten benötigt. Zur Verdeutlichung des Verlaufs werden in Abbildung 51 die Laufzeiten des Trigramm (GIN)- und Trigramm (GiST)-Index nochmals aufgeführt.

Im Vergleich zur Verarbeitung ohne Index wird mit dem Trigramm (GIN)-Index eine Beschleunigung um den Faktor 72 erreicht.

## Laufzeit

Ohne- / Trigramm (GIN)- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

## Gegenüberstellung der Laufzeiten

In Abbildung 52 erfolgt die Gegenüberstellung der Laufzeit der jeweils schnellsten Indizes der jeweiligen Operatoren:

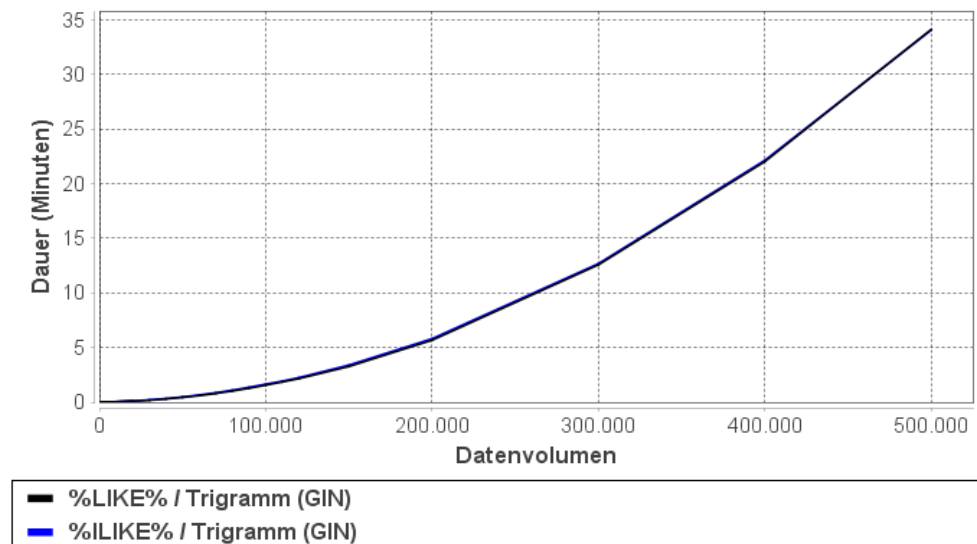


Abbildung 52: Maximale Laufzeit mit Operator %LIKE%, Indizes: Trigramm (GIN), Trigramm (GiST)

Es ist zu erkennen, dass der LIKE- und ILIKE-Operator in Kombination mit dem Trigramm (GIN)-Index nahezu die gleiche Performance haben. Eine Case Insensitive Suche kann demnach ohne Performanceeinbuße durchgeführt werden.

## Maximale Verarbeitung innerhalb 30 Minuten

Mit den Abfragen können in einer Zeitspanne von 30 Minuten ca. 450.000 Datensätze bearbeitet werden (Abbildung 52).

## Analyse der Zuordnungsmöglichkeiten

Abfragen unter Auslassung von beliebigen führenden und endenden Zeichen prüfen, ob der Verwendungszweck einer Zahlung eine bestimmte Zeichenkette enthält. Somit können auch Rechnungen und Zahlungen zugeordnet werden, wenn vor und nach dem geforderten Verwendungszweck beliebige Zeichen vorkommen.

Ist z.B. der Verwendungszweck „RE-6983761“ gefordert, kann durch die Abfrage eine Zuordnung zu der Zahlung mit dem Verwendungszweck „Bezahlung RE-6983761 Vielen Dank.“ vorgenommen werden. In Tabelle 23 sind weitere Beispiele aus den Testdaten aufgeführt:

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung                      |
|-------------------------------|---|
| #6501172                      | Rechnung RE#6501172<br>LOZFLLVSKUULQATUWVKL       |
| re\6087130                    | Rechnung re\6087130                               |
| Zahlung der Rechnung/2569     | Zahlung der Rechnung/2569<br>JVQXPALTMMOGPQSDBWOA |

Tabelle 23: Erfolgversprechende Zuordnungen der Abfrage %LIKE% / %ILIKE%

Die Zuordnungsstrategie funktioniert solange das Format des Verwendungszwecks sicherstellt, dass ein Verwendungszweck nicht als Teil eines anderen Verwendungszwecks vorkommt. In Tabelle 24 ist eine derartige Fehlzusammenordnung dargestellt.

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung                         |
|-------------------------------|--|
| Rechnung#850180               | Zahlung der Rechnung#8501807<br>CNVPJYDEARMFWWUEGLDS |

Tabelle 24: Fehlerhafte Zuordnung. Führende und Endende Zeichen umgehen

Wie auch bei der Zuordnung unter Auslassung von beliebigen führenden wie auch unter Auslassung von beliebigen endenden Zeichen hilft die Angabe der Zahlen in fester Länge.

## 4.3 Abfragen mit Teilen des Verwendungszwecks

Für die Suche nach Teilen des Verwendungszwecks werden reguläre Ausdrücke verwendet. Für die im Kapitel 3.2 vorgestellten Testdaten gilt, dass die Zahlen in dem Verwendungszweck ein besonderes Merkmal sind. Deshalb werden sie auf die Eignung für die Zuordnung von Rechnungen und Zahlungen geprüft.

Dafür werden bei der Suche nach passenden Zahlungen alle Zeichen bis auf die Zahlen aus dem Verwendungszweck der Rechnungen entfernt. Somit wird bei dem Verwendungszweck „Rechnung RE\_6034214“ nur nach dem Vorkommen der Zahl „6034214“ innerhalb der Zahlungen gesucht.

Es wird sichergestellt, dass nur die Zahlungen gefunden werden, die die exakte Zahl enthalten. Die Zahl darf dabei nicht als Teil einer anderen Zahl enthalten sein.

Dafür wurde folgender regulärer Ausdruck definiert, der nach einer Zahlung mit der Zahl 6034214 sucht:

```
(.*[^\\d]|^)^6034214([\\d].*|$)
```

Mit dem Ausdruck wird definiert, dass die Zeichenkette mit beliebigen Zeichen (.\* ) beginnen kann, diese Zeichen dürfen allerdings unmittelbar vor dem gesuchten Begriff keine anderen Zahlen enthalten ([^\\d]). Die Zeichenkette kann jedoch auch direkt mit der suchenden Zahl 6034214 beginnen (^).

Nach der gesuchten Zahl dürfen unmittelbar danach keine weiteren Zahlen ([\\d]) jedoch das Zeilenende (\$) folgen. Durch das Sicherstellen, dass vor und nach der gesuchten Zahl keine anderen Zahlen vorkommen dürfen, wird verhindert, dass die Zahl innerhalb einer anderen Zahl gefunden wird.

### Unterstützte Indizes

Für die Abfragen mit regulären Ausdrücken werden folgende Indizes unterstützt:

| Operator               | B-Baum | Trigramm (GIN) | Trigramm (GiST) |
|------------------------|--------|----------------|-----------------|
| Regulärer Ausdruck (~) | ✗      | ✓              | ✓               |

Tabelle 25: Unterstützte Indizes bei regulären Ausdrücken (~)

Die im Folgenden aufgeführten Diagramme sind jeweils im größeren Maßstab im Anhang zu finden.



## Operator

~ (Regulärer Ausdruck)

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND documenttext ~  
'(.*[^\\d]|^)^6034214([^\\d].*|$)'
```

## Laufzeitanalyse

Der reguläre Ausdruck wird mit dem Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

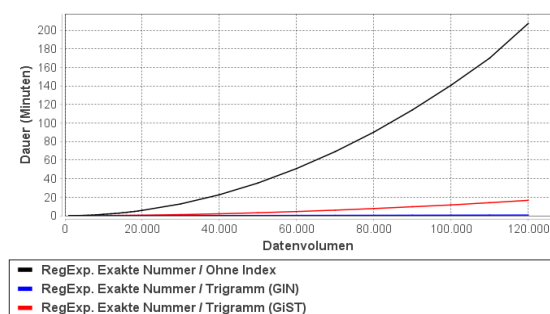


Abbildung 53: Laufzeit mit Operator RegExp., Indizes: Ohne, Trigramm (GIN), Trigramm (GiST)

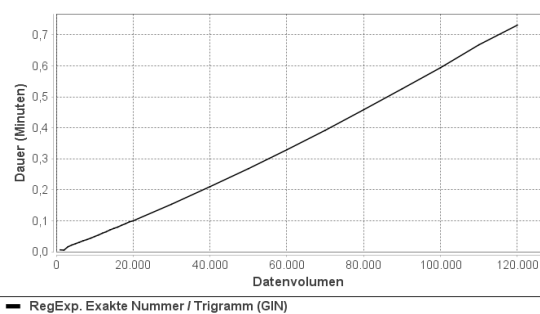


Abbildung 54: Laufzeit mit Operator RegExp., Trigramm (GIN), Trigramm (GiST)

In Abbildung 53 sind die Laufzeiten der Abfragen bei der Verwendung der verschiedenen Indizes gegenübergestellt.

Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 207,36 Minuten (3,46 Stunden) benötigt. Hingegen braucht der Trigramm (GiST)-Index 16,58 Minuten. Die beste Performance ist mit dem Trigramm (GIN)-Index zu erreichen, der 0,73 Minuten benötigt. Sein Verlauf ist in Abbildung 54 nochmals aufgeführt.

Im Vergleich zur Verarbeitung ohne Index wird mit dem Trigramm (GIN)-Index eine Beschleunigung um den Faktor 284 erreicht.

## Laufzeit

Ohne- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

Trigramm (GIN)-Index: leicht überlinear  $O(N \log N)$

## Maximale Verarbeitung innerhalb 30 Minuten

Da die beste Performance mit dem Trigramm (GIN)-Index zu erzielen ist, wird er auf seine maximale Verarbeitung untersucht:

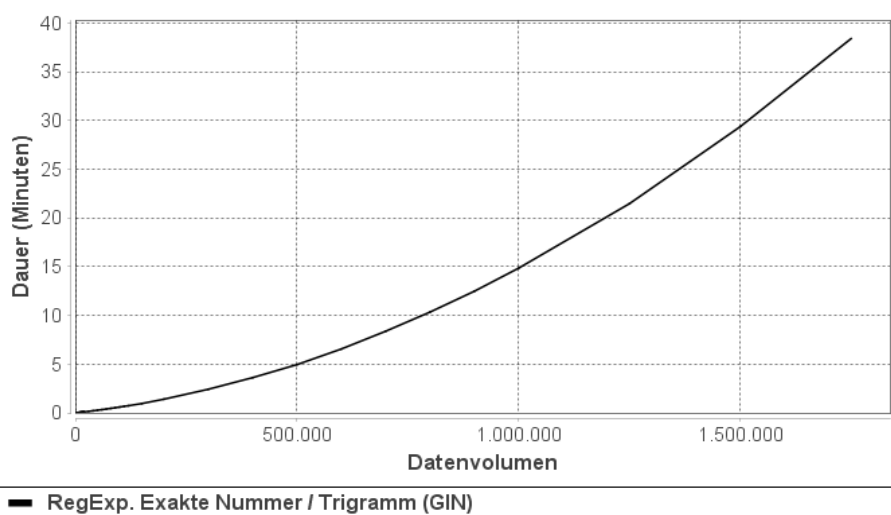


Abbildung 55: Maximale Laufzeit mit Operator RegExp., Index: Trigramm (GIN)

Aus Abbildung 55 ist zu entnehmen, dass innerhalb von 30 Minuten ca. 1.500.000 Datensätze bearbeitet werden können.

## Analyse der Zuordnungsmöglichkeiten

Bei den Abfragen wird geprüft, ob der Verwendungszweck einer Zahlung eine bestimmte Zahl enthält.

Ist z.B. der Verwendungszweck der Rechnung „Rechnung-21061983“, wird eine Abfrage definiert, die nach der Zahl „21061983“ innerhalb des Verwendungszwecks der Zahlungen sucht. So kann durch die Abfrage eine Zuordnung zu der Zahlung „Vielen Dank. 21061983-R“ erfolgen, da in beiden Verwendungszwecken die gleiche Zahl (21061983) enthalten ist. Sämtliche Texte und Zeichen vor und nach den Zahlen werden hierbei ignoriert. In Tabelle 26 sind hierzu weitere erfolgversprechende Zuordnungen aus den Testdaten:

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung        |
|-------------------------------|-------------------------------------|
| Re-8492348298                 | Rechnung 8492348298                 |
| 7238928937                    | Rechnung 7238928937DE. Vielen Dank. |

Tabelle 26: Erfolgversprechende Zuordnungen der Abfrage mit regulärem Ausdruck

Voraussetzung für die ausschließliche Zuordnung über die Zahl ist, dass nach dem Entfernen aller weiteren Zeichen eine eindeutige Zahl entsteht, die als

Zuordnungskriterium verwendet werden kann. Stellen die Zahlen nur in Kombination mit den Zeichen des Verwendungszwecks eine Eindeutigkeit dar, kann keine eindeutige Zuordnung nach dem Entfernen der Zeichen erfolgen. Im folgenden Beispiel sind die Zahlen nur in Kombination mit den Länderkürzeln eindeutig: „DE-5000“ und „AT-5000“. Die Zahl wiederholt sich je Länderkürzel.

Die Zuordnungsstrategie funktioniert solange keine Zeichen zwischen den Zahlen existieren. Der Verwendungszweck „RE-2017-9483“ einer Rechnung würde eine Suche nach der Zahl „20179483“ innerhalb der Verwendungszwecke der Zahlungen durchführen. Somit würde mit diesem Suchverfahren eine Zahlung mit gleichem Verwendungszweck (RE-2017-9483) nicht gefunden werden, da nur in dem Verwendungszweck, mit dem gesucht wird, alle Zeichen bis auf die Zahlen entfernt werden.

## 4.4 Abfragen auf aufbereiteten Daten des Verwendungszwecks

Bei den Abfragen erfolgt die Zuordnung von Rechnungen und Zahlungen mithilfe von aufbereiteten Daten. Bei der Aufbereitung werden sämtliche Zeichen aus dem Verwendungszweck entfernt, die keine Zahlen sind. Die gefundenen Zahlen werden in einer separaten Spalte gespeichert.

Hierfür wird die im Kapitel 3.1 vorgestellte Tabelle „journal“ um eine Spalte erweitert.

| Spalte             | Datentyp      | Beschreibung                               |
|--------------------|---------------|--|
| documentTextNumber | VARCHAR(1000) | Aus dem Verwendungszweck extrahierte Zahl. |

Tabelle 27: Neue Spalte zur Speicherung aufbereiteter Daten

Ein Tupel mit dem Verwendungszweck „RE210683“ wird in der Spalte "documentTextNumber" den Wert 210683 haben. Bei der Suche nach passenden Zahlungen erfolgt die gleiche Aufbereitung auf dem Verwendungszweck der Rechnungen. Auch hier werden alle Zeichen bis auf die Zahlen entfernt. Mit dieser Zahl wird innerhalb der neuen Spalte nach Übereinstimmungen gesucht.

Diese Abfrage ähnelt der Abfrage mit Teilen des Verwendungszwecks, bei der auch nach übereinstimmenden Zahlen gesucht wurde.

Die Aufbereitung der Daten und das Befüllen der Spalte kann durch Trigger oder serverseitig beim Speichern des Tupels erfolgen.

Die Suche innerhalb der neuen Spalte erfolgt mit den Operatoren = (Equal) und LIKE.

### Unterstützte Indizes

Für die Abfrage werden folgende Indizes unterstützt:

| Operator | B-Baum | Trigramm (GIN) | Trigramm (GiST) |
|----------|--------|----------------|-----------------|
| =        | ✓      | ✗              | ✗               |
| LIKE     | ✓      | ✓              | ✓               |

Tabelle 28: Unterstützte Indizes bei exakter Übereinstimmung (extra Spalte)

Die im Folgenden aufgeführten Diagramme sind jeweils im größeren Maßstab im Anhang zu finden.

## Operator

=

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND  
documenttextnumber = '6034214'
```

## Laufzeitanalyse

Der Equal-Operator wird mit dem B-Baum-Index ausgeführt:

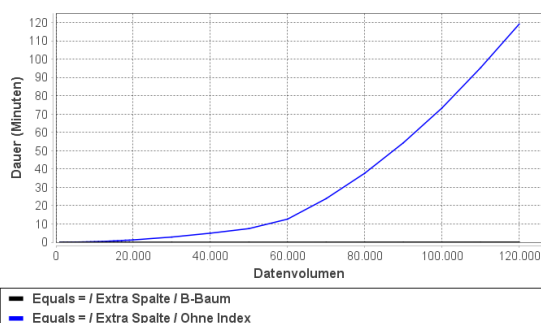


Abbildung 56: Laufzeit mit Operator Equal auf extra Spalte, Indizes: Ohne, B-Baum

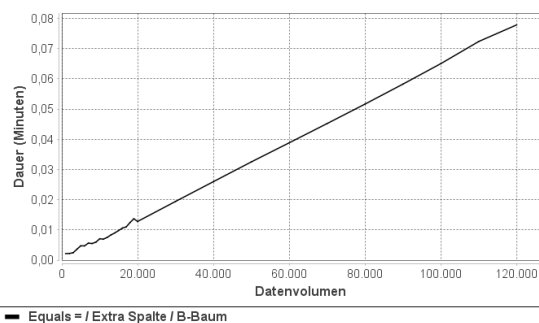


Abbildung 57: Laufzeit mit Operator Equal auf extra Spalte, Index: B-Baum

In Abbildung 56 erfolgt die Gegenüberstellung der Laufzeiten.

Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 119,22 Minuten (1,99 Stunden) benötigt, während mit dem B-Baum-Index 0,08 Minuten benötigt werden. Zur Verdeutlichung des Verlaufs wird in Abbildung 57 die Laufzeit des B-Baum-Index nochmals aufgeführt.

Im Vergleich zur Verarbeitung ohne Index wird mit dem B-Baum-Index eine Beschleunigung um den Faktor 1490 erreicht.

## Laufzeit

Ohne Index: Quadratisch  $O(N^2)$

B-Baum-Index: leicht überlinear  $O(N \log N)$

## Operator

LIKE

## Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND  
documenttextnumber LIKE '6034214'
```

## Laufzeitanalyse

Der LIKE-Operator wird mit dem B-Baum, Trigramm (GIN)- und Trigramm (GiST)-Index ausgeführt:

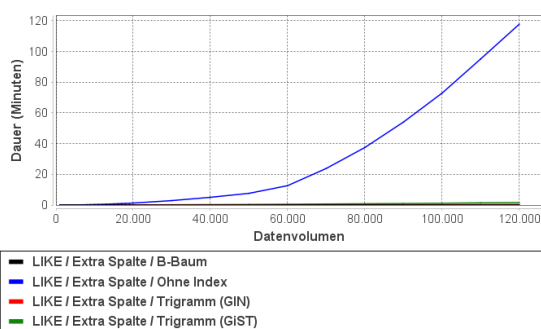


Abbildung 58: Laufzeit mit Operator LIKE auf extra Spalte, Indizes: Ohne, B-Baum, Trigramm (GIN), Trigramm (GiST)

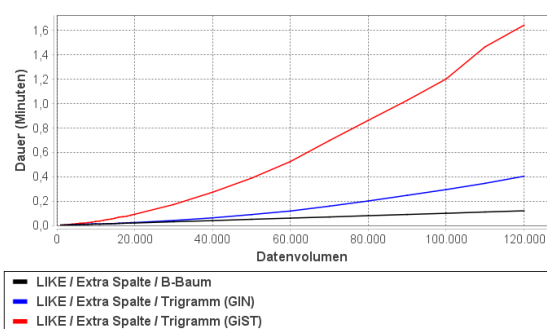


Abbildung 59: Laufzeit mit Operator LIKE auf extra Spalte, Indizes: B-Baum-, Trigramm (GIN), Trigramm (GiST)

In Abbildung 58 erfolgt die Gegenüberstellung der Laufzeiten.

Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 117,63 Minuten (1,96 Stunden) benötigt, wobei der Trigramm (GiST)-Index 1,64 Minuten braucht. Der Trigramm (GIN)-Index kommt auf 0,40 Minuten (Abbildung 59). Die beste Performance ist mit dem B-Baum-Index zu erzielen, der 0,12 Minuten benötigt (Abbildung 60).

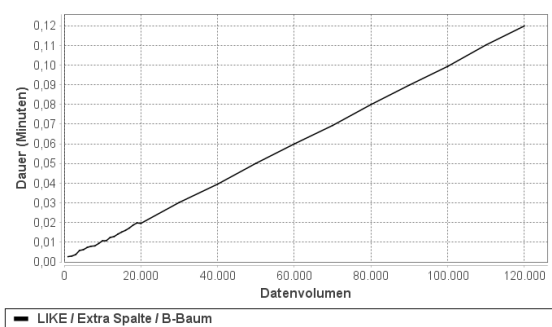


Abbildung 60: Laufzeit mit Operator LIKE auf extra Spalte, Index: B-Baum

Im Vergleich zur Verarbeitung ohne Index wird mit dem B-Baum-Index eine Beschleunigung um den Faktor 980 erreicht.

## Laufzeit

Ohne-, Trigramm (GIN)-, Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

B-Baum-Index: leicht überlinear  $O(N \log N)$

## Gegenüberstellung der Laufzeiten

In Abbildung 61 erfolgt die Gegenüberstellung der Laufzeit der jeweils schnellsten Indizes der jeweiligen Operatoren:

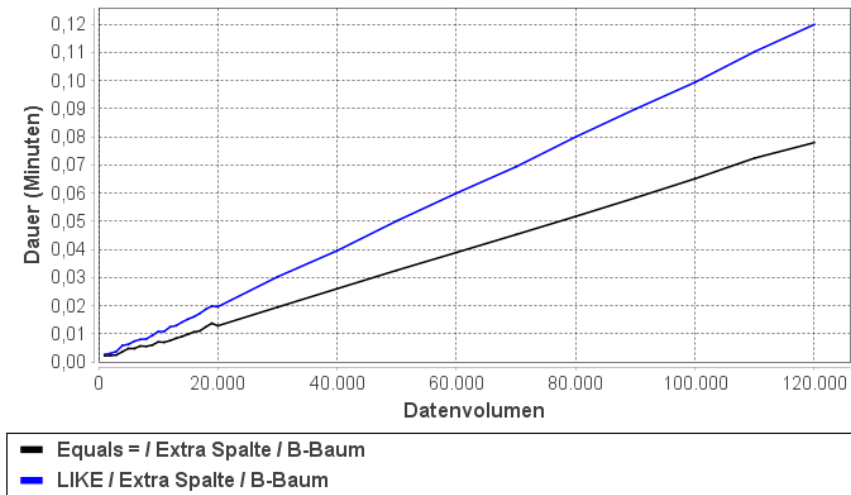


Abbildung 61: Laufzeit auf extra Spalte mit Operator Equal, Index: B-Baum / Operator LIKE, Index: B-Baum

Der Equal-Operator ist mit dem B-Baum-Index schneller als der LIKE-Operator.

## Maximale Verarbeitung innerhalb 30 Minuten

Da der Equal-Operator die beste Performance liefert, wird er auf seine maximale Verarbeitung untersucht.

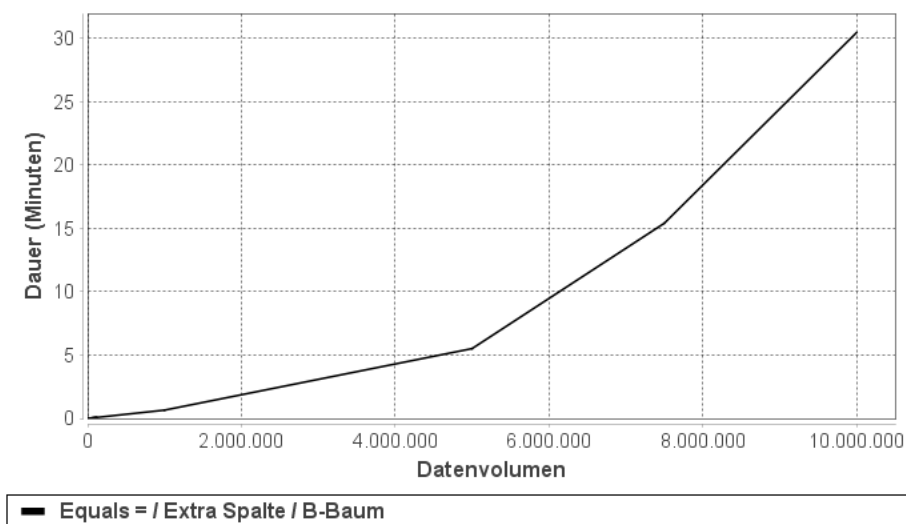


Abbildung 62: Maximale Laufzeit auf extra Spalte mit Operator Equal, Index: B-Baum / Operator LIKE, Index: B-Baum

Mit der Abfrage können in Kombination mit dem B-Baum-Index innerhalb von 30 Minuten ca. 10.000.000 Daten bearbeitet werden (Abbildung 62).

## Analyse der Zuordnungsmöglichkeiten

Bei den Abfragen wird geprüft, ob der Verwendungszweck einer Zahlung nach der Aufbereitung einer bestimmten Zahl entspricht.

Ist z.B. der Verwendungszweck einer Rechnung „Rechnung – 210683“ wird eine Abfrage definiert, die nach der Zahl 210683 innerhalb der neu angelegten Spalte „documentTextNumber“ nach Übereinstimmungen sucht. So kann durch die Abfrage eine Zuordnung zu der Zahlung „Vielen Dank.210683Rechnung“ erfolgen, da in beiden Verwendungszwecken die gleiche Zahl (210683) enthalten ist. Hierzu folgen weitere Beispiele aus den Testdaten.

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung      |
|-------------------------------|-----------------------------------|
| Re-8492348298                 | Rechnung 8492348298               |
| 7238928937                    | Rechnung 7238928937. Vielen Dank. |
| RE-2017-8387-2                | RE 201783872                      |

Tabelle 29: Erfolgversprechende Zuordnungen der Abfrage

Die Zuordnungsstrategie ähnelt der Strategie bei der Abfrage mit Teilen des Verwendungszwecks. Auch hier wird nach dem Vorkommen einer Zahl gesucht.

In der letzten Zeile der Tabelle 29 ist zu sehen, dass hier auch Rechnungen und Zahlungen zugeordnet werden können, wenn Zeichen zwischen den Zahlen im Verwendungszweck vorkommen. Der Grund liegt darin, dass durch die Aufbereitung auf beiden Seiten eine Normalisierung stattfindet und die Werte dadurch miteinander vergleichbar sind.

Voraussetzung ist auch hier, dass nach dem Entfernen aller Zeichen eine eindeutige Zahl entsteht, die als Zuordnungskriterium verwendet werden kann. Beispielhaft würde ein Verwendungszweck der Form „Mayer Rechnung2017Nummer1“ nicht zu einer eindeutigen Zahl führen. Das Format entspricht dem Kundennamen gefolgt vom Geschäftsjahr und einer laufenden Nummer für Rechnungen in dem Jahr. Die daraus resultierende Zahl (20171) kann auf mehrere Zahlungen unterschiedlicher Kunden zutreffen.



## 4.5 Volltextsuche

Es wird überprüft, inwieweit sich die im Kapitel 2.7.5 vorgestellte Volltextsuche und das dabei verwendete Stemming für die Zuordnung von Rechnungen und Zahlungen eignet.

Für die Volltextsuche werden folgende Indizes unterstützt:

| Operator | B-Baum | GIN | GiST |
|----------|--------|-----|------|
| @@       | x      | ✓   | ✓    |

Tabelle 30: Unterstützte Indizes bei der Volltextsuche

Die im Folgenden aufgeführten Diagramme sind jeweils im größeren Maßstab im Anhang zu finden.

### Operator

@@

### Beispielabfrage

```
SELECT * FROM journal WHERE type = 'ZAHLUNG' AND  
to_tsvector('german', coalesce(documenttext, '')) @@  
to_tsquery('german', 'RE-238843')
```

### Laufzeitanalyse

Die Volltextsuche wird mit dem GIN- und GiST-Index ausgeführt:

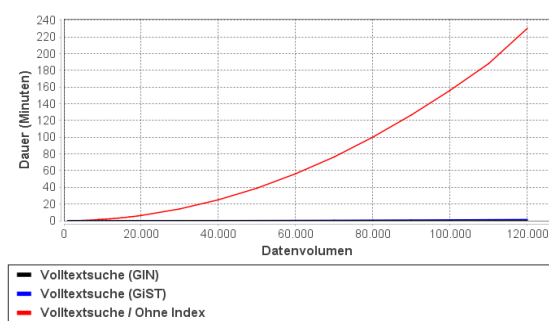


Abbildung 63: Laufzeit der Volltextsuche, Indizes: Ohne, GIN, GiST

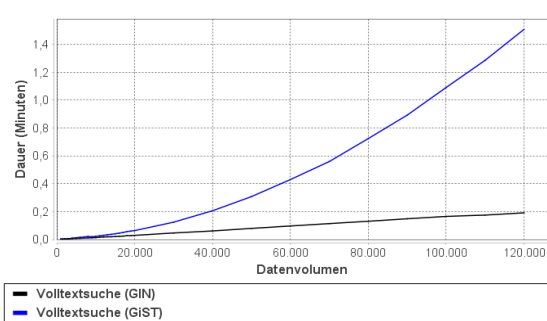


Abbildung 64: Laufzeit der Volltextsuche, Indizes: GIN, GiST

In Abbildung 63 erfolgt die Gegenüberstellung der Laufzeiten. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 230,14 Minuten (3,84 Stunden) benötigt, während der GiST-Index 1,51 Minuten braucht. Sein Verlauf ist in Abbildung 64 besser zu erkennen.

Die beste Performance ist mit dem GIN-Index zu erzielen, der 0,19 Minuten benötigt (Abbildung 65). Im Vergleich zur Verarbeitung ohne Index wird mit dem (GIN)-Index eine Beschleunigung um den Faktor 1211 erreicht.

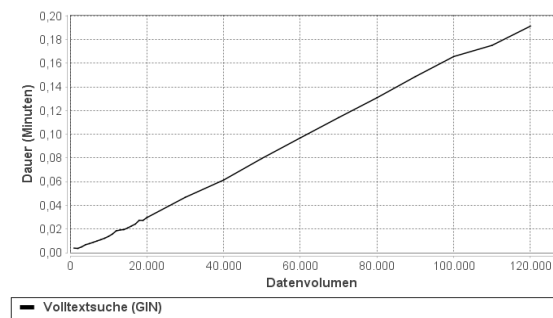


Abbildung 65: Laufzeit der Volltextsuche, Indizes: GIN

## Laufzeit

Ohne- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

Trigramm (GIN)-Index: leicht überlinear  $O(N \log N)$

## Maximale Verarbeitung innerhalb 30 Minuten

Da die beste Performance mit dem GIN-Index zu erzielen ist, wird die maximale Verarbeitung untersucht.

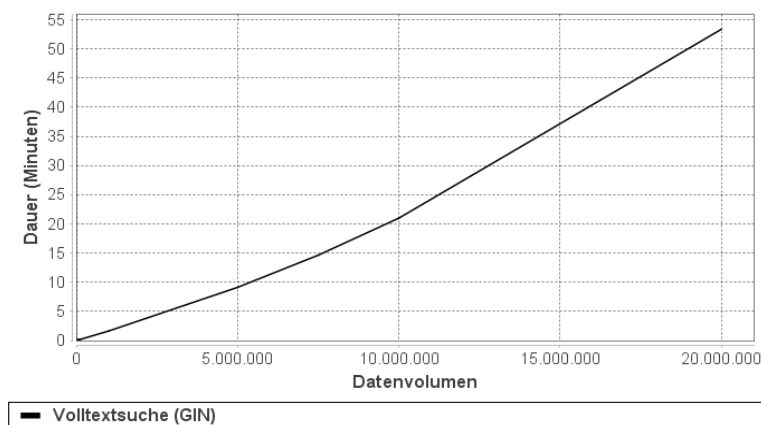


Abbildung 66: Maximale Laufzeit der Volltextsuche, Index: Trigramm (GIN)

Aus Abbildung 66 ist zu entnehmen, dass innerhalb einer halben Stunde ca. 13.000.000 Daten bearbeitet werden können.

## Analyse der Zuordnungsmöglichkeiten

Die Zuordnungsmöglichkeiten ähneln denen der Suche unter Auslassung von führenden und endenden Zeichen. Die Zuordnungsstrategie funktioniert auch dann, wenn das Format des Verwendungszwecks nicht sicherstellt, dass ein Verwendungszweck nicht als Teil eines anderen Verwendungszwecks vorkommt.

Wie gut Abweichungen abgefangen werden können, ist unter anderem von den verwendeten Zeichen im Verwendungszweck abhängig. Es werden zuerst exemplarisch erfolgversprechende Zuordnungen aus den Testdaten gezeigt (Tabelle 31).

| Verwendungszweck der Rechnung | Verwendungszwecke der Zahlung   |
|-------------------------------|---------------------------------|
| PFB089131                     | Rechnung pfb089131 Vielen Dank! |
| r*6706102                     | R*6706102 IJÜJKJdHB             |
| RECHNUNGEN#3501404            | Rechnung RE\3501404             |

Tabelle 31: Erfolgversprechende Zuordnungen der Volltextsuche

Im Folgenden werden die Möglichkeiten und Grenzen des Stemming gezeigt.

### Gelungene Zuordnung trotz Abweichung

Es erfolgt die Zuordnung der Verwendungszwecke:

„RECHNUNGEN#3501404“ und „Rechnung RE\3501404“.

```
testdb=# SELECT to_tsquery('german','RECHNUNGEN#3501404');
           to_tsquery
-----
'rechnung' & '3501404'
(1 Zeile)
```

Abbildung 67: tsquery-Erzeugung des Verwendungszwecks „RECHNUNG#3501404“

Aus dem Verwendungszweck der Rechnung „RECHNUNGEN#3501404“ wird eine tsquery erzeugt (Abbildung 67). Dabei werden zwei Token gebildet, die mit dem logischen Operator „und“ (&) verknüpft sind. Mit der tsquery wird demnach nach Zahlungen gesucht, die beide Token besitzen.

Aus dem Verwendungszweck der Zahlung wird ein tsvector erstellt, der aus drei Token besteht (Abbildung 68).

```
testdb=# SELECT to_tsvector('german','Rechnung RE\3501404');
           to_tsvector
-----
'3501404':3 're':2 'rechnung':1
(1 Zeile)
```

Abbildung 68: to\_tsvector-Erzeugung des Verwendungszwecks „Rechnung RE\3501404“

Da beide Token der Rechnung in dem tsvector der Zahlung vorkommen, liefert die Abfrage in Abbildung 69 „t“ (true) zurück. Die Zuordnung zwischen Rechnung und Zahlung wird erfolgreich durchgeführt.

```
testdb=# SELECT to_tsvector('german','Rechnung RE\3501404') @@ to_tsquery('german','RECHNUNGEN#3501404');
?column?
-----
t
(1 Zeile)
```

Abbildung 69: Volltextsuche der Zahlung Rechnung „RE\3501404“ und Rechnung „RECHNUNG#3501404“

## Misslungene Zuordnung

Zuordnung der Verwendungszwecke: RECHNUNG.3501404 und Rechnung\3501404.

Mit dem Verwendungszweck der Rechnung „RECHNUNG.3501404“ wird eine tsquery erzeugt (Abbildung 70). Es ist zu sehen, dass der Parser aus dem

```
testdb=# SELECT to_tsquery('german', 'RECHNUNG.3501404');
to_tsquery
-----
'rechnung.3501404'
(1 Zeile)
```

Abbildung 70: tsquery-Erzeugung des Verwendungszwecks „RECHNUNG.3501404“

Verwendungszweck nur einen Token generiert. Informationen über die Aufspaltung der Token werden mit der Funktion ts\_debug abgerufen (Abbildung 71).

```
testdb=# SELECT * FROM ts_debug('german', 'RECHNUNG.3501404');
 alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
 file  | File or path name | RECHNUNG.3501404 | {simple} | simple | {rechnung.3501404}
(1 Zeile)
```

Abbildung 71: ts\_debug-Information des Verwendungszwecks „RECHNUNG.3501404“

Durch den Punkt im Verwendungszweck wird der Token dem Typ „file“ („File or path name“) zugeordnet. Er wird nicht als Trenner (blank) erkannt, wie es im Folgenden bei dem Verwendungszweck der Zahlung erfolgt.

Bei der Erzeugung des tsvector der Zahlung werden zwei Token gebildet (Abbildung 72).

```
testdb=# SELECT to_tsvector('german', 'Rechnung\3501404');
to_tsvector
-----
'3501404':2 'rechnung':1
(1 Zeile)
```

Abbildung 72: tsvector-Erzeugung des Verwendungszwecks „Rechnung\3501404“

Mit der Funktion ts\_debug werden die Typen der Token ermittelt (Abbildung 73).

```
testdb=# SELECT * FROM ts_debug('german', 'Rechnung\3501404');
 alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
 asciiword | Word, all ASCII | Rechnung | {german_stem} | german_stem | {rechnung}
 blank     | Space symbols   | \       | {}            |             | {}
 uint      | Unsigned integer | 3501404 | {simple}       | simple      | {3501404}
(3 Zeilen)
```

Abbildung 73: ts\_debug-Information des Verwendungszwecks „Rechnung\3501404“

Dabei wurde der Backslash (\) dem Typ „blank“ (Space symbols) zugeordnet.

Die Suche mit dem tsquery innerhalb des tsvector ergibt keinen Treffer, da der Token der Rechnung „rechnung.3501404“ nicht im tsvector der Zahlung vorkommt (Abbildung 74). Würde der Punkt ebenfalls als „blank“ erkannt werden, so könnte eine Zuordnung erfolgen. Beide Verwendungszwecke unterscheiden sich nur durch unterschiedliche Trenner.

```
testdb=# SELECT to_tsvector('german','RECHNUNG\3501404') @@ to_tsquery('german', 'RECHNUNG.3501404');
?column?
-----
f
(1 Zeile)
```

Abbildung 74: Volltextsuche der Zahlung Rechnung „RECHNUNG.3501404“ und Zahlung „RECHNUNG\3501404“

Das Abfangen von Abweichungen funktioniert solange keine Zeichen im Verwendungszweck vorkommen, die unterschiedlichen Typen zugeordnet werden. Aufgrund der abweichenden Typen kann es dazu kommen, dass die Aufspaltung der Token, wie im letzten Beispiel, unterschiedlich ausfällt. Zeichen, die nicht als „blank“ geparkt werden, sind unter anderem: Minus(-), Punkt (.), Slash (/).

Zudem ist darauf zu achten, dass keine Wörter im Verwendungszweck zur Unterscheidung genutzt werden, die durch das Stemming normalisiert werden. Zum Beispiel werden die Wörter „Rechnungs“ und „Rechnungen“ zum Lexem „rechnung“ normalisiert.

## 4.6 Ähnlichkeitsüberprüfung

Es wird überprüft, inwieweit sich die Ähnlichkeitsüberprüfung auf Basis der `word_similarity()`-Funktion des Trigramm-Moduls, die in Kapitel 2.7.6 vorgestellt wurde, für die Zuordnung von Rechnungen und Zahlungen eignet.

Hierfür wird eine Abfrage mit dem Verwendungszweck der Rechnung definiert, die innerhalb der Zahlungen nach dem Verwendungszweck mit der „kleinsten Entfernung“ sucht.

Es werden alle Treffer berücksichtigt, die eine Entfernung von höchstens 0,4 besitzen. Das Ergebnis wird nach dem Wert aufsteigend sortiert und nur der Tupel mit dem niedrigsten Wert zurückgeliefert.

Für die Abfragen werden folgende Indizes unterstützt:

| Operator | B-Baum | Trigramm (GIN) | Trigramm (GiST) |
|----------|--------|----------------|-----------------|
| <<->     | ✗      | ✗              | ✓               |

Tabelle 32: Unterstützte Indizes für die Similarity-Überprüfung

Die im Folgenden aufgeführten Diagramme sind jeweils im größeren Maßstab im Anhang zu finden.

## Operator

<<->

## Beispielabfrage

```
SELECT *, 'RE-289512'::varchar <<-> documenttext AS dist FROM  
journal WHERE type='ZAHLUNG' ORDER BY dist LIMIT 1;
```

## Laufzeitanalyse

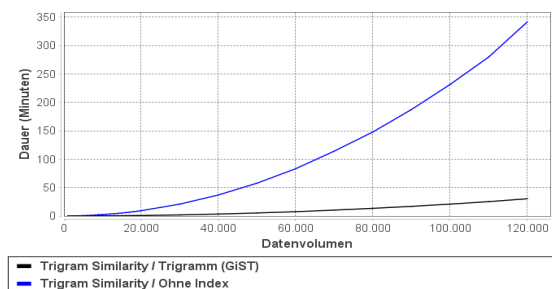


Abbildung 75: Laufzeit mit Similarity-Operator, Indizes:  
Ohne, Trigramm (GiST)

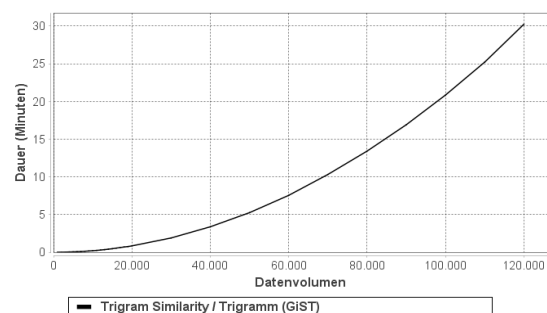


Abbildung 76: Laufzeit mit Similarity-Operator, Indizes:  
Trigramm (GiST)

In Abbildung 75 erfolgt die Gegenüberstellung der Laufzeiten. Für die Bearbeitung von 120.000 Datensätzen werden ohne Index 341,43 Minuten (5,69 Stunden) benötigt. Hingegen werden mit dem Trigramm (GiST)-Index 30,25 Minuten benötigt. Zur Verdeutlichung wird sein Verlauf in Abbildung 76 nochmals dargestellt.

Im Vergleich zur Verarbeitung ohne Index entspricht dies einer Beschleunigung um den Faktor 11.

## Laufzeit

Ohne- / Trigramm (GiST)-Index: Quadratisch  $O(N^2)$

## Maximale Verarbeitung innerhalb 30 Minuten

Mit dem Trigramm (GiST)-Index können innerhalb von 30 Minuten ca. 120.000 Daten bearbeitet werden.

## Analyse der Zuordnungsmöglichkeiten

Existiert in der Menge der Zahlungen eine exakte Übereinstimmung, so wird diese durch die Abfrage gefunden (Tabelle 33).

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung |
|-------------------------------|------------------------------|
| Rechnung re.5325606           | Rechnung re.5325606          |
| r7579904                      | r7579713                     |

*Tabelle 33: Erfolgversprechende Zuordnung durch die Abfrage auf Ähnlichkeit*

Durch die Suche wird jedoch immer maximal nur eine passende Zahlung zurückgeliefert. Wurde eine Rechnung mit mehreren Zahlungen beglichen, können diese nicht mit einer Abfrage gefunden werden.

Gibt es keine eindeutige Zuordnung, werden Ergebnisse zurückgeliefert, die sich nur in gewissen Merkmalen unterscheiden. In Tabelle 34 sind einige Beispiele aufgeführt.

| Verwendungszweck der Rechnung | Verwendungszweck der Zahlung |
|-------------------------------|------------------------------|
| r/2817695                     | R/281769                     |
| Rechnung RE\1412107           | Rechnung RE 12107            |

*Tabelle 34: Fehlzusordnungen von Rechnungen und Zahlungen. Operator: Similarity*

## 4.7 Größenentwicklung der Indizes

In Abbildung 77 ist die Größenentwicklung der Indizes bei skalierenden Daten angegeben.

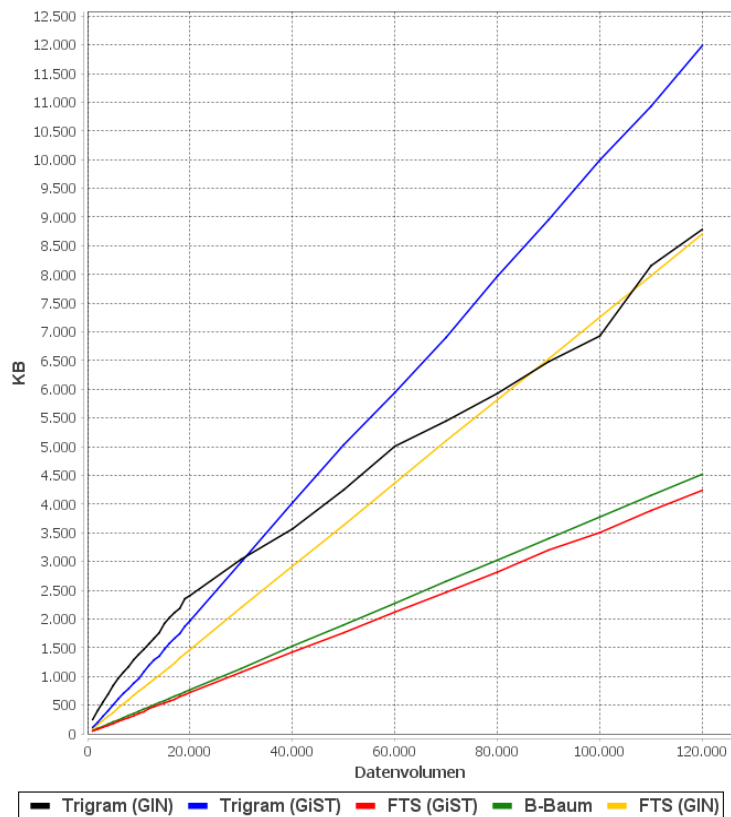


Abbildung 77: Indexgrößenentwicklung bei skalierenden Daten

Der FB-B-Baum-Index ist aufgrund der Übersichtlichkeit nicht aufgeführt, da sich sein Verlauf mit dem B-Baum-Index deckt.

Es ist zu entnehmen, dass alle Indizes bis auf den Trigramm (GIN)-Index ein lineares Wachstum haben.

Der Trigram (GIN)-Index hat eine effiziente Art, doppelte Werte zu speichern.

Es ist damit zu rechnen, dass bei steigender Anzahl der Daten die Speichergröße allmählich abnimmt.

Das liegt daran, dass mit steigenden Daten sich bestimmte Trigramme wiederholen und somit lediglich die Posting List des Trigramms um die neue Adresse des Tupels erweitert wird.



## 4.8 Zusammenfassung Kapitel 4

In diesem Kapitel wurden Abfragen aus den bei der Einführung des Kapitels vorgestellten Abfragegruppen durchgeführt.

Dabei wurden diese mit den jeweils unterstützten Indizes ausgeführt, wobei die Laufzeiten gegenübergestellt und anhand der O-Notation kategorisiert wurden. Nach jeder Abfragegruppe erfolgte eine Gegenüberstellung der Laufzeiten sowie eine Analyse der Zuordnungsmöglichkeiten.

Bei der Gegenüberstellung der Laufzeiten wurden die schnellsten Indizes der jeweiligen Operatoren verglichen. Für den dabei ermittelten schnellsten Index wurde zusätzlich die maximale Verarbeitung innerhalb einer Zeitspanne von dreißig Minuten bestimmt.

Die Analyse der Zuordnungen untersuchte, welche Zuordnung mit dem jeweiligen Abfragetyp erzielt werden kann und welche Bedingung der Verwendungszweck der Rechnung erfüllen muss, um eine bestmögliche Zuordnung zu ermöglichen.

Abschließend wurde die Größenentwicklung der einzelnen Indizes bei skalierenden Daten gegenübergestellt.

Es stehen nun alle Daten zur Verfügung, um im folgenden Kapitel Handlungsanweisungen in Form von Zuordnungsstrategien sowie Formatvorschläge für den Verwendungszweck wie auch allgemeine Erkenntnisse über Indizes zusammenzutragen.

## 5 Ergebnis mit Ausblick

Ziel der Arbeit war es, zu überprüfen, wie die Zuordnung von Rechnungen und Zahlungen ausschließlich über den Verwendungszweck durch Indizes beschleunigt und durch Teilzeichenketten-Suchstrategien optimiert werden kann.

Hierfür wurden in Kapitel 2.5 unterschiedliche Indizes sowie in Kapitel 2.7 verschiedene SQL-Operatoren behandelt. Um diese auf ihre Performance und Stringmatching Möglichkeiten zu untersuchen, wurden in Kapitel 3 Testdaten und -fälle erstellt. In Kapitel 4 wurden innerhalb der Testumgebung anhand der Operatoren konkrete Abfragen formuliert, wobei die unterschiedlichen Indizes zum Einsatz kamen.

In diesem Kapitel wird aus den Messungen des 4. Kapitels eine Zuordnungsstrategie und Handlungsanweisungen abgeleitet sowie Vorschläge für das Format des Verwendungszwecks gemacht. Unabhängig von der Problemstellung werden Erkenntnisse darüber gewonnen, welcher Operator mit welchem Index die beste Performance liefert. Abschließend erfolgt der Ausblick.

### 5.1 Zuordnungsstrategien

Die zu verwendende Zuordnungsstrategie hängt sowohl von dem Format des Verwendungszwecks als auch von den zu tolerierenden Abweichungen vom ursprünglichen Verwendungszweck ab, die umgangen werden sollen.

Grundsätzlich ist eine Case Insensitiv- der Case Sensitive-Abfrage vorzuziehen, um Abweichungen in der Groß- und Kleinschreibung zu ignorieren. Wie aus den Messungen zu entnehmen ist, kann eine Case Insensitive-Suche ohne Performanceeinbuße durchgeführt werden. Die ermittelte Größenentwicklung der Indizes beeinflusst nicht die Wahl des passenden Indexes.

Im Folgenden werden Zuordnungsstrategien und ihre Realisierung vorgestellt:

#### **Zuordnung bei exakter Übereinstimmung des Verwendungszwecks**

Ist eine Zuordnung von Rechnungen und Zahlungen nur über die exakte Übereinstimmung des Verwendungszwecks gewünscht, empfiehlt sich der Funktionsbasierte B-Baum-Index unter Verwendung des Equal-Operators. Die einzige Abweichung, die dabei abgefangen werden kann, ist die Ignorierung der Groß- und

Kleinschreibung (Case Insensitive-Suche). Innerhalb einer halben Stunde werden bis zu 30 Millionen Daten bearbeitet (Kapitel 4.2.1).

Soll darüber hinaus auch abweichender Text vor oder nach dem Verwendungszweck umgangen werden, ist die Umsetzung der folgenden Zuordnungsstrategien zu überprüfen.

### **Zahl als ausschließliches Zuordnungskriterium**

Sind Zahlen im Verwendungszweck vorhanden und sind diese als ausschließliches Zuordnungskriterium verwendbar, so ist die Zuordnung von Rechnungen und Zahlungen über diese zu empfehlen.

Sind die Zahlen im Verwendungszweck nicht durch Zeichen getrennt und beläuft sich die Datenmenge auf weniger als 1,5 Millionen Datensätze, ist die Suche nach dem exakten Vorkommen der Zahl mit dem regulären Ausdruck und dem Trigramm (GIN)-Index zu empfehlen (Kapitel 4.3).

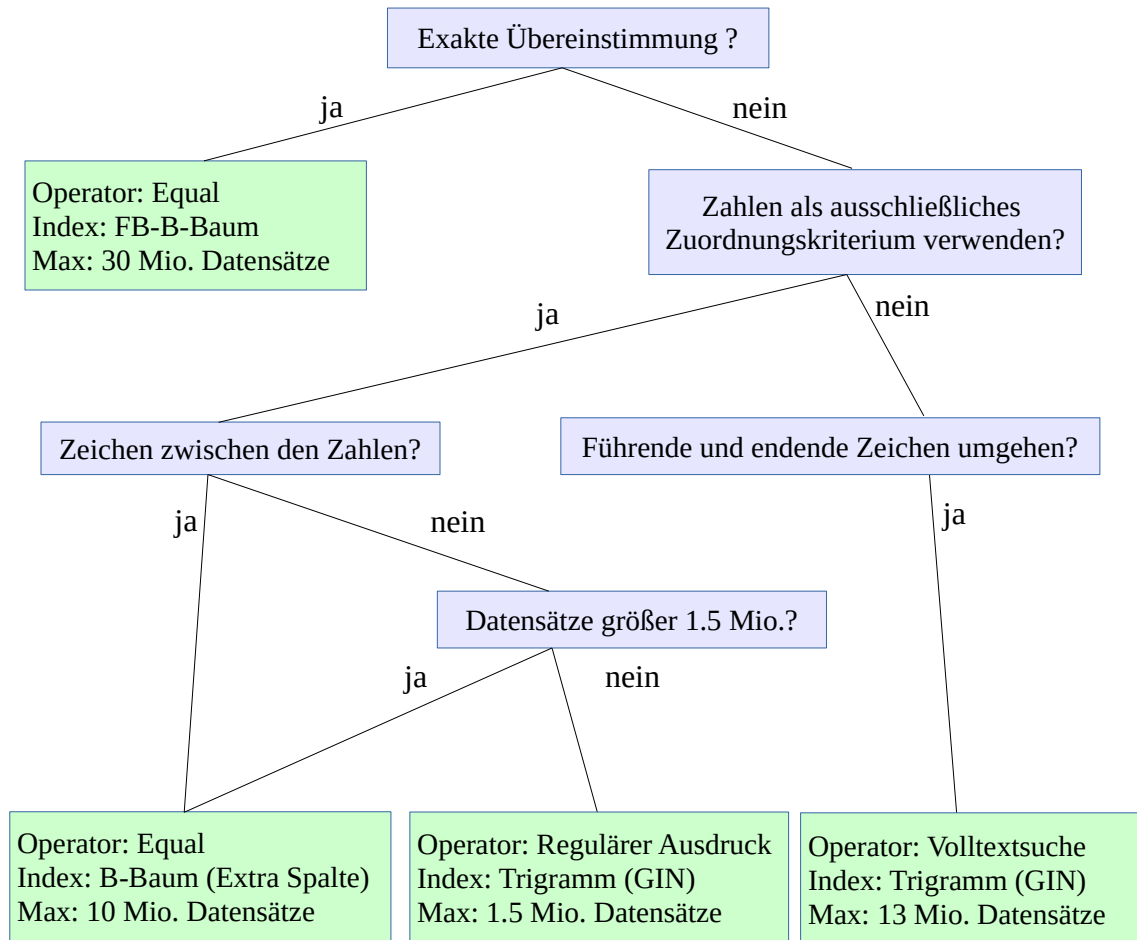
Müssen hingegen Datenmengen im Bereich von 10 Millionen bearbeitet werden oder sind Zeichen zwischen den Zahlen vorhanden, sind die Zahlen aus dem Verwendungszweck zu extrahieren und in einer separaten Spalte zu speichern und mit einem B-Baum-Index zu versehen (Kapitel 4.4). Die neu erstellte Spalte muss hierbei mit den Zahlen des Verwendungszwecks befüllt werden. Dies kann durch Trigger (Kapitel 2.1.3) oder serverseitig vor dem Speichern des Tupels erfolgen.

### **Zuordnung unter Auslassung führender und endender Zeichen**

Enthält der Verwendungszweck keine Zahlen oder bilden die vorhandenen Zahlen kein eindeutiges Zuordnungskriterium, ist eine Zuordnung von Rechnungen und Zahlungen über den vollständigen Verwendungszweck unter Auslassung führender und endender Zeichen zu empfehlen. Hierbei sollte die Wahl auf die Volltextsuche in Kombination mit dem GIN-Index fallen. Durch das Stemming können zusätzlich Abweichungen in der Schreibweise sowie auch Abweichungen in der Groß- und Kleinschreibung abgefangen werden. Innerhalb einer halben Stunde können 13 Millionen Datensätze bearbeitet werden (Kapitel 4.5).

## Entscheidungsbaum

Aus den eben vorgestellten Zuordnungsstrategien lässt sich folgender Entscheidungsbaum ableiten:



### Nur führende oder nur endende Zeichen auslassen

Ist es hingegen explizit gewünscht, dass nur führende oder nur endende Zeichen ausgelassen werden, ist Folgendes zu empfehlen:

Für die Auslassung führender Zeichen ist der ILIKE Operator mit dem Trigramm (GIN)-Index zu wählen. Innerhalb einer halben Stunde können 450.000 Datensätze bearbeitet werden (Kapitel 4.2.2).

Für die Auslassung endender Zeichen ist der LIKE-Operator in Kombination mit dem Funktionsbasierten B-Baum-Index zu empfehlen. Innerhalb einer halben Stunde können 15 Millionen Datensätze bearbeitet werden (Kapitel 4.2.3).

Beide Abfragen sind Case Insensitive.

## **Bewertung der Ähnlichkeitsüberprüfung**

Da die Suche der Ähnlichkeitsüberprüfung (Kapitel 4.6) maximal nur eine passende Zahlung zu einer Rechnung liefert, kann der Anwendungsfall „Auszifferung einer Rechnung durch mehrere Zahlungen“ nicht realisiert werden.

Exakte Übereinstimmungen werden zwar gefunden, bei Abweichungen wird jedoch ein unzureichender Vorschlag geliefert.

## **5.2 Formatvorschläge für den Verwendungszweck**

Bei der Testdurchführung und der jeweiligen Analyse der Zuordnungsmöglichkeiten haben sich Eigenschaften für das Format des Verwendungszwecks herauskristallisiert, die die Auszifferung begünstigen. Diese Eigenschaften gelten strategieübergreifend. Das Einhalten der Eigenschaften ermöglicht eine möglichst flexible Auswahl und ggf. einen Wechsel der Zuordnungsstrategie.

Folgende Eigenschaften sollte ein Verwendungszweck aufweisen:

- Kurzer Verwendungszweck

Die Länge des Verwendungszwecks sollte möglichst kurz sein. Eine lange Zeichenkette ist fehleranfällig und unkomfortabel.

- Feste Länge der Zahlen

Die Zahl sollte in einer festen Länge angegeben sein, z.B. in der Länge acht: RE-00000001, DE00002345

Dadurch wird sichergestellt, dass die Zahl nicht als Teil einer anderen Zahl gefunden wird.

- Zahl als eindeutiges Zuordnungskriterium

Die Zahl im Verwendungszweck sollte eindeutig sein. Die Eindeutigkeit sollte nicht nur in Kombination mit den restlichen Zeichen des Verwendungszwecks gegeben sein.

- Keine Zeichen zwischen den Zahlen

Die Zahlen sollten möglichst am Stück und ohne Unterbrechung von Zeichen erfolgen. Dies ermöglicht eine größere Flexibilität bei den Zuordnungsstrategien mittels der Zahlen.

- Groß- und Kleinschreibung kein Unterscheidungsmerkmal

Die Groß- und Kleinschreibung sollte kein Unterscheidungsmerkmal sein.

„2106A83de“ und „re2106a83DE“ sollen gleiche Rechnungen darstellen.

- Keine Sonderzeichen

Es sollten keine Sonderzeichen eingesetzt werden. Es sollte sich auf einfache Buchstaben und Zahlen beschränkt werden.

## 5.3 Erkenntnisse über Indizes

Bei den durchgeführten Tests haben sich neben den Zuordnungsmöglichkeiten auch grundlegende Erkenntnisse darüber herausgestellt, welcher Operator mit welchem Index die beste Performance liefert. Im Folgenden werden diese aufgeführt, wobei auch auf die Größenentwicklung des Index eingegangen wird.

### B-Baum - Index

Wenn der B-Baum Index eingesetzt werden kann, erzielte er bei allen Abfragen die beste Performance. Der B-Baum-Index kann für Equal- und LIKE-Operatoren eingesetzt werden. Der LIKE Operator wird unterstützt, solange keine führenden Platzhalter verwendet werden. Beide Operatoren führen eine Case Sensitive-Suche aus. Ist eine Case Insensitive-Suche gewünscht, kann diese bei annähernd gleicher Performance mit dem Funktionsbasierten B-Baum-Index erzielt werden.

Der Speicherbedarf vom B-Baum- und dem FB-B-Baum-Index steigt bei skalierenden Daten linear an.

### Trigramm (GIN) - Index

Aus den Messungen kam hervor, dass der Trigramm (GIN) Index für Abfragen, bei denen der B-Baum nicht genutzt werden kann, die beste Performance liefert. Hierzu zählen der ILIKE-Operator sowie der LIKE-Operator mit führenden Platzhaltern. Abfragen mit regulärem Ausdruck können ebenfalls deutlich beschleunigt werden.

Der Speicherbedarf des GIN-Index steigt unterlinear mit der Datenmenge, dies ist auf die optimierte Speicherung mehrfach vorkommender Trigramme zurückzuführen.

### **Volltextsuche (GIN) – Index**

Für die Volltextsuche liefert der GIN-Index die beste Performance.

Der Speicherbedarf des GIN-Indexes bei der Volltextsuche steigt bei skalierenden Daten linear an.

### **Trigramm (GiST) – Index**

Bei der Ähnlichkeitsüberprüfung bietet der Trigramm (GiST)-Index die beste Performance. Bei dieser Überprüfung geht es darum, Zeichenketten mit der kürzesten „Entfernung“ zu einem Suchwort zu ermitteln.

Der Speicherbedarf des Trigramm (GiST)-Indexes steigt bei skalierenden Daten linear an.

## **5.4 Ausblick**

In der Bachelorarbeit wurde die Zuordnung von Rechnungen und Zahlungen ausschließlich über den Verwendungszweck mittels Indizes beschleunigt und durch Teilzeichenketten-Suchstrategien optimiert. Es wurden Handlungsanweisungen und Zuordnungsstrategien in Abhängigkeit des vorliegenden Verwendungszwecks und der Datenmenge erarbeitet sowie Erkenntnisse über Indizes in Kombination mit den Operatoren zusammengetragen.

Diese Erkenntnisse können nun in die automatische Methode der Auszifferung der Scopevisio-Anwendung übernommen werden und mit den übrigen Kriterien der Zuordnung, wie z.B. dem Rechnungsbetrag oder -datum kombiniert werden. Die Ergebnisse können auch anderen Unternehmen dabei helfen, für ihr jeweiliges Format des Verwendungszwecks und einer bestimmten Datenmenge die richtige Zuordnungsstrategie als auch den richtigen Index für die Zuordnung zu finden.

Um die Volltextsuche bezüglich des Abfangens von Abweichungen zu verbessern, wäre die Implementierung eines eigenen Parsers denkbar. Dadurch könnte Einfluss auf die Token-Zerlegung genommen werden.

Die Auswertungen können in Zukunft dahingehend erweitert werden, dass neue Indizes mit ggf. neuen Operatoren getestet werden. Als Optimierung könnte der Testverlauf, der in Kapitel 3.3.4 vorgestellt wurde, automatisiert werden.

Die Ergebnisse der Arbeit können auch auf ähnliche Problemfälle, wie z.B. der Duplikaterkennung von Kontakten, übertragen werden.

## 6 Quellenverzeichnis

- Berkeley University of California, 1999: GiST: A Generalized Search Tree for Secondary Storage; <http://gist.cs.berkeley.edu/gist1.html>. Abgerufen am 04.06.2017
- Bayer, 2001/2002: B-Bäume, <http://www.bayer.in.tum.de/lehre/WS2001/HSEM-bayer/BTreesAusarbeitung.pdf>. Abgerufen am 07.05.2017
- Belaid, 2015: Introduction to PostgreSQL physical storage. <http://rachbelaid.com/introduction-to-postgres-physical-storage/>. Abgerufen am 13.05.2017
- Connolly, Thomas, Begg, Carolyn und Strachan, Anne: Database Systems – A Practical Approach to Design, Implementation, and Management, 1. Auflage, München (Addison Wesley) 2002.
- FH Köln, Funktionsbasierter Index, [http://wikis.gm.fh-koeln.de/wiki\\_db/Datenbanken/Funktionsbasierter-Index](http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Funktionsbasierter-Index). Abgerufen am 20.05.2017
- PostgreSQL, 1996-2017: PostgreSQL 9.6.3 Documentation / 8.11 Text Search Types. <https://www.postgresql.org/docs/9.6/static/datatype-textsearch.html>. Abgerufen am 13.05.2017
- PostgreSQL, 1996-2017: PostgreSQL 9.6.3 Documentation / 12.1 Introduction. <https://www.postgresql.org/docs/9.6/static/textsearch-intro.html>. Abgerufen am 13.05.2017.
- PostgreSQL, 1996-2017: PostgreSQL 9.6.3 Documentation / 12.1.3 Configurations. <https://www.postgresql.org/docs/9.6/static/textsearch-intro.html>. Abgerufen am 13.05.2017
- PostgreSQL, 1996-2017: PostgreSQL 9.6.3 Documentation / 12.5. Parsers. <https://www.postgresql.org/docs/9.6/static/textsearch-parsers.html>. Abgerufen am 13.05.2017.
- PostgreSQL, 1996-2017: PostgreSQL 9.6.3 Documentation / 12.8. Testing and Debugging Text Search. <https://www.postgresql.org/docs/9.6/static/textsearch-debugging.html>. Abgerufen am 13.05.2017



- PostgreSQL 1996-2017: PostgreSQL 9.6.3 Documentation / 28.2 The Statistics Collector. <https://www.postgresql.org/docs/9.6/static/monitoring-stats.html>. Abgerufen am 13.05.2017
- PostgreSQL, 1996-2017: PostgreSQL 9.6.3 Documentation / 63.1 Introduction. <https://www.postgresql.org/docs/9.6/static/gin-intro.html>. Abgerufen am 15.05.2017
- PostgreSQL, 1996-2017: PostgreSQL 9.6.3 Documentation / F.31. pg\_trgm. <https://www.postgresql.org/docs/9.6/static/pgtrgm.html>. Abgerufen am 13.05.2017
- PostgreSQL Usergroup Germany, Warum PostgreSQL benutzen?, <http://www.pgug.de/de/informationen/warum-postgresql-benutzen.html>, Abgerufen am 05.05.2017
- Scopevisio AG: Ausziffern-Ausgleich offener Posten, AusziffernScopevisio.pdf. Abgerufen am 05.05.2017
- Scopevisio AG: Unternehmenssoftware aus der Cloud - Unternehmensprofil Scopevisio AG, <https://www.scopevisio.com/downloads/unternehmen/Unternehmensprofil.pdf>. Abgerufen am 05.05.2017
- Sedgewick, Robert und Wayne, Kevin: Algorithmen - Algorithmen und Datenstrukturen, 4. Auflage, Hallbergmoos (Pearson Verlag) 2014.
- Sedgewick, Robert: Algorithmen, 2. Auflage, München (Addison Wesley) 1992
- Sigaev, Teodor und Bartunov Oleg, 2001-2007: Full-Text Search in PostgreSQL, <https://www.pgcon.org/2007/schedule/attachments/12-fts.pdf>. Abgerufen am 13.05.2017
- Sigaev Teodor und Bartunoc Oleg, 2006: GIN, <http://www.sigaev.ru/gin/Gin.pdf>. Abgerufen am 15.05.2017
- Smith, Gregory: PostgreSQL 9.0 High Performance, 1. Auflage, Birmingham (Packt Publishing Ltd.) 2010.
- UC BERKELEY DATABASE GROUP, 1995: Generalized Search Trees for Database Systems, <http://db.cs.berkeley.edu/papers/vldb95-gist.pdf>. Abgerufen am 04.06.2017

Vossen, Gottfried: Datenmodelle, Datenbanksprachen und  
Datenbankmanagementsysteme, 5. Auflage, München 2008.

Winand, Markus: SQL Performance Explained, 1. Auflage, Wien (Winand (Verleger))  
2012.

w3schools, 1999-2017: SQL LIKE Operator,  
[https://www.w3schools.com/sql/sql\\_like.asp](https://www.w3schools.com/sql/sql_like.asp). Abgerufen am 21.06.2017

w3schools, 1999-2017: SQL Views, [https://www.w3schools.com/sql/sql\\_view.asp](https://www.w3schools.com/sql/sql_view.asp).  
Abgerufen am 04.06.2017

## 7 Anhang

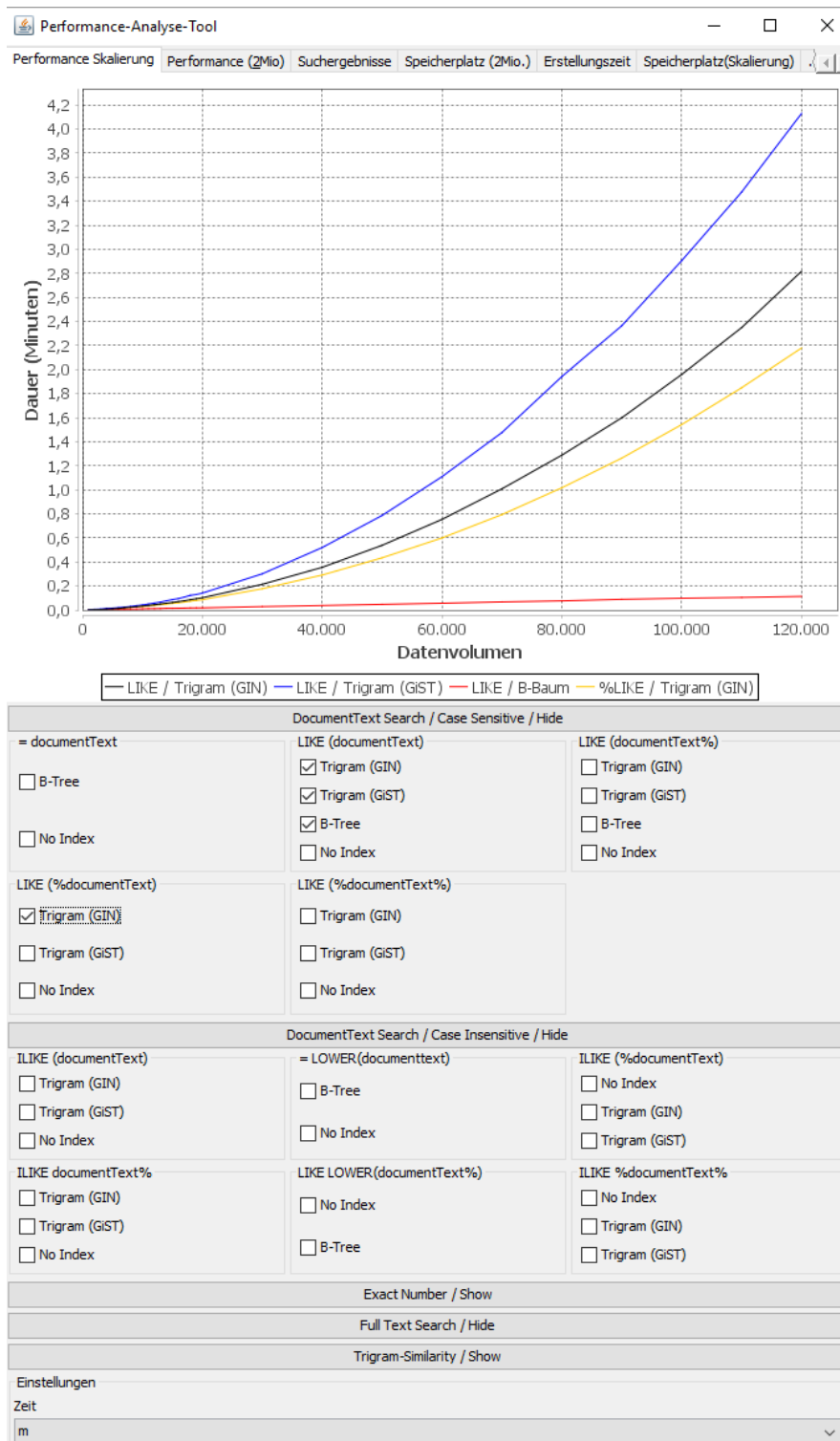


Abbildung 20

### Abfragen auf exakte Übereinstimmung des Verwendungszwecks

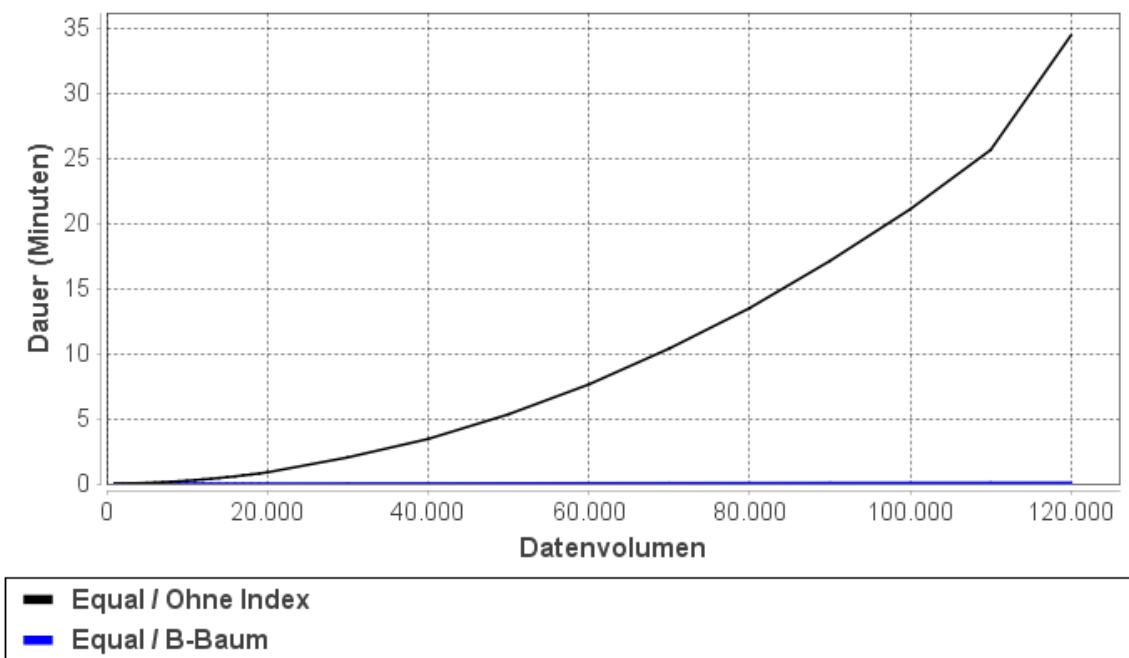


Abbildung 21

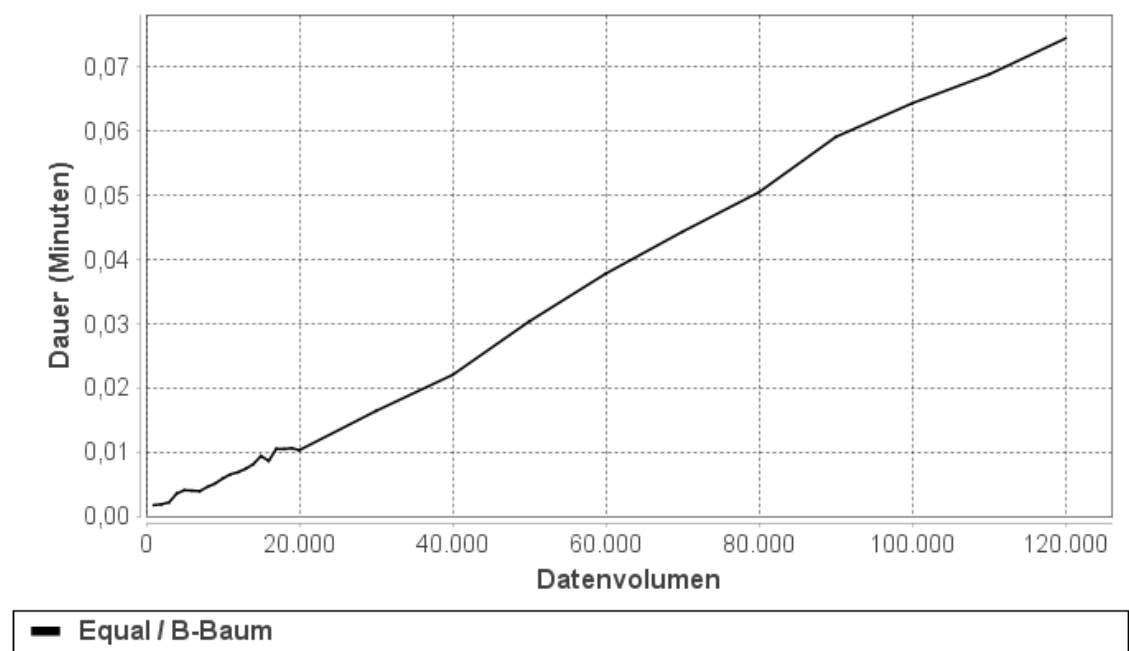


Abbildung 22

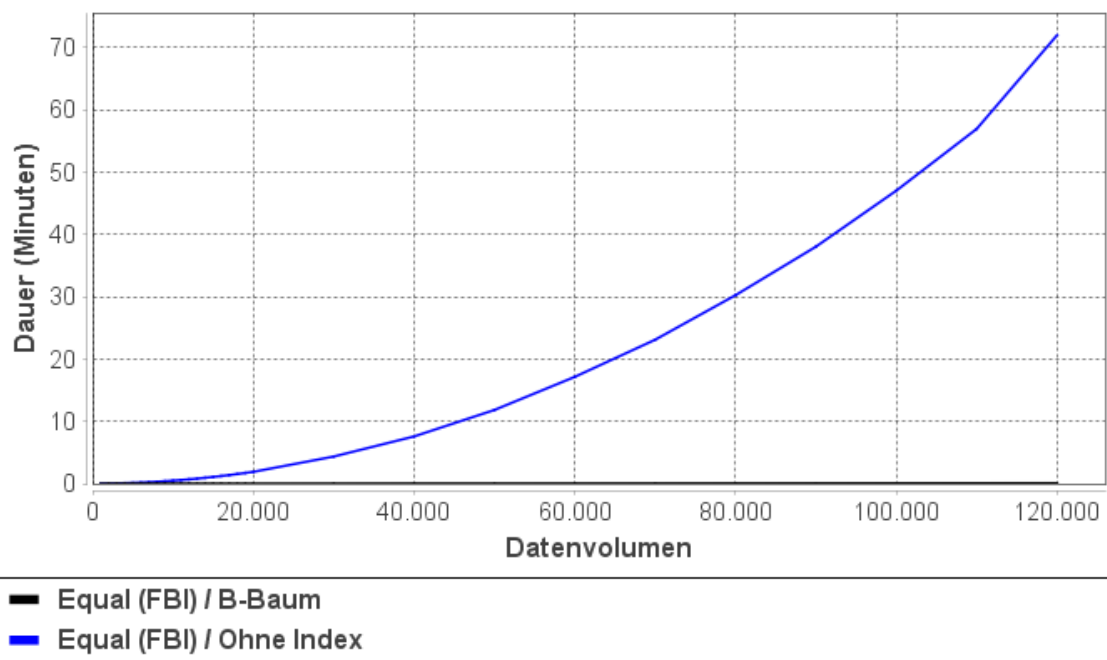


Abbildung 23

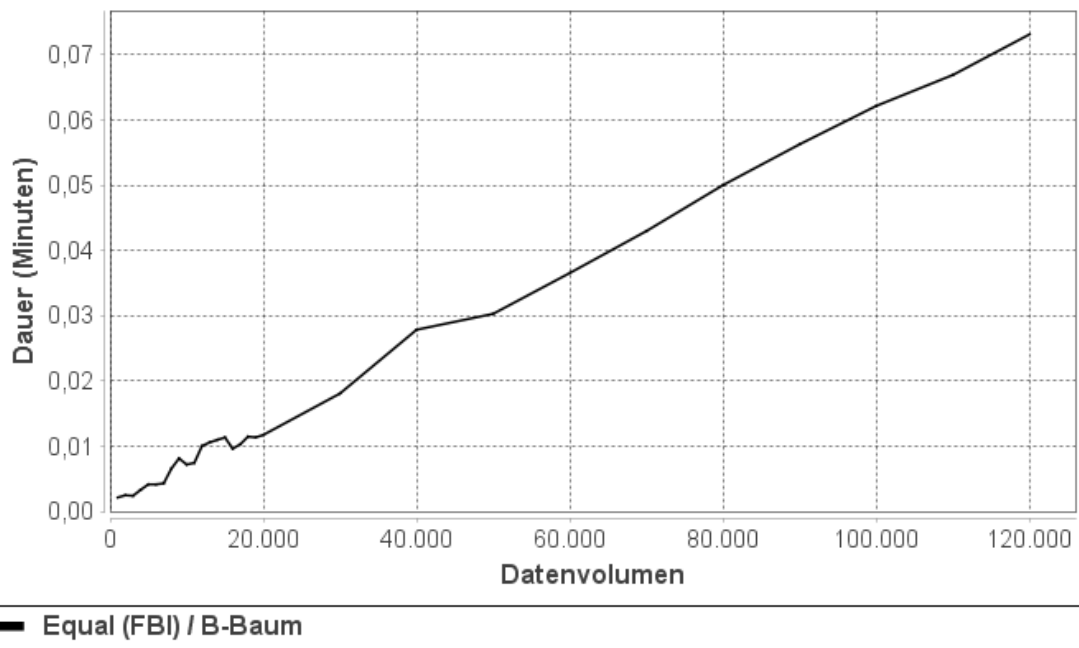


Abbildung 24

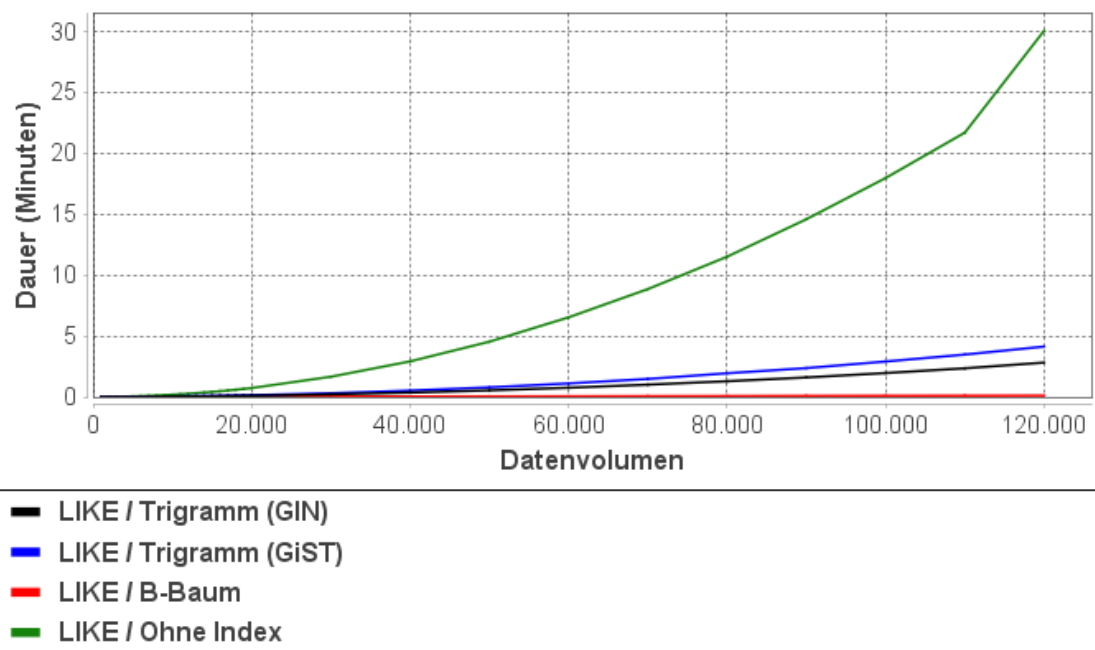


Abbildung 25

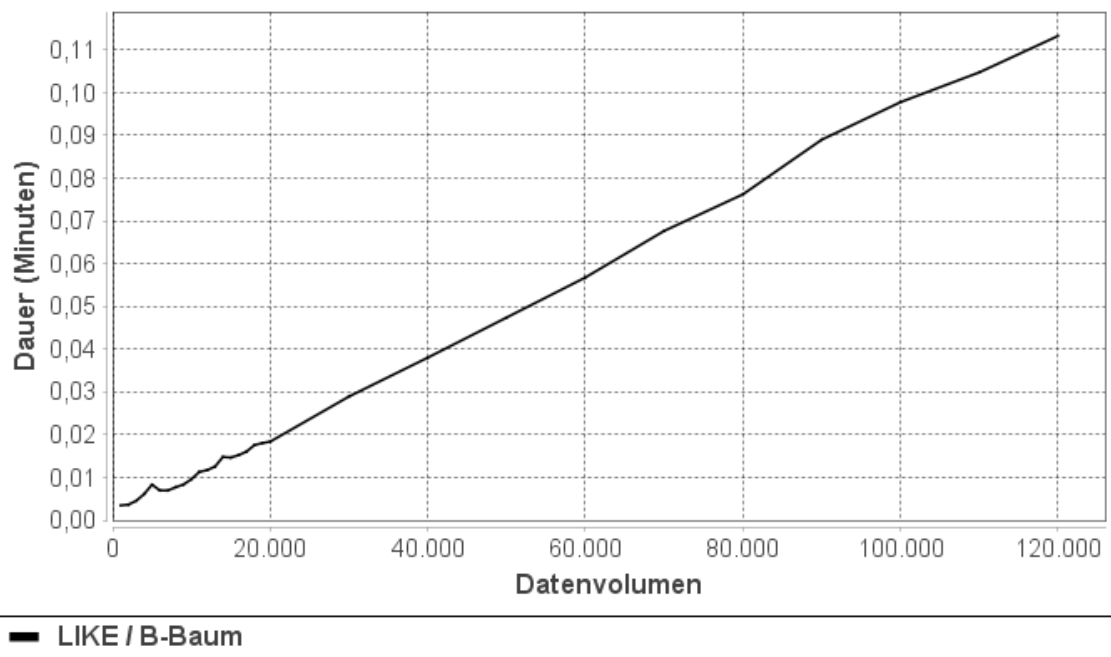


Abbildung 26

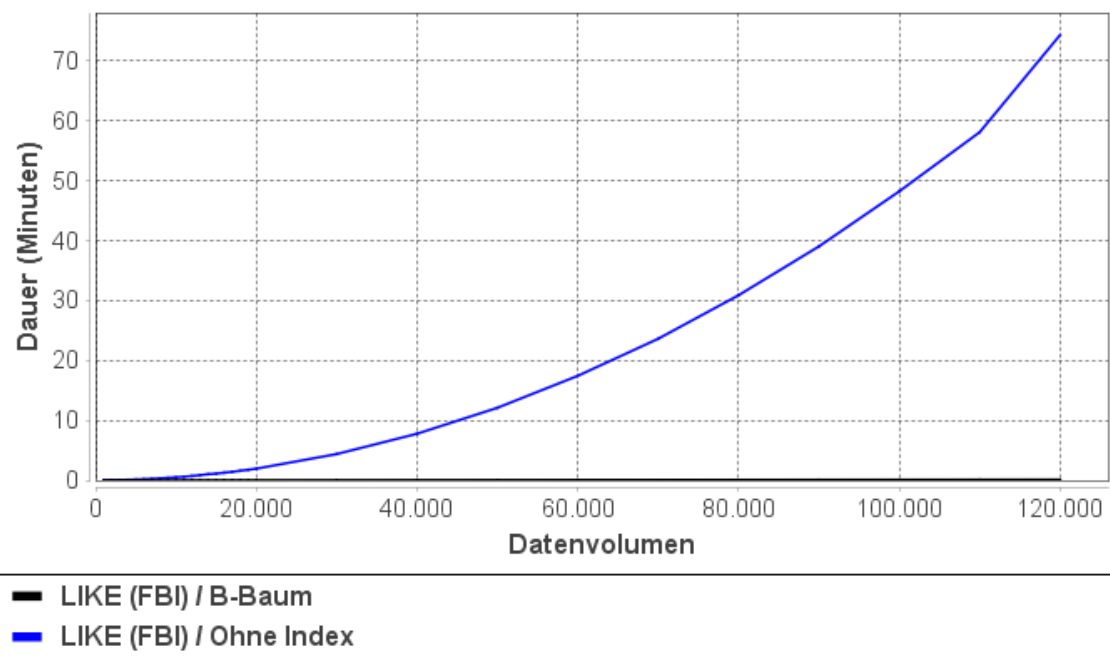


Abbildung 27

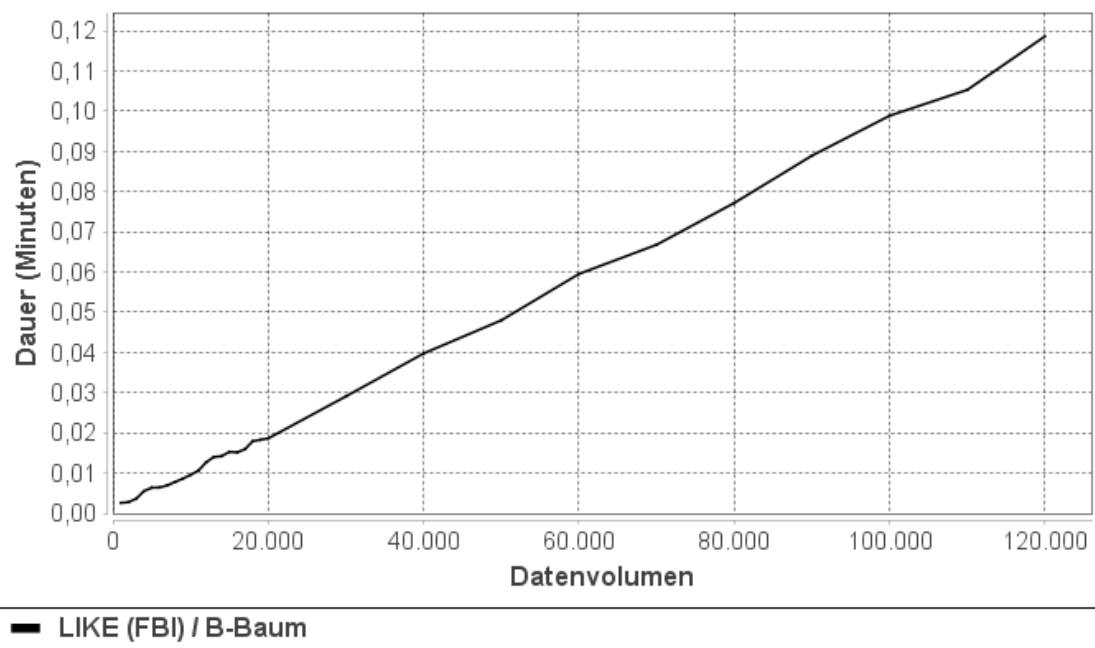


Abbildung 28

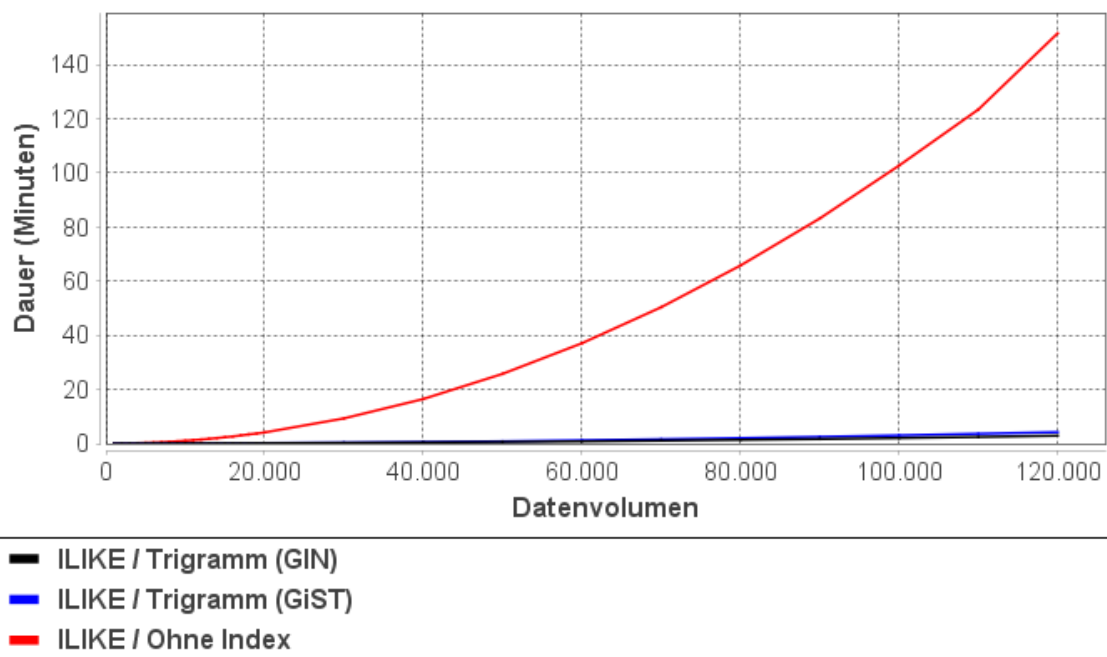


Abbildung 29

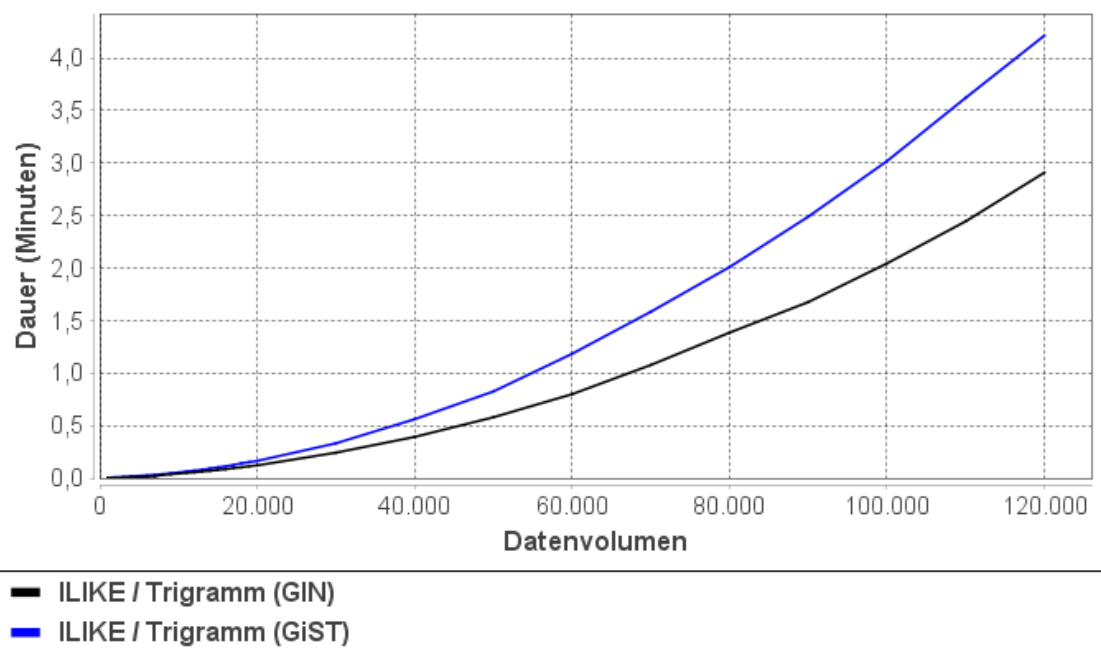


Abbildung 30



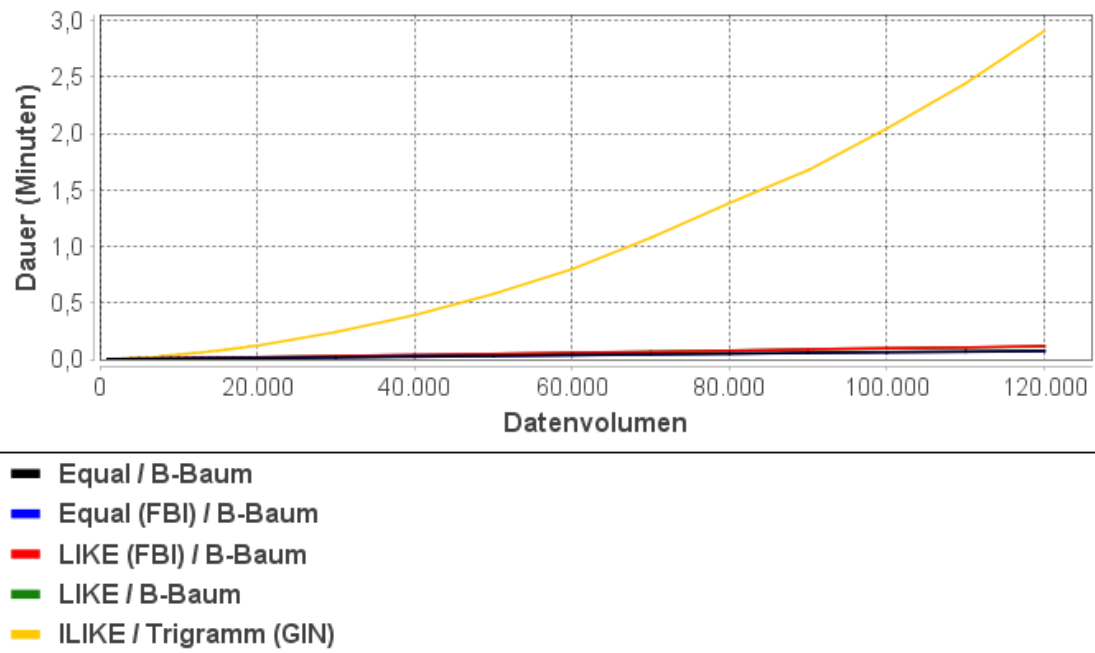


Abbildung 31

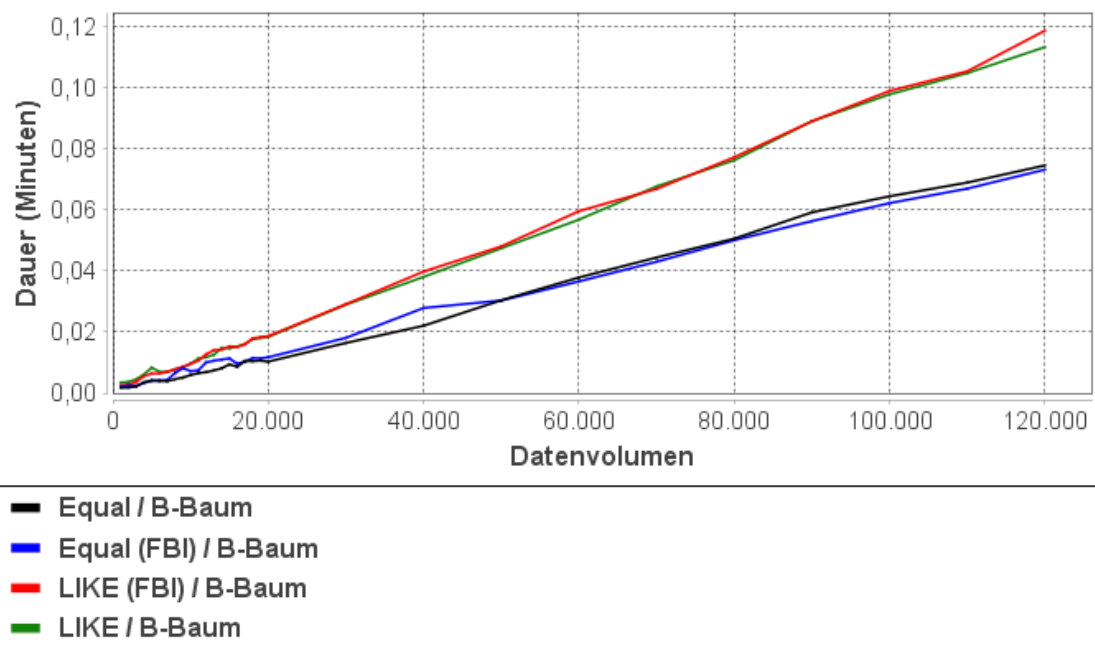


Abbildung 32

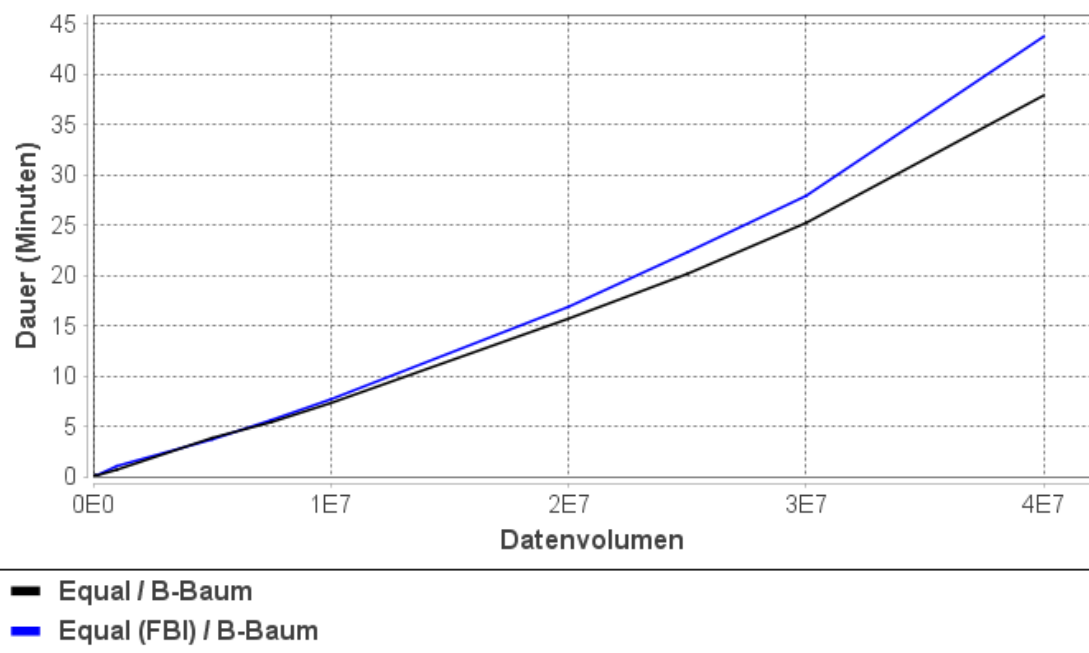


Abbildung 33

#### Abfragen unter Auslassung von beliebigen f hrenden Zeichenkette

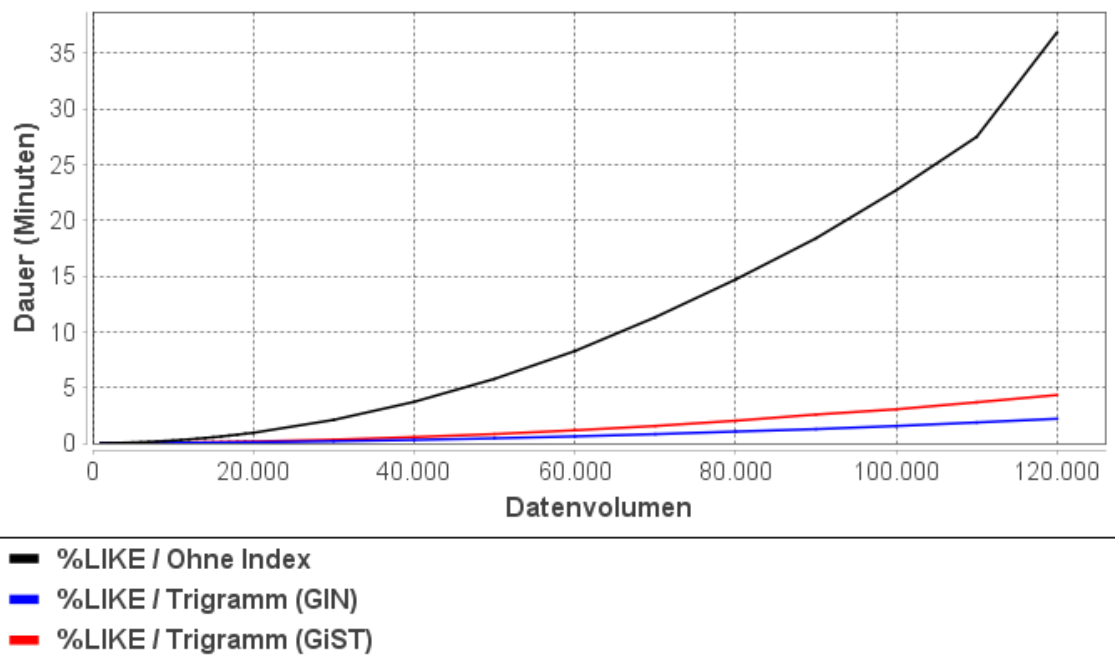


Abbildung 34

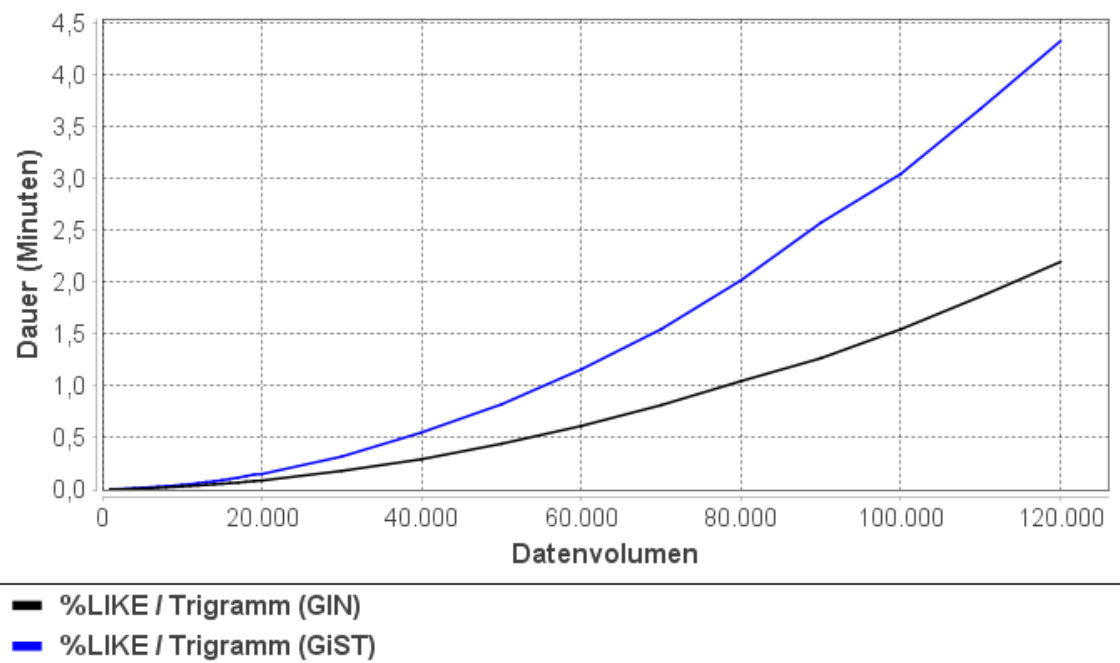


Abbildung 35

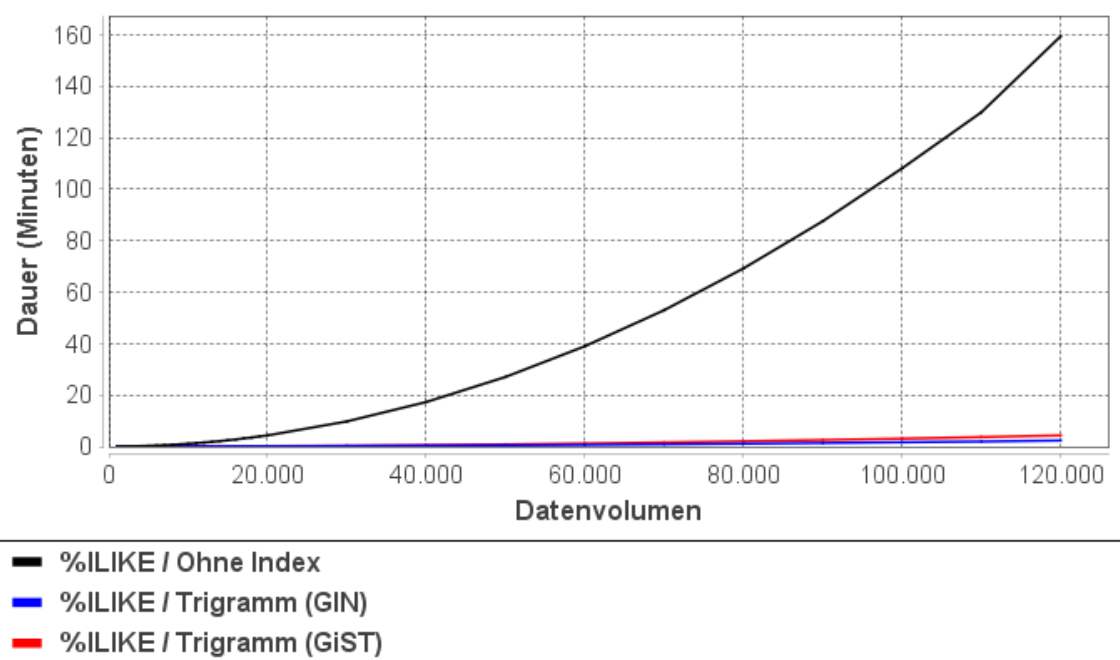


Abbildung 36

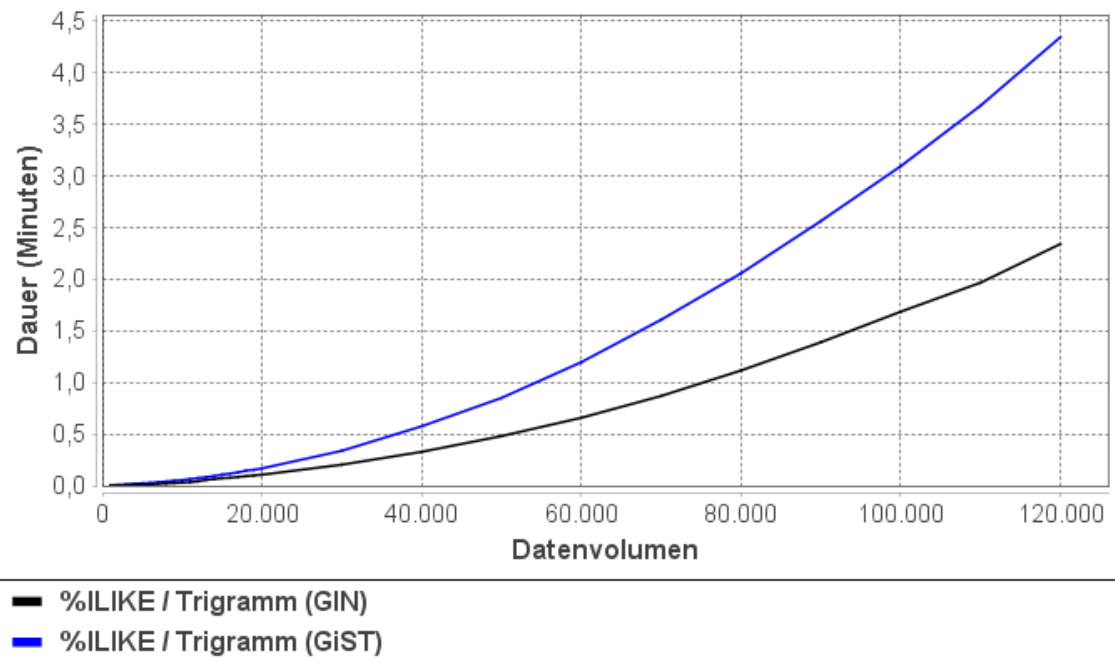


Abbildung 37

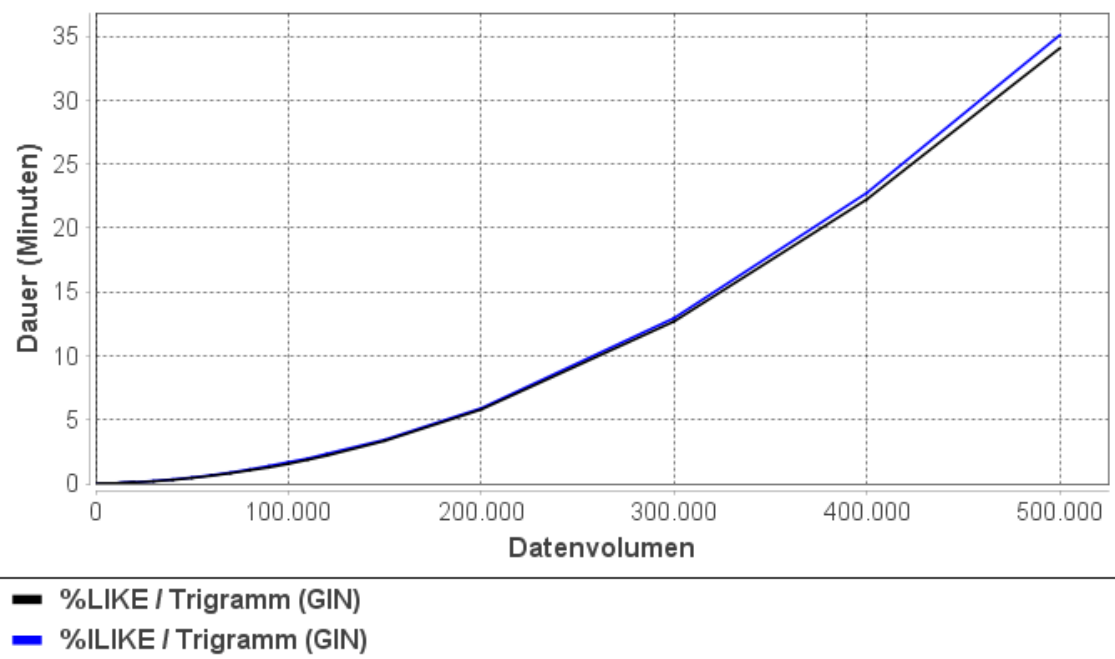


Abbildung 38

### Abfragen unter Auslassung von beliebig endender Zeichen

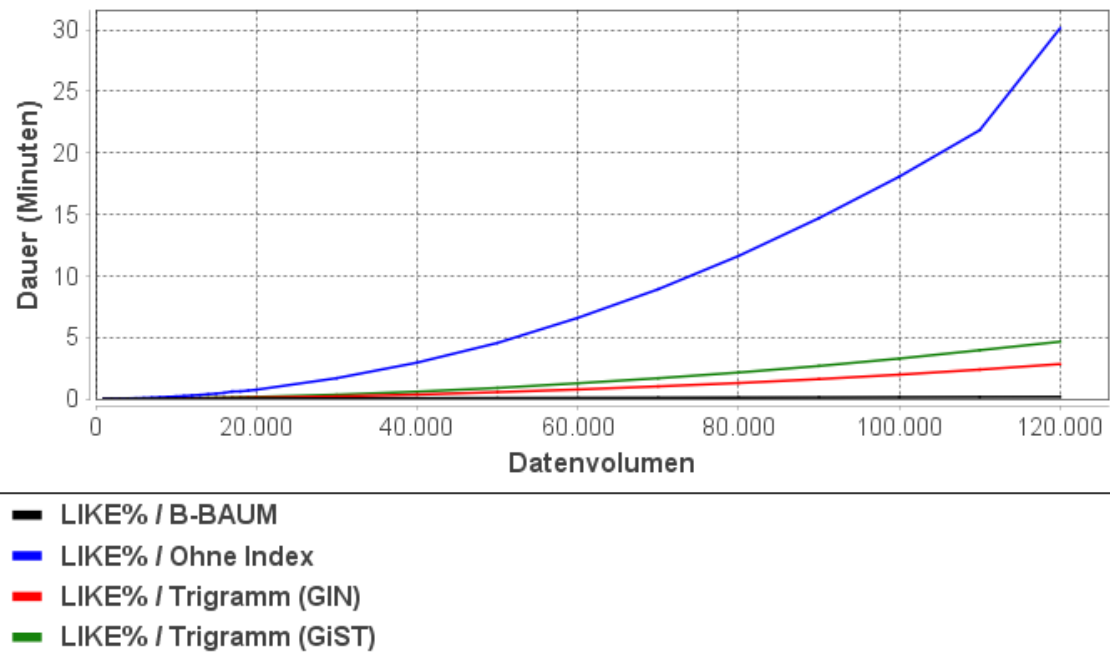


Abbildung 39

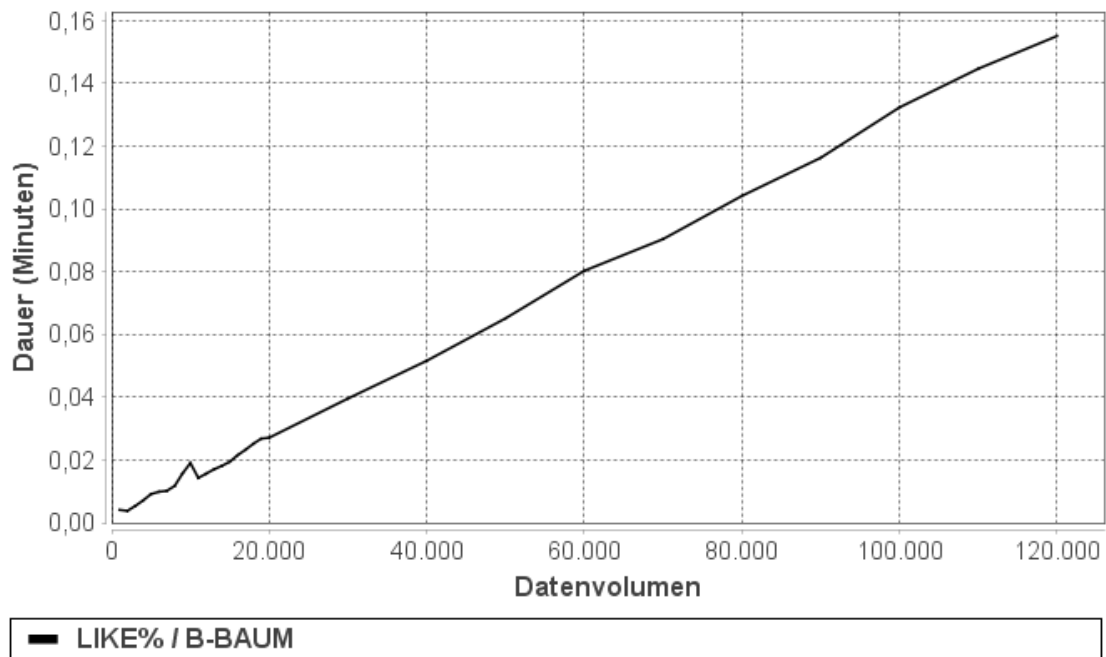


Abbildung 40

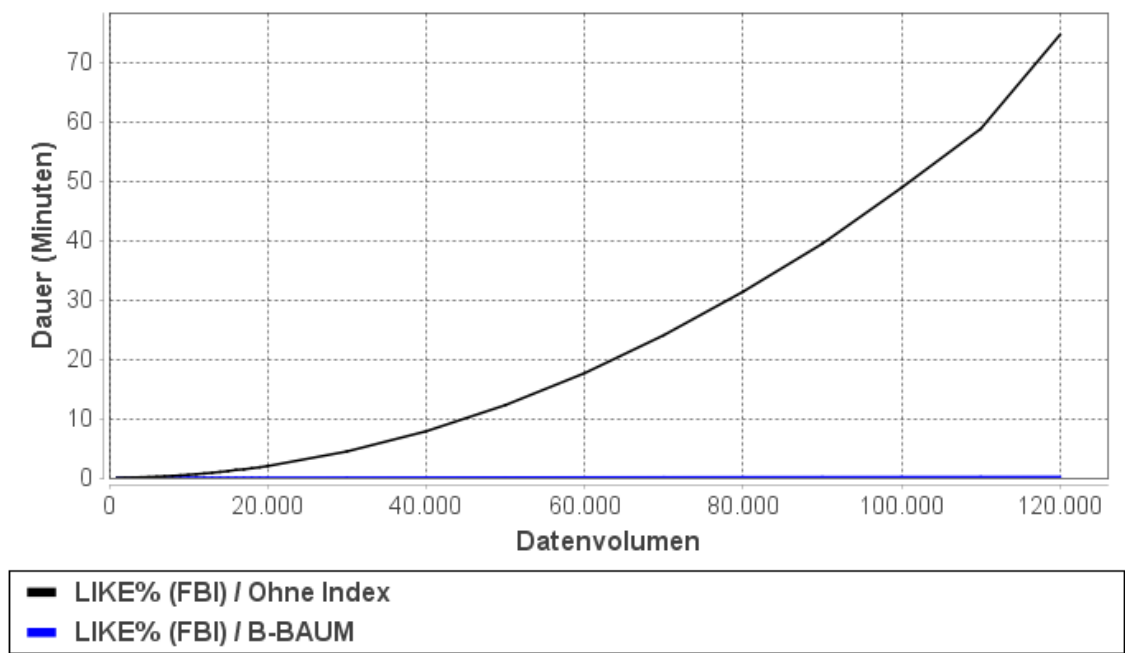


Abbildung 41

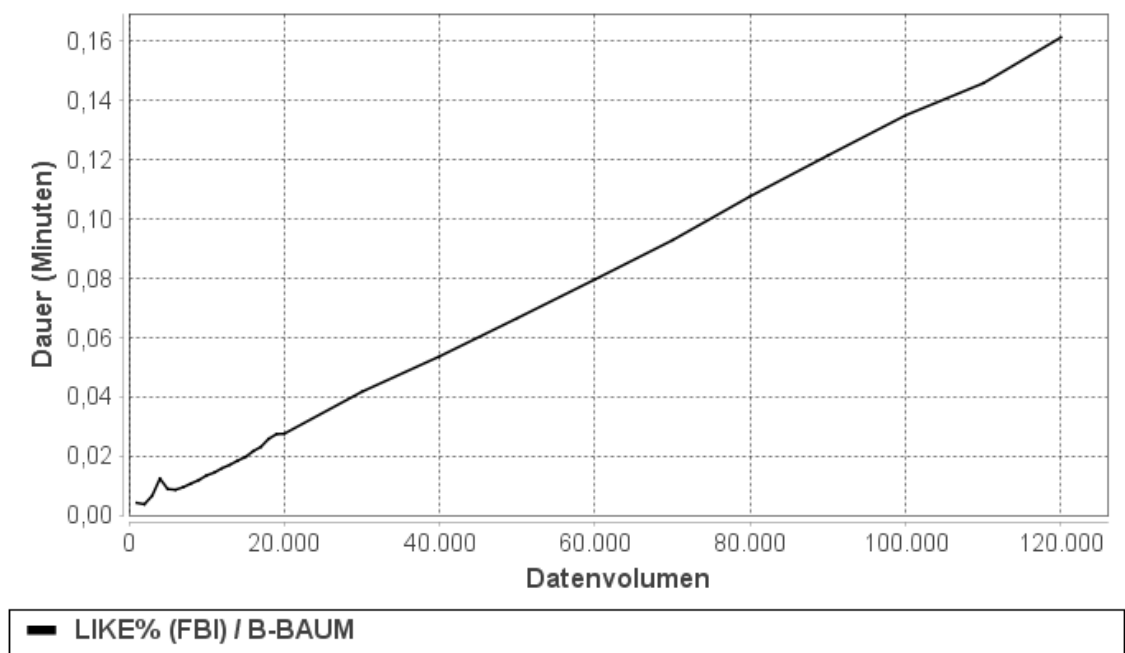


Abbildung 42

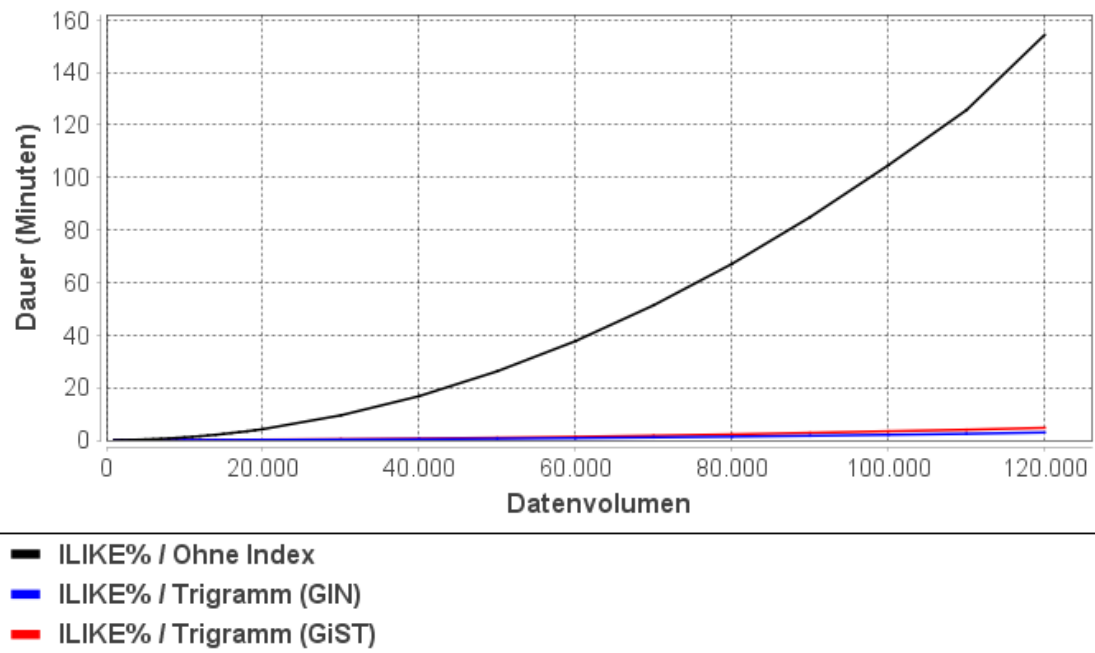


Abbildung 43

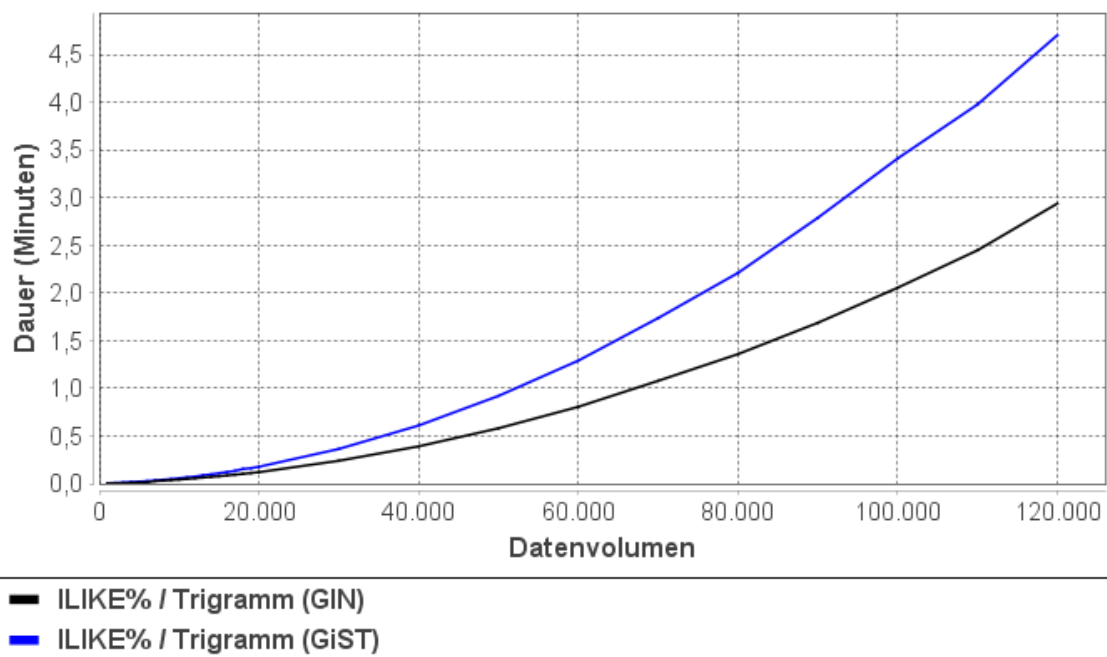


Abbildung 44

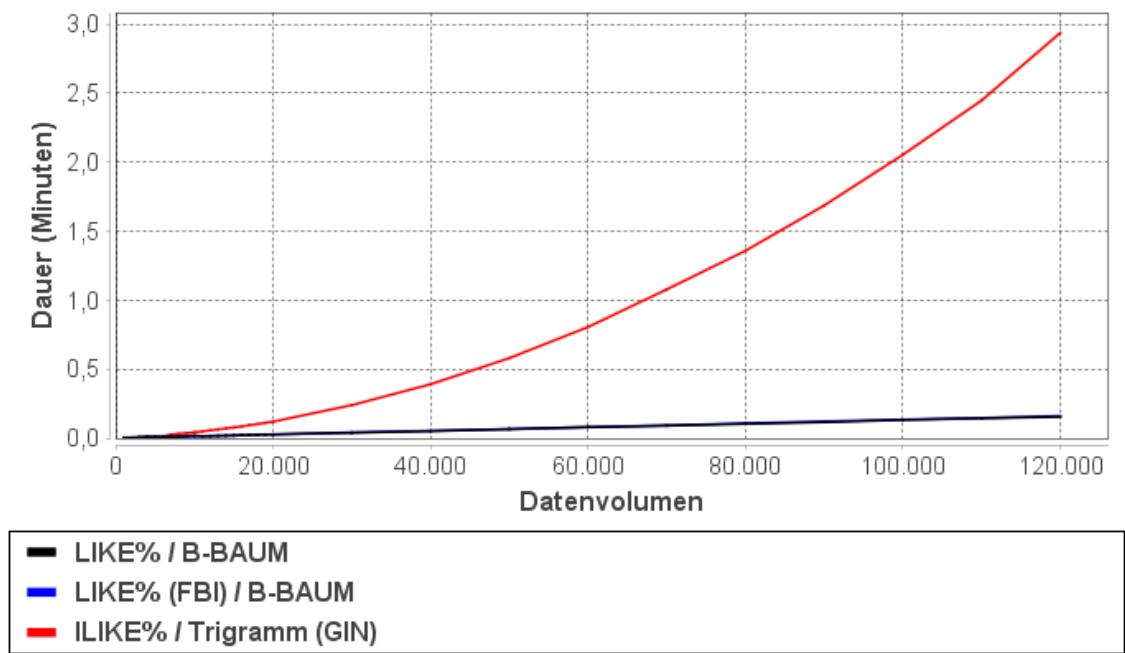


Abbildung 45

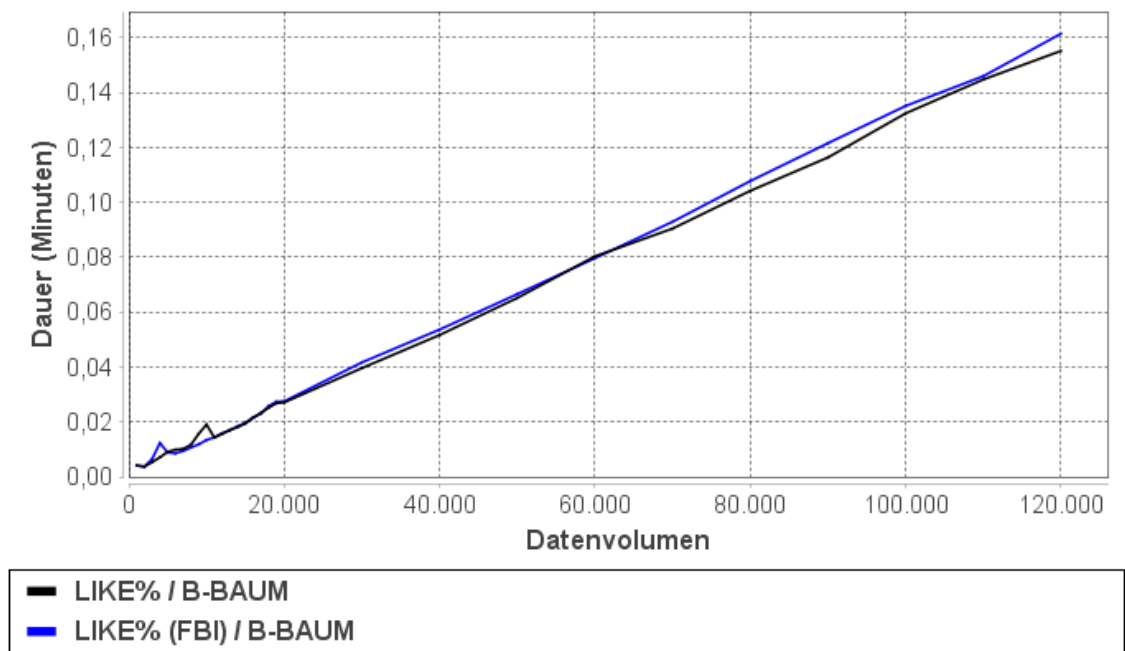


Abbildung 46



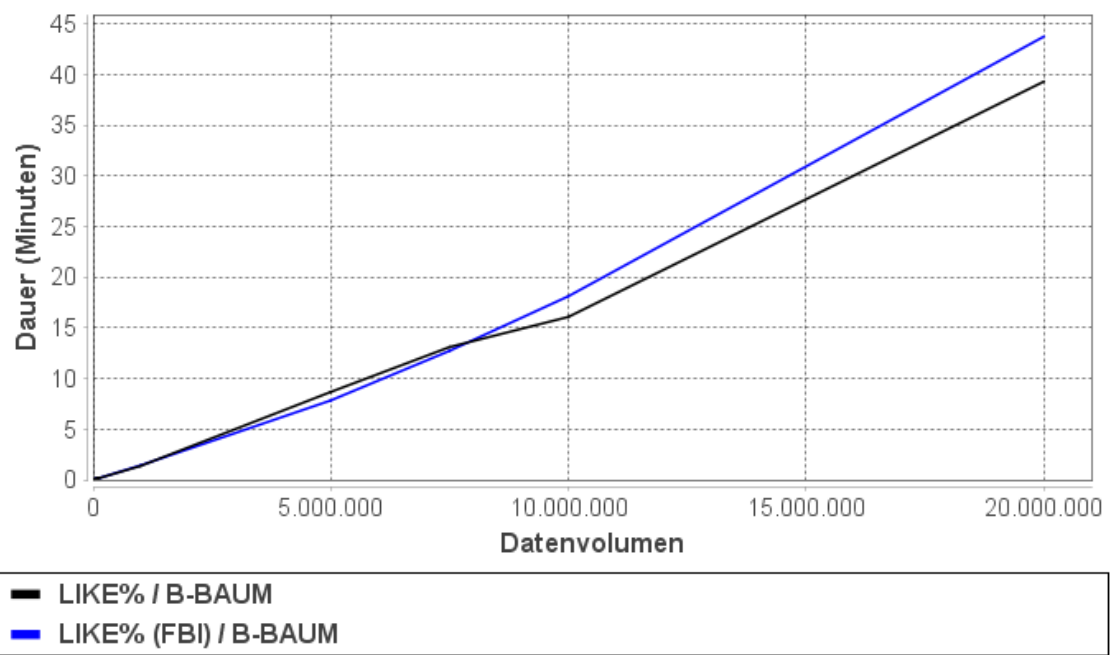


Abbildung 47

## Abfragen unter Auslassung von beliebig führenden und endenden Zeichenkette

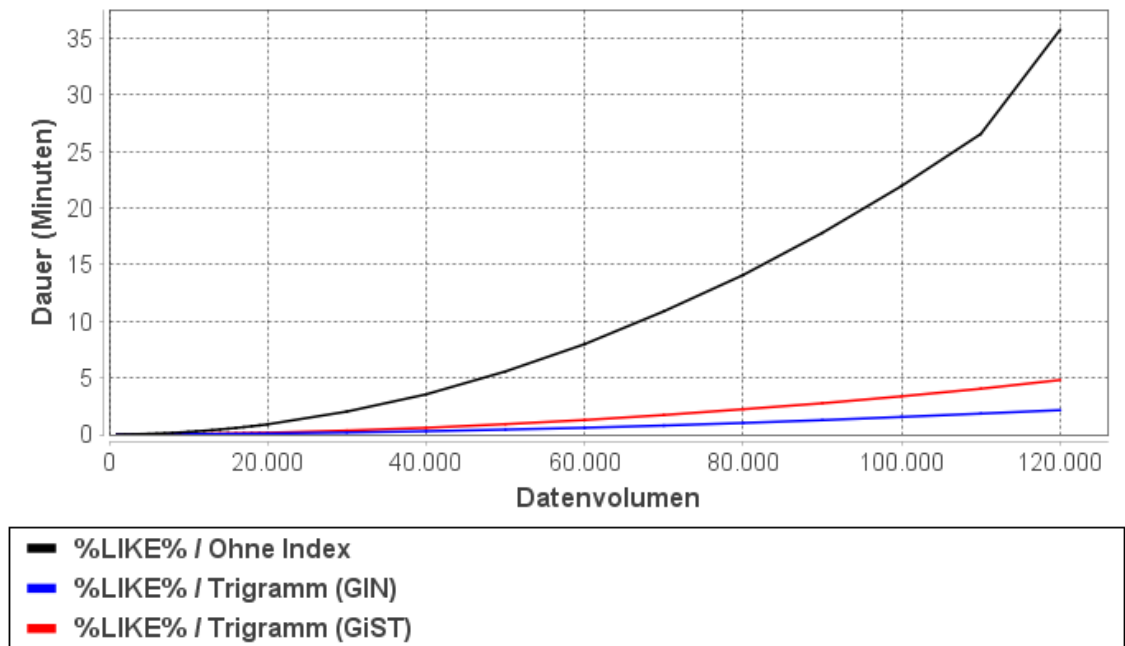


Abbildung 48

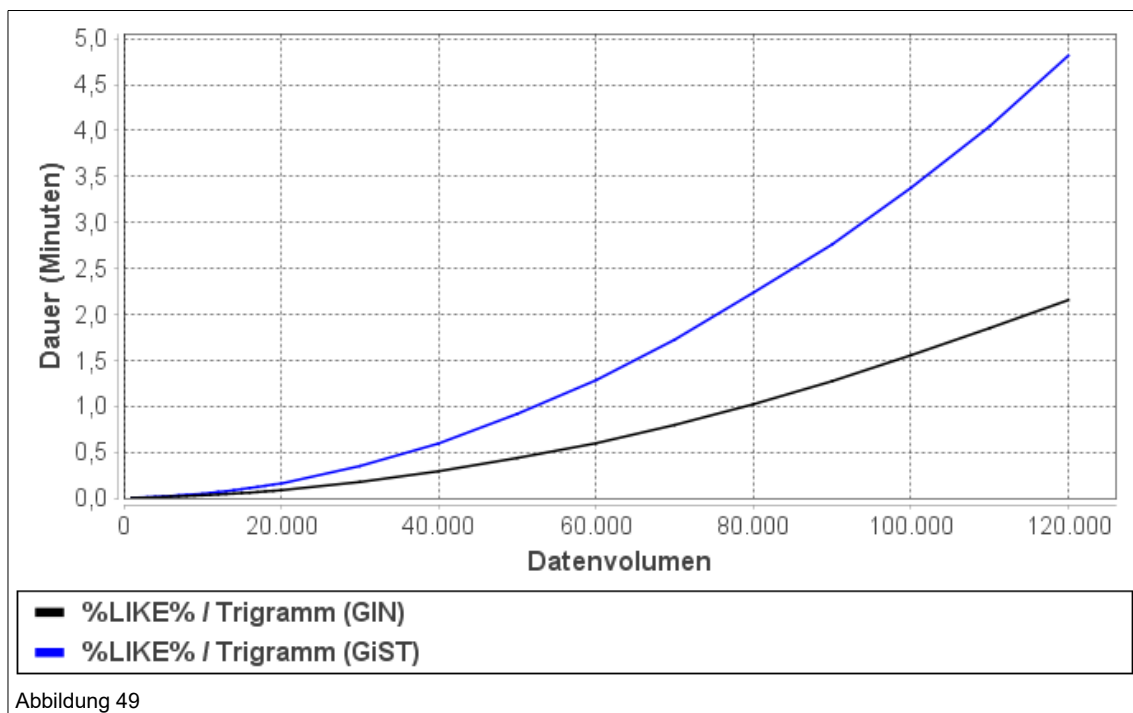


Abbildung 49

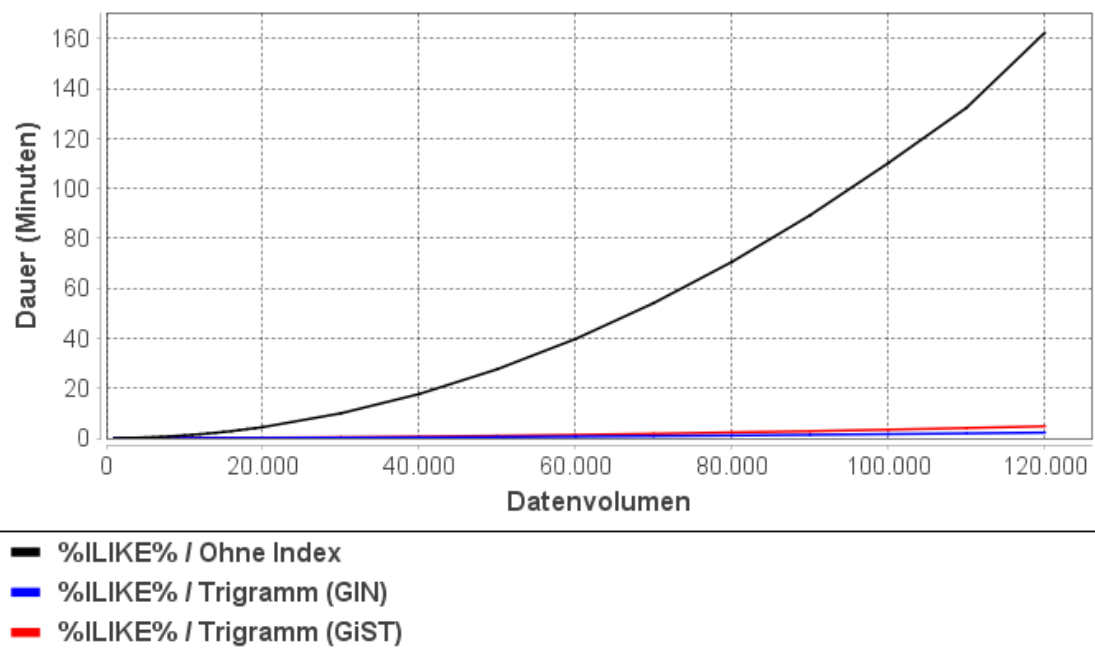


Abbildung 50

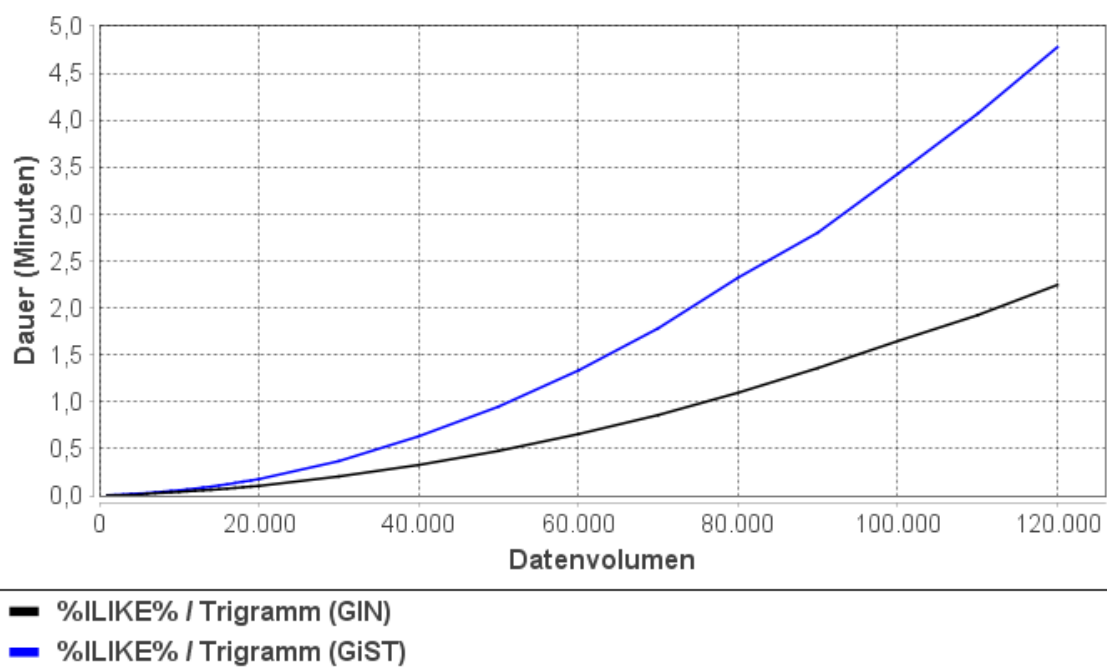


Abbildung 51

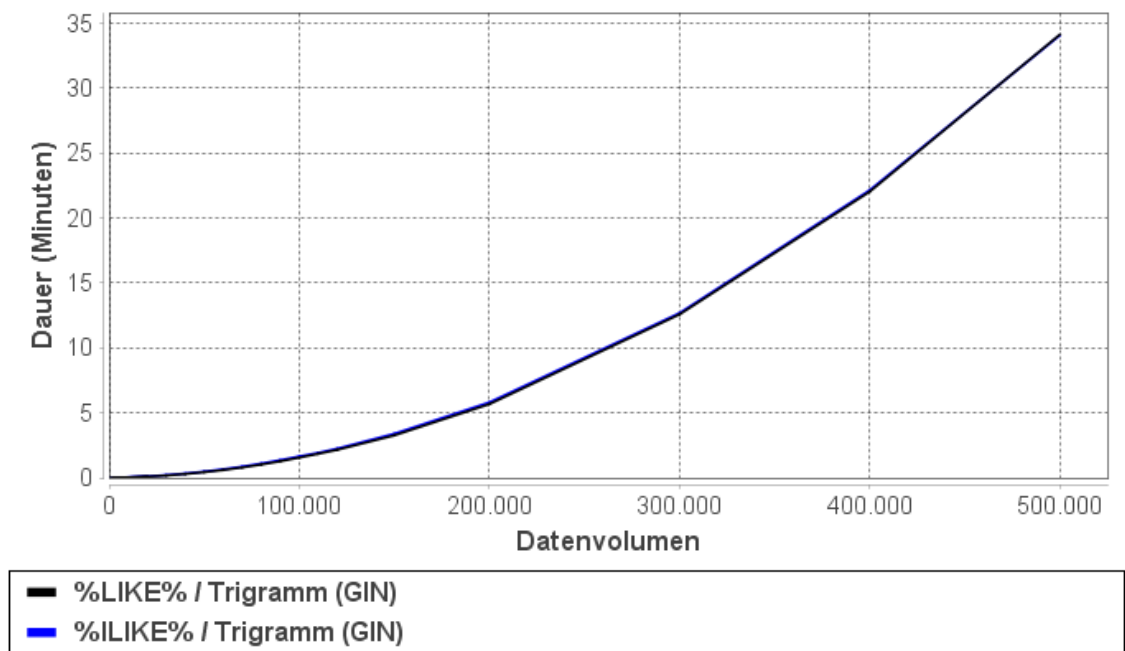


Abbildung 52

### Abfragen mit Teilen des Verwendungszwecks

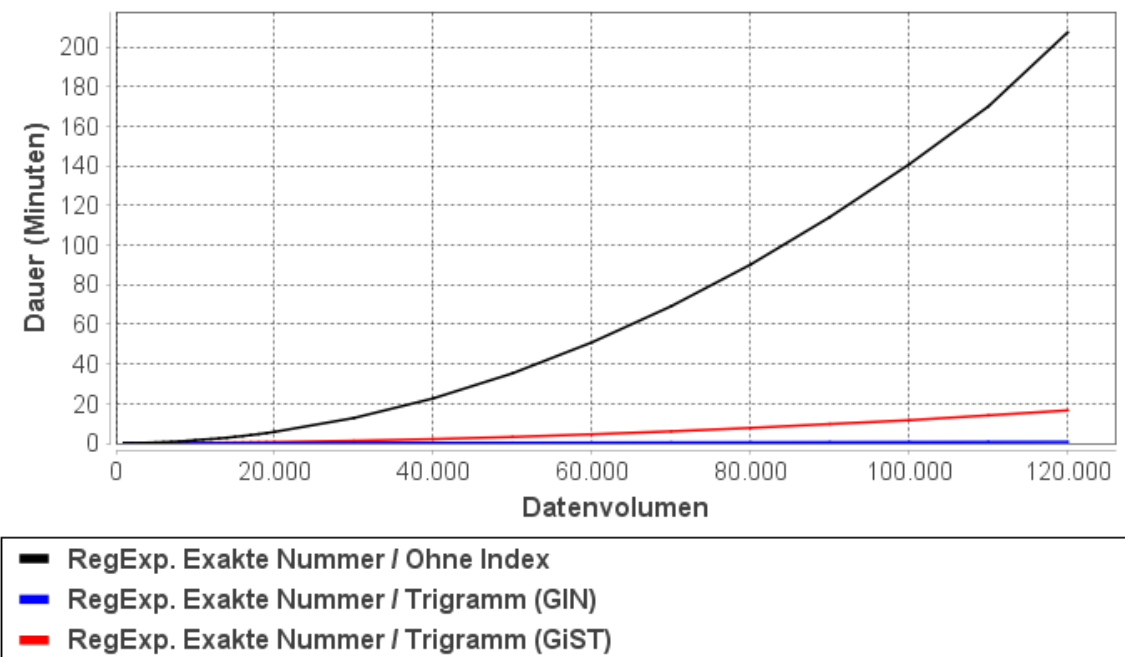


Abbildung 53 (.)

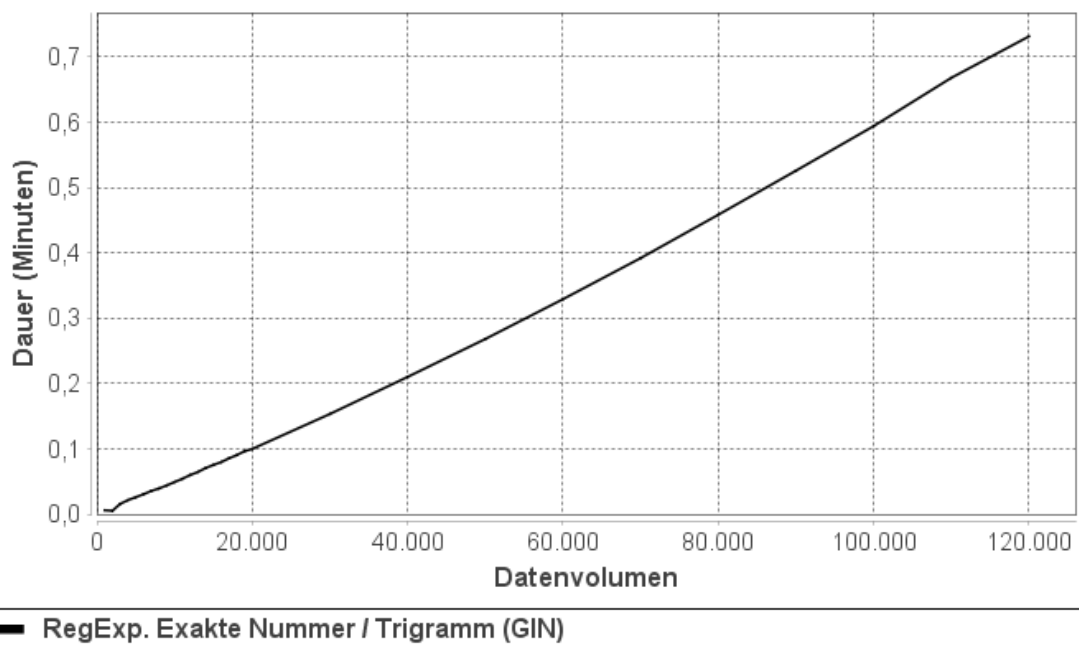


Abbildung 54

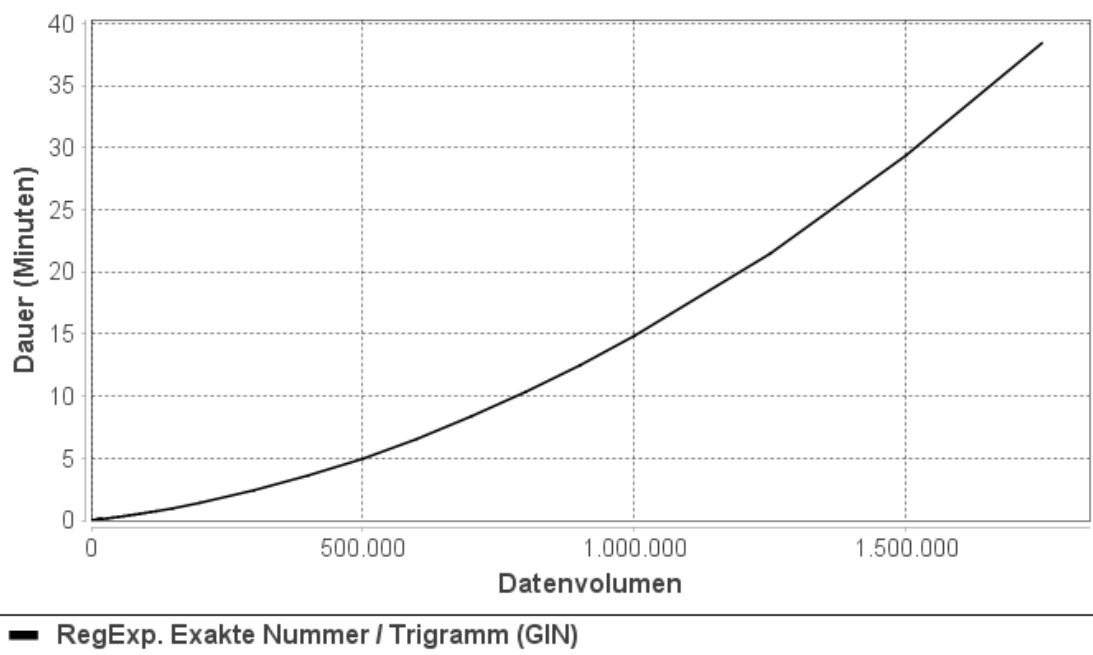


Abbildung 55

## Abfragen auf aufbereiteten Daten des Verwendungszwecks

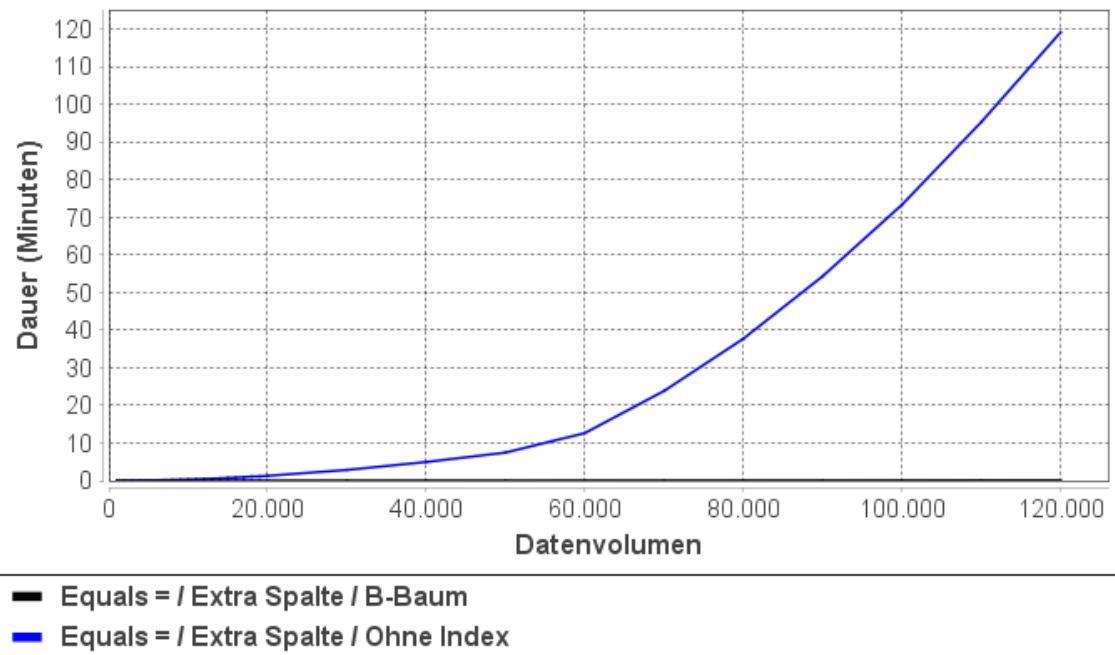


Abbildung 56

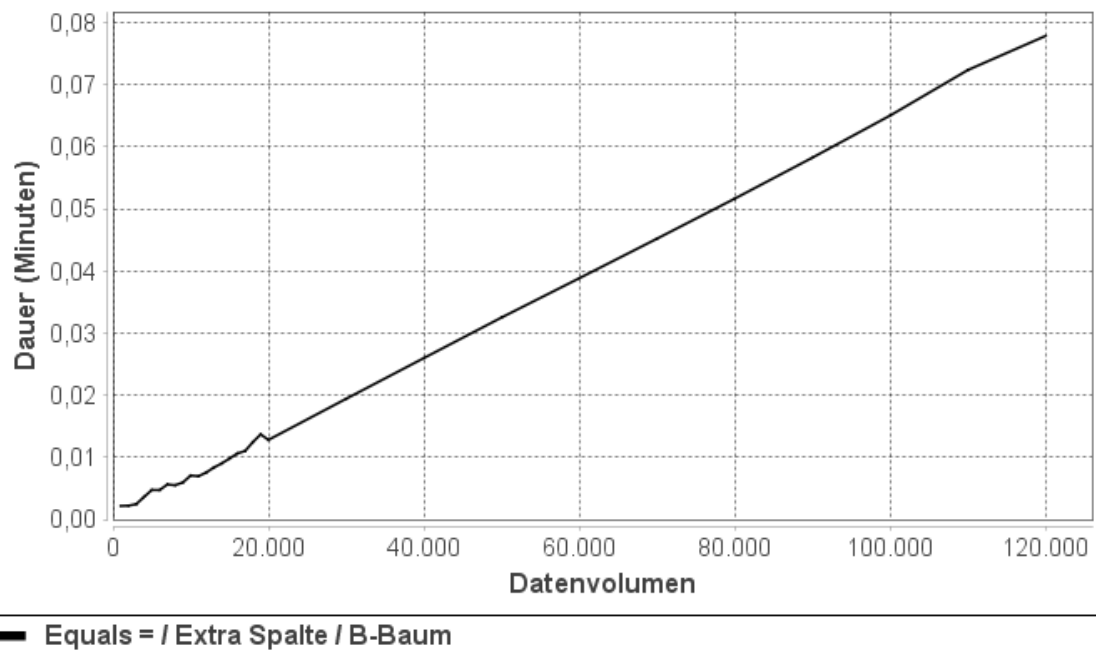
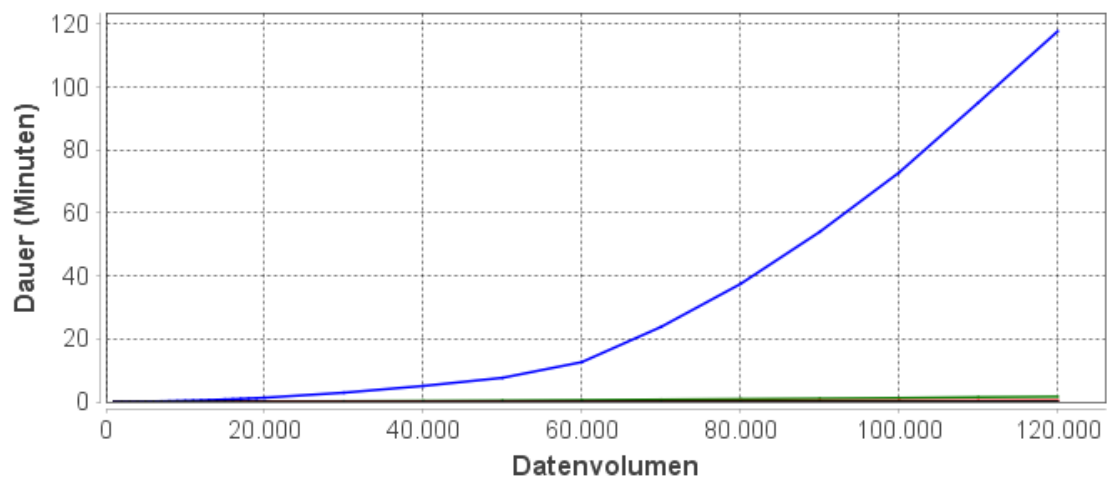
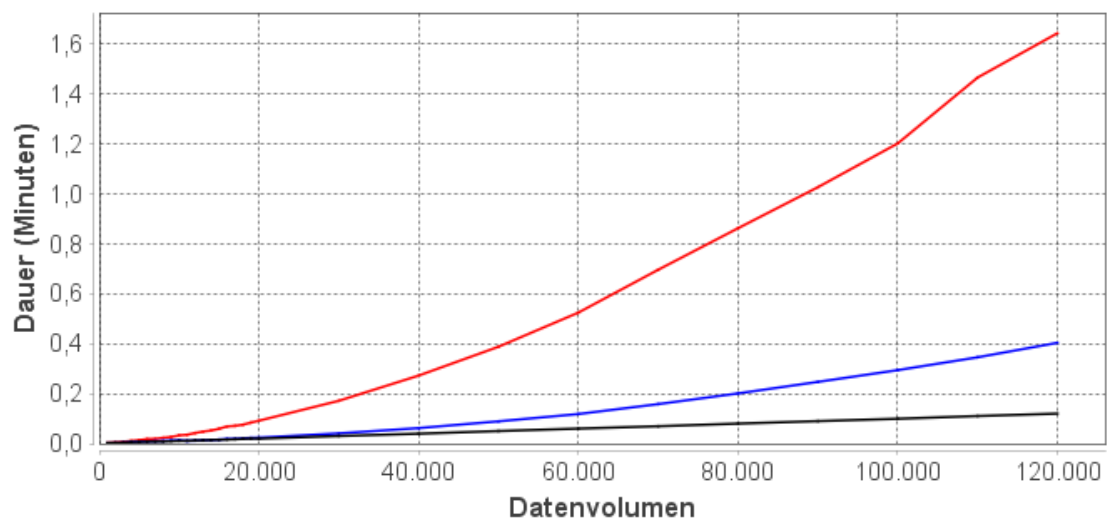


Abbildung 57



- LIKE / Extra Spalte / B-Baum
- LIKE / Extra Spalte / Ohne Index
- LIKE / Extra Spalte / Trigramm (GIN)
- LIKE / Extra Spalte / Trigramm (GiST)

Abbildung 58



- LIKE / Extra Spalte / B-Baum
- LIKE / Extra Spalte / Trigramm (GIN)
- LIKE / Extra Spalte / Trigramm (GiST)

Abbildung 59

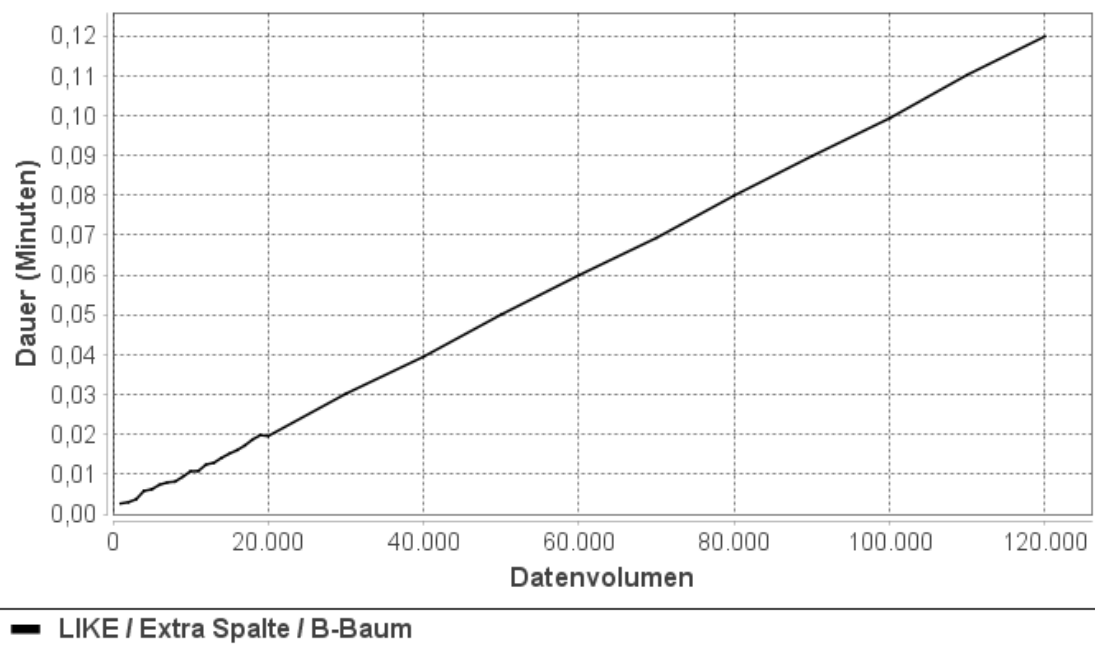


Abbildung 60

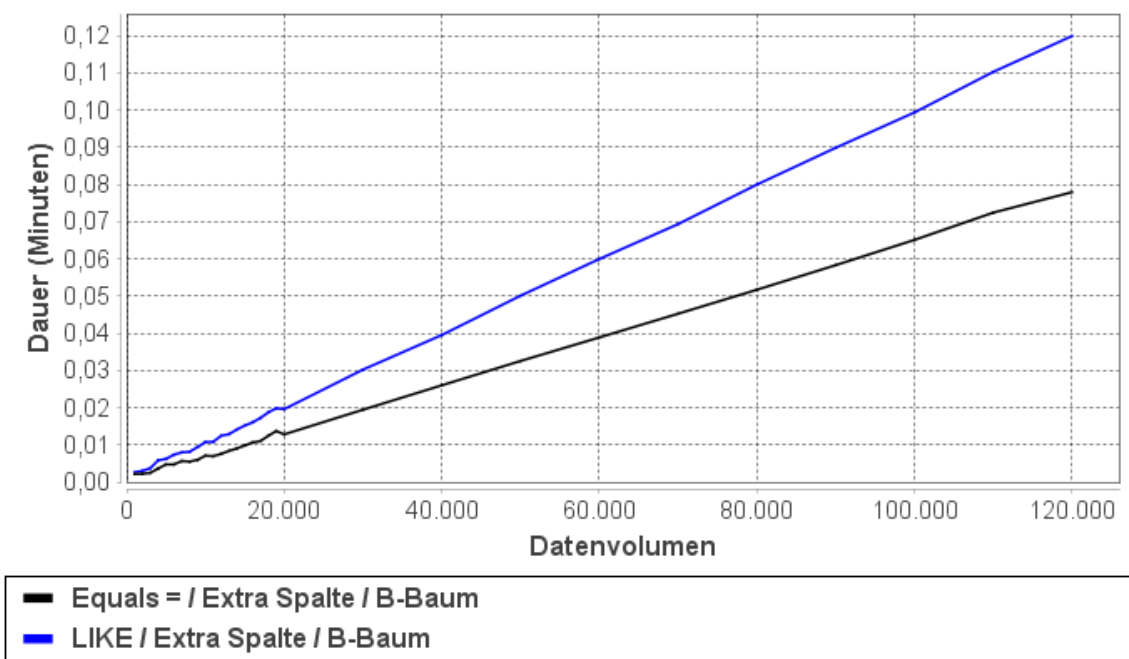


Abbildung 61



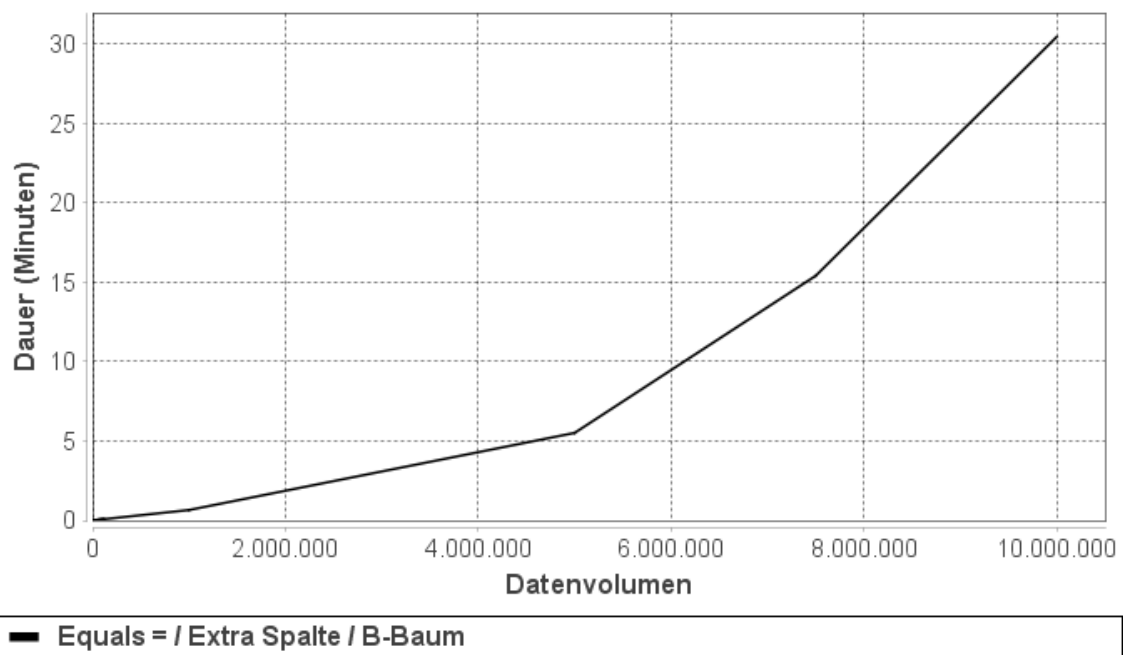


Abbildung 62

## Volltextsuche

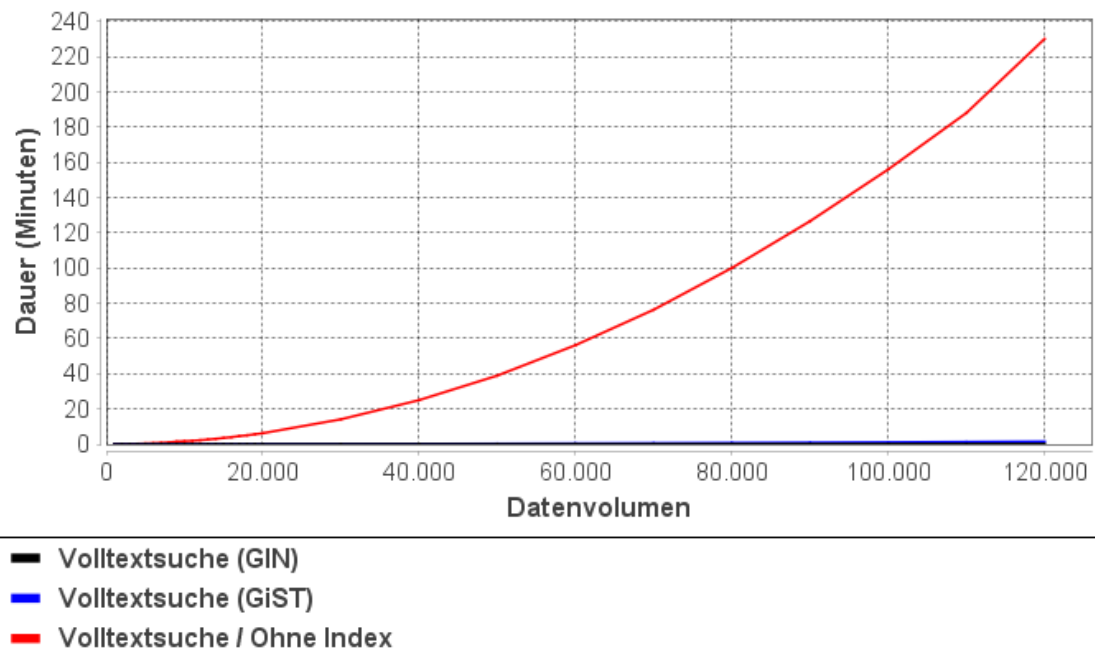


Abbildung 63

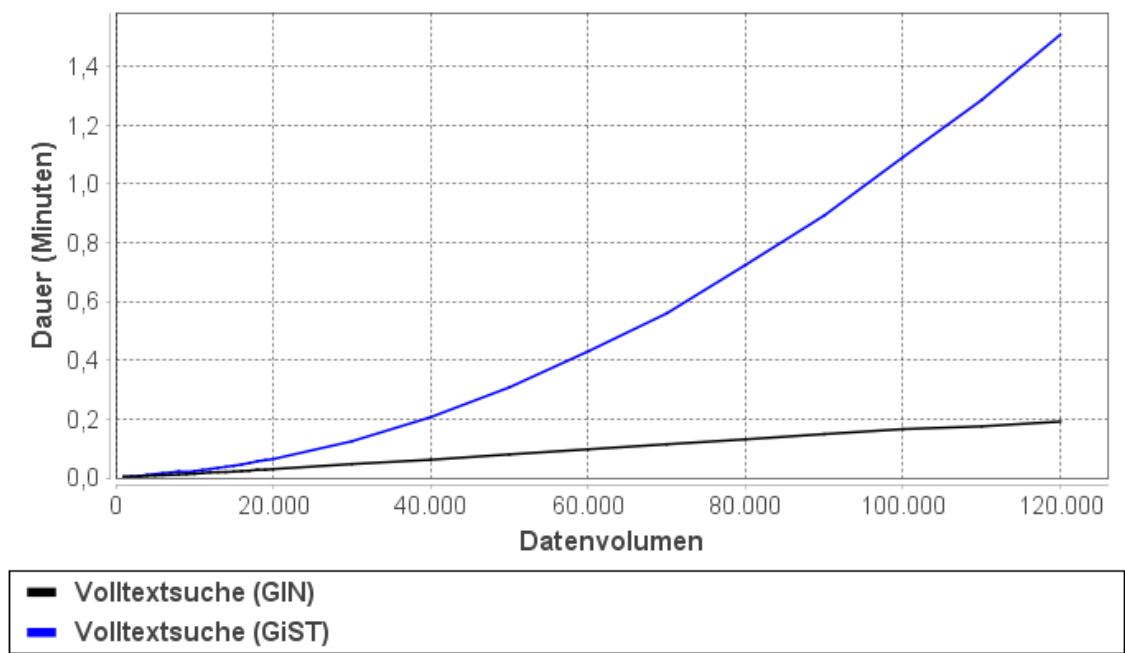


Abbildung 64

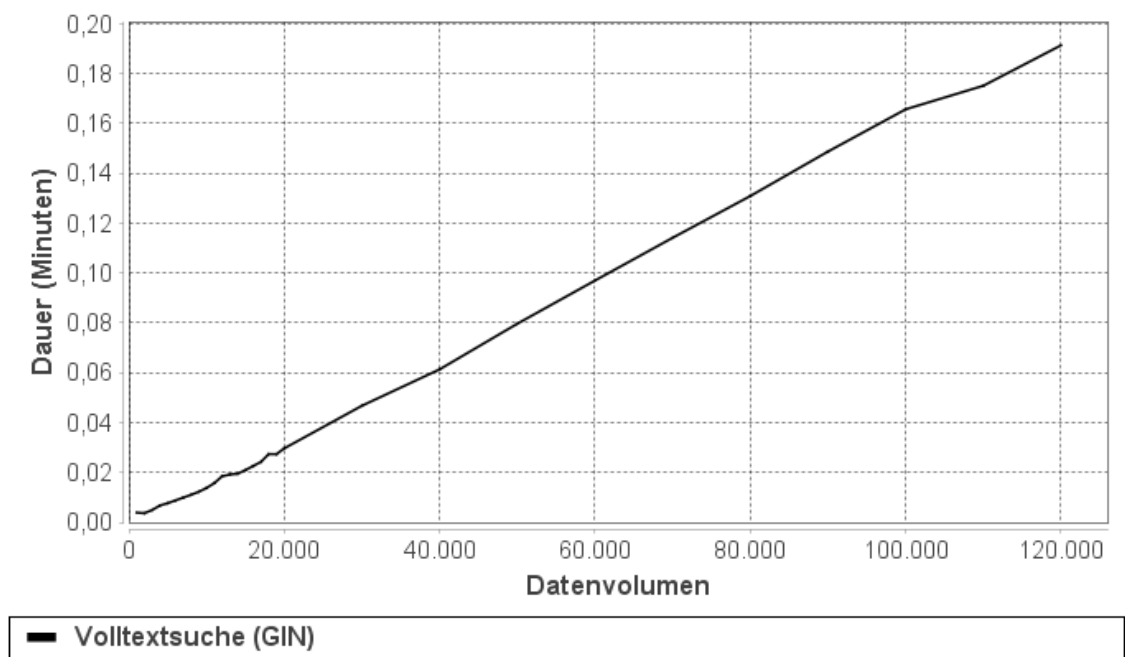


Abbildung 65

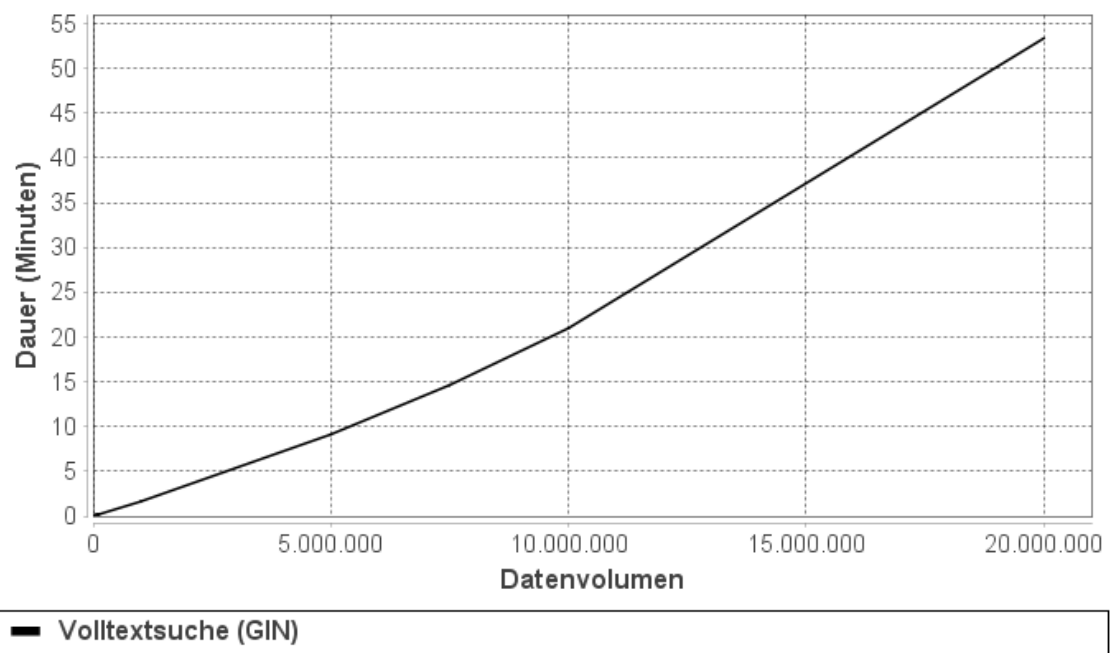


Abbildung 66

### Ähnlichkeitsüberprüfung

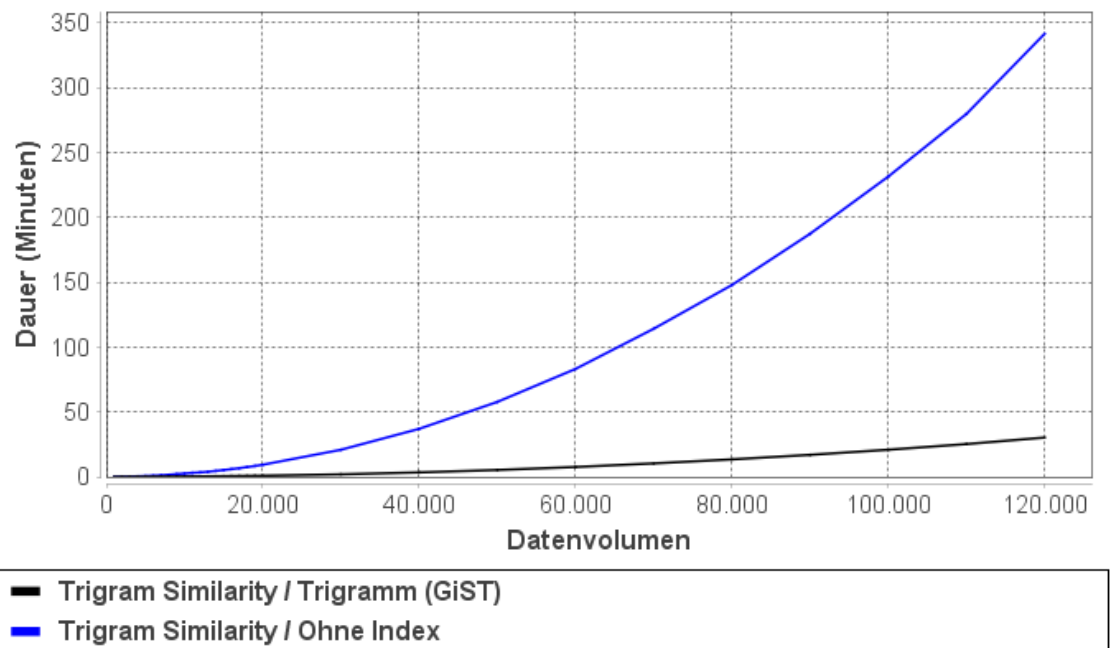


Abbildung 75

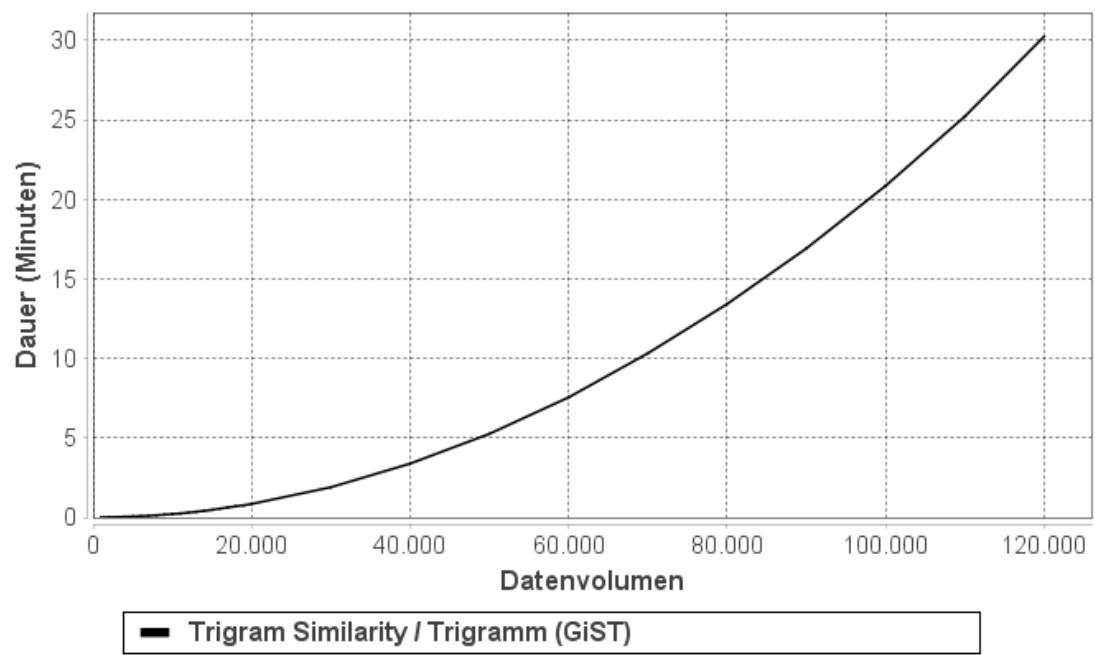


Abbildung 76

## Größenentwicklung bei skalierenden Daten

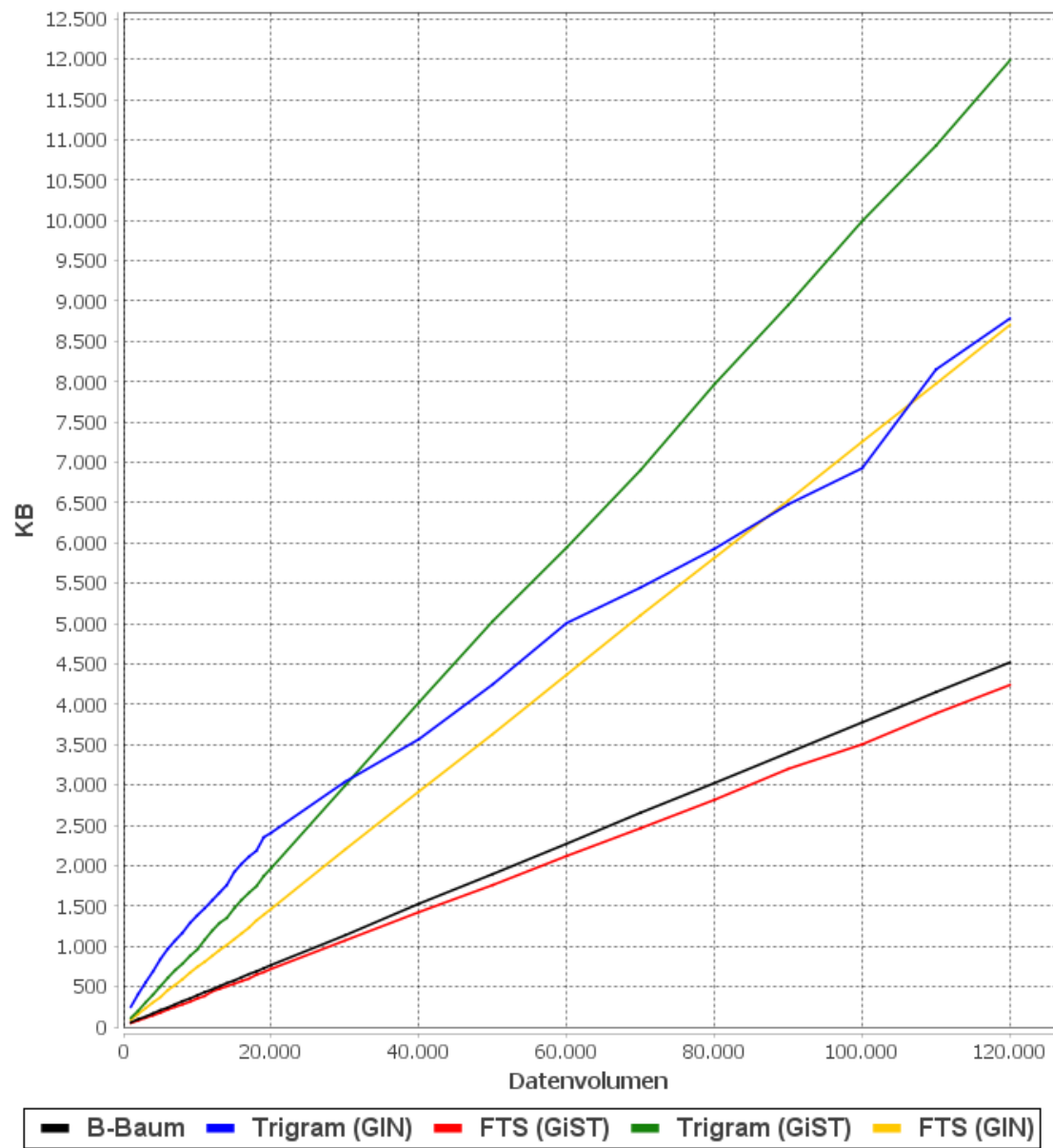


Abbildung 77

## Eidesstattliche Erklärung

Marko, Kovacevic, 882059

Hiermit erkläre ich, dass ich diese Arbeit selbstständig abgefasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

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

.....

Ort, Abgabedatum

.....

Unterschrift (Vor- und Zuname)