Institut für Parallele und Verteilte Systeme
Abteilung Anwendersoftwre
Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Diplomarbeit Nr. 3645

Entwicklung und Bewertung eines relationalen Link Stores

Norman Hood

Studiengang:	Softwaretechnik
Prüfer:	PD Dr. rer nat. habil. Holger Schwarz
Betreuer:	M. Sc. Christoph Gröger
Beginn am:	13.3.2014
Beendet am:	12.9.2014
CR-Nummer:	H.2.7, H.2.4, J.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation, Durability
AdMA	Advanced Manufacturing Analytics
CMIS	Content Management Interoperability Services
CMS	Content-Management-System
DBMS	Datenbank Management System
DW	Data Warehouse
ER-Diagramm	Entity Relationship-Diagramm
GATD	Generic Attribute Table per Data Type
JDBC	Java Database Connectivity
LTA3NF	Link Type Attributes in 3NF
MEA	Melted Elements and Links
MLAA	Melted Links and Attributes
MCAT	Multiple Concrete Attribute Tables
NF	Normalform (3NF = dritte Normalform etc.)
RDF	Resource Description Framework
SCAT	Single Concrete Attribute Table
SGAT	Singe Generic Attribute Table
SGATHIA	Singe Generic Attribute Table with Horizontal Implicit Attributes
SGATHOA	Singe Generic Attribute Table with Horicontal Attributes
SGATIA	Single Generic Attribute Table with Implicit Attributes
URI	Uniform Resource Identifier

1 Einleitung

Der fachliche Hintergrund dieser Arbeit liegt im Projekt Advanced Manufacturing Analytics (AdMA) begründet [1] [2]. AdMA befasst sich mit der Verbesserung von Fertigungsprozessen in Industrieunternehmen. Im Rahmen dieser Fertigungsprozesse fallen viele Daten strukturierten und unstrukturierten Typs an. Bestehende Ansätze zur Informationsgewinnung aus Daten bauen entweder lediglich auf strukturierten Daten oder auf unstrukturierten Daten auf. Diese Unterteilung ist allerdings durch die technischen Gegebenheiten bedingt und verhindert eine integrierte, holistische Sicht auf die Fertigungsprozesse [3]. So können beispielsweise für eine Maschine die zugehörigen Fehlerberichte, gespeichert als Textdokument, zusammen mit ihren Performancemetriken, gespeichert in einem relationalen Data Warehouse (DW), zusammengenommen die Information ergeben, dass diese Maschine ein Prozesshindernis darstellt.

In dieser Arbeit wird diese Problematik angegangen, indem eine Integration strukturierter und unstrukturierter Daten vorgenommen wird. Der gewählte Lösungsansatz orientiert sich dabei an [3], wo in diesem Zusammenhang von einem Deep Data Warehouse gesprochen wird. Zentraler Bestandteil des Deep Data Warehouse ist der sogenannte Link Store, wo semantisch reiche Verlinkungen zwischen den strukturierten und unstrukturierten Daten vorgenommen werden. Auch wenn die Anforderungen für das Deep Data Warehouse aus den Gegebenheiten typischer Fertigungsprozesse von Industrieunternehmen abgeleitet sind, so erhebt dieser Lösungsansatz doch Anspruch auf Allgemeingültigkeit. Damit soll in dieser Arbeit auch nicht der fachliche Hintergrund der Fertigungsprozesse im Vordergrund stehen, sondern mögliche technische Umsetzungen der Verlinkungen, unabhängig vom gegebenen fachlichen Kontext.

Konkret ist das Ziel dieser Arbeit die Konzeption, prototypische Implementierung und Bewertung eines Link Stores auf relationaler Basis zur Verknüpfung strukturierter und unstrukturierter Daten. Hinsichtlich der technischen Umsetzung wurde bereits ein auf einer Graphdatenbank basierender Prototyp für die Verlinkung geschrieben [4], an welchen diese Arbeit als Nachfolgearbeit anknüpft. Damit müssen sich die relationalen Verlinkungen also anhand der Konkurrenzlösung der graphbasierten Verlinkungen messen lassen.

Der Aufbau dieses Dokuments gliedert sich in sechs Kapitel und den Anhang. Kapitel 1 ist die Einleitung und enthält allgemeine Informationen über diese Diplomarbeit für einen schnellen Überblick. Im Grundlagen-Kapitel 2 wird der Kontext der Diplomarbeit vorgestellt. Der Kontext umfasst dabei auf fachlicher Seite das AdMA-Projekt und auf technischer Seite die für diese Diplomarbeit relevanten Technologien. Weiterhin gehört zum Kontext der Diplomarbeit das Kapitel 3, wo verwandte Ansätze vorgestellt werden, graphorientierte Strukturen in relationalen Datenbanken abzubilden. Kapitel 4 enthält die Konzeption des relationalen Link Stores sowohl auf logischer als auch auf physischer Ebene. Es werden verschiedene relationale Lösungsvarianten entworfen und evaluiert und aufbauend auf der Evaluation, wird dann eine Entscheidung getroffen, welche Ansätze weiter zu verfolgen sind. Die Implementierung der entworfenen Ansätze, inklusive der graphbasierten Variante, wird dann in Kapitel 5 beschrieben. Weiterhin werden in Kapitel 5 die Lösungen anhand von Performanceuntersuchungen evaluiert. Den Abschluss der Arbeit bildet das Kapitel 6, welches ein abschließendes Fazit darüber enthält, inwiefern sich die Lösungen als praxistauglich erwiesen haben.

2 Grundlagen

In diesem Kapitel wird sowohl der fachliche als auch der technische Kontext dieser Diplomarbeit vorgestellt. Von fachlicher Seite her ist dies das Forschungsprojekt AdMA und von technischer Seite her geht es um die Integration strukturierter und unstrukturierter Daten und um den Einsatz von relationalen Datenbanken und Graphdatenbanken.

2.1 Advanced Manufacturing Analytics-Plattform

Die AdMA-Plattform ist eine integrierte Business Intelligence-Plattform für datengetriebene Fertigungsprozess-Optimierung [1] [5] [2] [6]. Im Vordergrund steht dabei die Ganzheitlichkeit der Informationsquellen, aus welchen Wissen extrahiert werden soll.

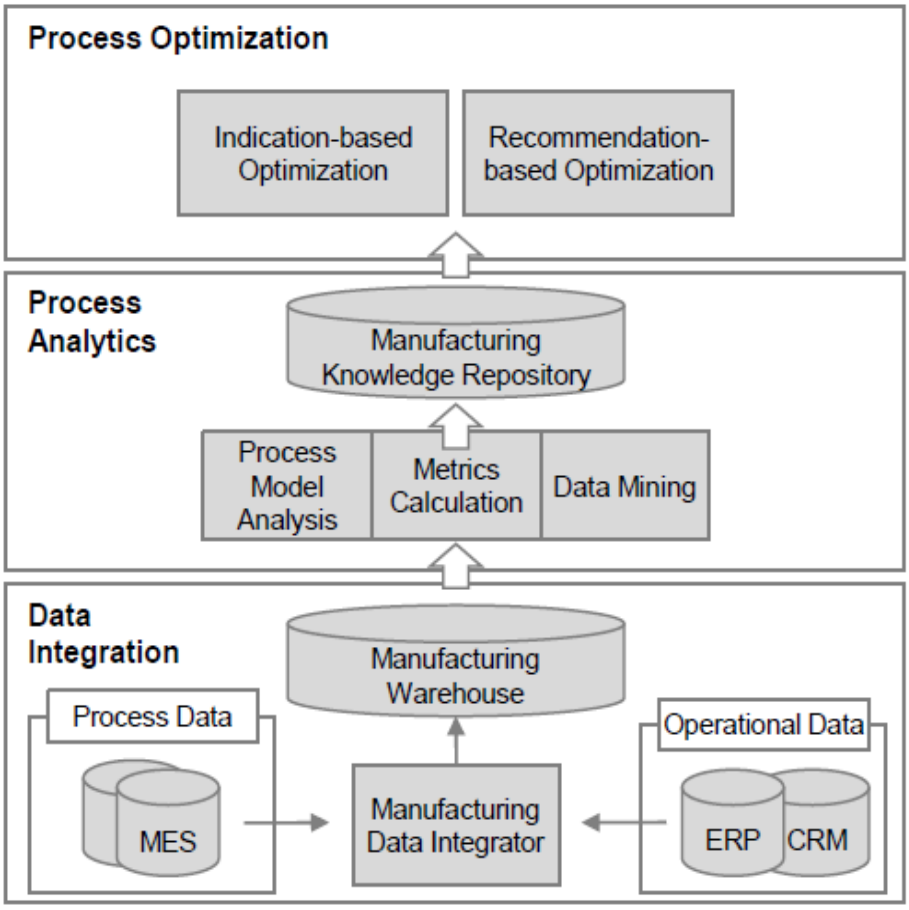

Abbildung 1: Konzeptionelle Architektur von AdMA, in Anlehnung an [1],[6]

Abbildung 1 stellt die AdMA-Architektur auf konzeptioneller Ebene dar. Auf höchster Ebene gibt es dabei die drei Schichten namens „Data Integration“, „Process Analytics“ und „Process Optimization“.

Die Data Integration-Schicht ist dafür verantwortlich, Daten aus unterschiedlichen Datenquellen in einer integrierten Datenbank, dem „Manufacturing Warehouse“, zu vereinigen. Diese Vereinigung geschieht im Manufacturing Data Integrator. Im Rahmen dieser Diplomarbeit wurde das Manufacturing Warehouse zu Debugzwecken mit Testwerten befüllt, anstatt die Daten aus den Quelldaten zu extrahieren. Hinsichtlich der Quelldaten wird die fachliche Unterscheidung zwischen sogenannten „Process Data“ und „Operational Data“ gemacht. Die Process Data umfassen sowohl Ist-Daten auf Fabrikebene als auch Soll-Daten auf Planungsebene. Die Operational Data hingegen beschreiben die Subjekte der Fertigungsprozesse, beispielsweise die verwendeten Maschinen. Die Erfassung dieser unterschiedlichen Aspekte der Fertigungsprozesse dient der angestrebten ganzheitlichen Sicht.

In der Process Analytics-Schicht werden die vorhandenen Daten um Wissen angereichert. Hierzu werden verschiedene Techniken zur Wissensextrahierung wie beispielsweise Data Mining angewendet. Als Ausgangspunkt für die Algorithmen dient das integrierte Manufacturing Warehouse. Da im Manufacturing Warehouse allerdings lediglich strukturierte Daten gespeichert werden, fließt zusätzlich Information aus unstrukturierten Daten ein, welche gewöhnlicherweise in einem Content-Management-System (CMS) gespeichert sind. Dieser Sachverhalt wird in der Graphik allerdings nicht dargestellt. Im Manufacturing Knowledge Repository werden die Analyseergebnisse sowie unstrukturierte Daten gespeichert. Die Analyseergebnisse können dabei in unterschiedlichster Form vorliegen, beispielsweise textuell oder in einer XML-Datei.

Nachdem in der Process Analytics Schicht das notwendige Wissen aus den Daten extrahiert wurde, ist die Process Optimization-Schicht dann dafür verantwortlich, dass die Fertigungsprozesse optimiert werden. Unterteilt wird in die Indication-based optimization und die Recommendation-based optimization. Die Indication-based optimization liefert dem Nutzer Indikatoren zum Verständnis bestimmter Prozess-Attribute.

Auf dieser Basis aufbauend, kann der Nutzer dann Prozessverbesserungen vornehmen. Die Recommendation-based optimization geht weiter als die Indication-based optimization, da in ihr bereits konkrete Schritte für die Prozessverbesserung vorgeschlagen werden.

Die Ergebnisse dieser Arbeit betreffen das Manufacturing Knowledge Repository, wo die Anforderung auftritt, strukturierte und unstrukturierte Daten flexibel interagieren zu können und auf die Analyseergebnisse in einer kohärenten Art und Weise zuzugreifen.

2.2 Integration strukturierter und unstrukturierter Daten

Im letzten Abschnitt wurde das Manufacturing Knowledge Repository verbunden mit der Notwendigkeit eines Konzepts für die Integration strukturierter und unstrukturierter Daten vorgestellt. In diesem Abschnitt werden dann allgemeine Anforderungen an eine derartige Integrationsarchitektur vorgestellt und argumentiert, weshalb bestehende Ansätze diese Anforderungen nur unbefriedigend erfüllen. Das Nicht-Vorhandensein befriedigender Lösungen dient dann als Motivation für das Deep Data Warehouse, welches im nächsten Abschnitt detailliert vorgestellt wird.

Gröger et al. identifizieren die vier Anforderungen Flexibilität, Anreicherung, Unifizierung und Kompatibilität für die Integration strukturierter und unstrukturierter Daten [3]. Flexibilität bezieht sich dabei auf die Möglichkeit, flexibel neuartige Datenquellen hinzufügen zu können. Falls erst eine komplexe Schema-Transformation benötigt wird, widerspricht dies der Anforderung der Flexibilität. Unter Anreicherung wird verstanden, dass die Daten nicht nur integriert werden, sondern um weiteres Wissen angereichert werden können. Die Unifizierung bezieht sich darauf, dass der Nutzer die Daten in einer einheitlichen Art und Weise abfragen kann. Dies bedeutet beispielsweise, dass Dimensions-Hierarchien im Data Warehouse auf die gleiche Art traversiert werden können wie andere Verlinkungen. Auch sollte es für die Anfrage keine Rolle mehr spielen, ob ein Element aus dem DW oder dem CMS stammt. Die Anforderung der Kompatibilität bedeutet, dass die integrierte Sicht über die anderen Systeme gelegt

wird, ohne dass jene dazu angepasst werden müssen. Insbesondere das bereits bestehende Data Warehouse wird also nicht extra präpariert.

In [3] werden neben dem im Deep Data Warehouse verfolgten Ansatz auch andere bestehende Ansätze vorgestellt, strukturierte und unstrukturierte Daten zu vereinigen, nämlich:

- Unstrukturierte Daten in Data Warehouses
- Föderierte Informationssysteme
- Link-basierte Integration

Unstrukturierte Daten in Data Warehouses zu speichern ist der klassische Ansatz zur Integration. In diesem Zusammenhang wird auch von Data Warehouses 2.0 gesprochen, worin die Problematik der unstrukturierten Daten adressiert wird [7].
Gröger et al. lehnen diesen Ansatz jedoch aus zwei Gründen ab. Zum einen muss das Data Warehouse-Schema kostspielig erweitert werden muss und zum anderen, sind die durchführbaren Analyse-Szenarien sehr eng umrissen und müssen auf auf im Vorhinein bekannten Attributen aufbauen.

Ein föderiertes Informationssystem [8] bietet Zugriff auf mehrere heterogene Datenquellen, unterscheidet sich aber vom klassischen Data Warehouse-Ansatz dadurch, dass die Daten nicht aus den Quellsystemen heraus kopiert werden. Statt einer materialisierten Integration findet also nur eine virtuelle Integration bei Nutzeranfrage statt. Da die Quelldaten typischerweise in unterschiedlichen Schemata abgespeichert werden, müssen jene Schemata zu einem integrierten Schema vereinigt werden. Diese Schema-Integration wird für das Deep Data Warehouse nicht benötigt.

Die Link-basierte Integration [9] [10], wählt wie das Deep Data Warehouse das Konzept der Verlinkungen. Allerdings sind die Links hier nicht um Attribute angereichert und werden auch nicht als separate Entitäten abgespeichert. Der Kontext der Bioinformatik, wo die Link-basierte Integration verwendet wird, ähnelt, so wie er in [9] und [10] vorgestellt wird, demjenigen der Fertigungsprozesse. Die Quelldaten sind eher umfangreich, weisen eine hohe Komplexität auf, liegen in unterschiedlichsten struktu-

rierten und unstrukturierten Formaten vor, können sich ändern und sind, von ihrer Semantik her, hochgradig vernetzt. Von dieser Ausgangsbasis wäre die Bioinformatik auch ein geeigneter Kandidat für das Deep Data Warehouse.

2.3 Deep Data Warehouse

Basierend auf [3] soll im Folgenden das Deep Data Warehouse im Detail vorgestellt werden. Das Deep Data Warehouse basiert auf semantisch reichen Verlinkungen. Diese Verlinkungen können dabei zwei Zwecken dienen, nämlich zum einen der Integration strukturierter und unstrukturierter Daten und zum anderen der Anreicherung der bestehenden Daten durch zusätzliches Wissen, welches beispielsweise durch Data Mining Algorithmen gewonnen wurde.

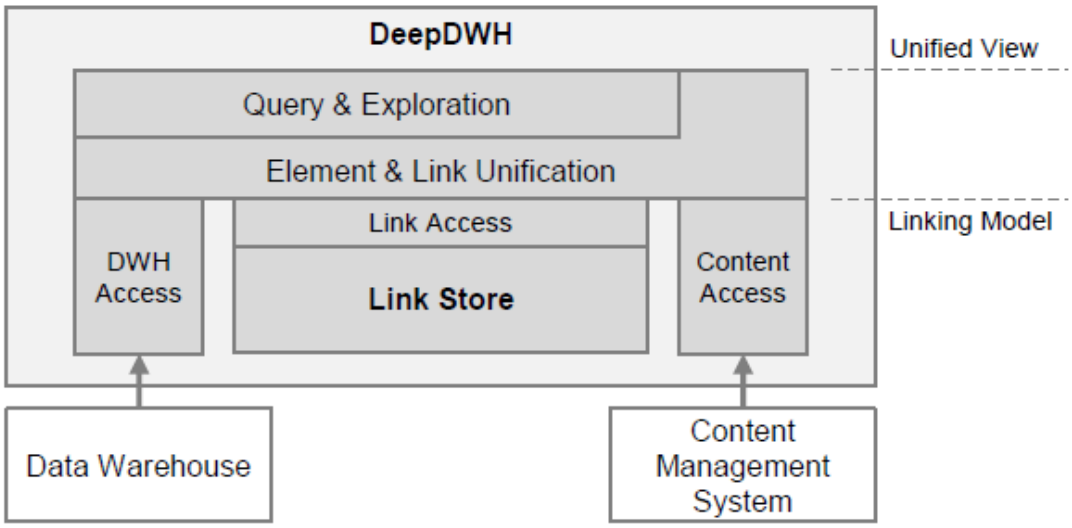

Abbildung 2: Generelle Architektur des Deep Data Warehouse, entnommen aus [3]

Abbildung 2 stellt die generelle Architektur des Deep Data Warehouse dar. Außerhalb des Deep Data Warehouse befinden sich die strukturierten Daten in einem externen Data Warehouse und die unstrukturierten Daten in einem CMS. Jene Quellsysteme werden nicht angepasst und es besteht keine Notwendigkeit, sie auf ein integriertes Schema zu vereinigen. Stattdessen werden Verlinkungen zwischen Elementen des CMS und des DW vorgenommen. Jene Verlinkungen geschehen im Link Store, welcher bislang in einer Graphdatenbank implementiert ist und in dieser Arbeit relational implementiert wird. Im Data Warehouse gibt es bereits interne Warehouse-Links wie

beispielsweise die Zuweisung verschiedener Produkte zu einer Produktgruppe. Um der Query & Exploration-Komponente eine integrierte Sicht auf die Links des Link-Stores und der Data Warehouse-Links zu bieten, werden sie in der Element & Link Unification-Komponente vereinigt. Damit kann dann über die Links unabhängig davon navigiert werden, ob sie vom Link Store oder vom Data Warehouse direkt kommen. Auch die Elemente werden vereinigt, sodass es keine Rolle mehr spielt ob ein Element vom CMS oder vom DW stammt.

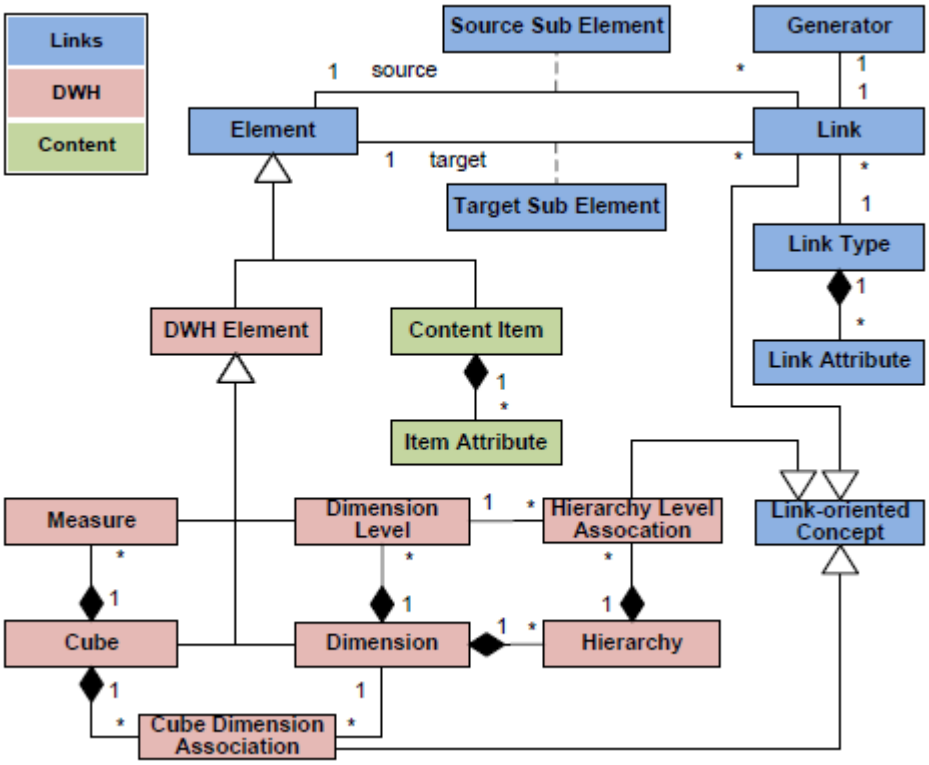

Abbildung 3: Generisches Link-Modell auf konzeptioneller Ebene, entnommen aus [3]

Abbildung 3 stellt das generische Link-Modell auf konzeptioneller Ebene dar. Die Linkstruktur besteht aus Knoten, genannt Elements und gerichteten Kanten, genannt Links. Ein Element repräsentiert dabei eine Entität aus einem der Quellsysteme, kann also beispielsweise ein Dokument aus dem CMS oder einen Angestellten aus dem Data Warehouse repräsentieren. Ein Link verknüpft ein Start-Element mit einem Ziel-Element. Jeder Link ist von einem bestimmten Linktyp und der Linktyp des Links legt dann fest, welche Attribute der Link enthält. Abbildung 4 zeigt ein Beispiel für einen Link , welcher einen Failure Report mit einer Machine verknüpft. Der Linktyp ist hier

Explains und demgemäß enthält der Link passende Attribute für diesen Linktyp. Welche Verlinkungen Sinn machen, hängt vom konkreten Anwendungsfall und der konkreten DW-Struktur ab. Entsprechend dem Kontext der Fertigungsprozesse werden in dieser Ausarbeitung Links gewählt, welche für Fertigungsprozesse Sinn ergeben.

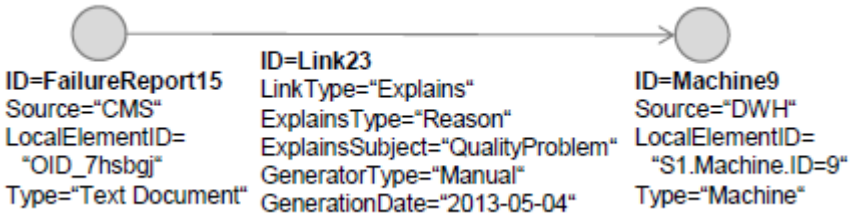

Abbildung 4: Beispielhafter Link für Deep Data Warehouse, entnommen aus [3]

2.4 Relationale Datenbanken und Graphdatenbanken

Da für die Implementierung des Deep Data Warehouse in dieser Arbeit, neben den relationalen Varianten, eine graphbasierte Variante untersucht wurde, soll in diesem Abschnitt ein Vergleich von Graphdatenbanken und relationalen Datenbanken erfolgen. Die relationale Datenbank-Technologie ist, bedingt durch die SQL-Standardisierung, relativ homogen, sodass sich einfach allgemeine Aussagen treffen lassen [11]. Diese Homogenität ist jedoch im Falle der Graphdatenbanken nicht gegeben [12], sodass, wenn in diesem Abschnitt von Graphdatenbanken gesprochen wird, verstärkt Neo4j und das darin verwendete Property Graph-Modell im Vordergrund stehen soll. So ist es beispielsweise ein Kennzeichen von NoSQL, zu welchem die Graphdatenbanken gehören, auf ACID zu verzichten[13], was jedoch im Falle von Neo4j nicht gegeben ist [14].

In [15] wird ein Vergleich zwischen Neo4j und der relationalen Datenbank MySQL aufgestellt. Als Vergleichskriterien werden dabei folgende Punkte identifziert:

- Performance
- Reifegrad
- Einfachheit der Programmierung
- Flexibilität
- Sicherheitskonzepte

Das Performanceverhalten von Graphdatenbanken, im Vergleich zu relationalen Datenbanken, hängt von den unterschiedlichen zugrundeliegenden Datenmodellen ab. Im Falle der relationalen Datenbanken ist dies das relationale Modell und im Falle von Graphdatenbanken ist dies ein Graph. Der schematische Aufbau relationaler Datenbanken ist in Abbildung 5 dargestellt. Eine relationale Datenbank besteht aus einer Ansammlung von Tabellen, Relationen genannt. Jede Tabelle besteht aus einer Anhäufung gleichartig aufgebauter Tupel, welche die Zeilen der Tabelle darstellen. Ein Eintrag eines Tupels wird dabei über sein Attribut, also den zugehörigen Spaltennamen identifiziert.

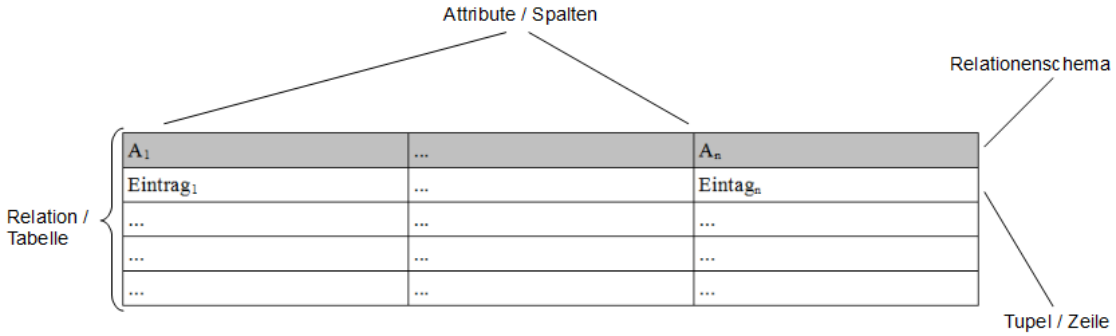

Abbildung 5: Schematischer Aufbau relationaler Datenbanken

Im Falle einer Graphdatenbank ist das physikalisch zugrundeliegende Datenmodell ein Graph, bestehend aus Knoten und Kanten. Ein Knoten repräsentiert dabei eine Entität wie „Maschine 1“ und eine Kante repräsentiert eine Beziehung zwischen zwei Entitäten. Im Falle von Neo4j wird ein Property-Graph verwendet, was bedeutet, dass sowohl Knoten als auch Kanten Eigenschaften, Properties genannt, haben können. Seit Neo4j 2.0 lassen sich zusätzlich zur Klassifzierung der Knoten und Kanten Label verwenden [16]. In einem Graph kennt nun jeder Knoten seine benachbarten Knoten. Damit ist die Laufzeit für einen Sprung zum benachbarten Knoten also konstant, unabhängig von den insgesamt gespeicherten Datensätzen. Im relationalen Modell hingegen muss der Sprung zu einer in Beziehung liegenden Entität über Joins hergestellt werden. Die Performance hängt dabei stark von den verwendeten Indexstrukturen ab und schlechtestenfalls erhöht jede für den Join benötigte Tabelle die Laufzeit um den Multiplikator der in ihr gespeicherten Tupel.

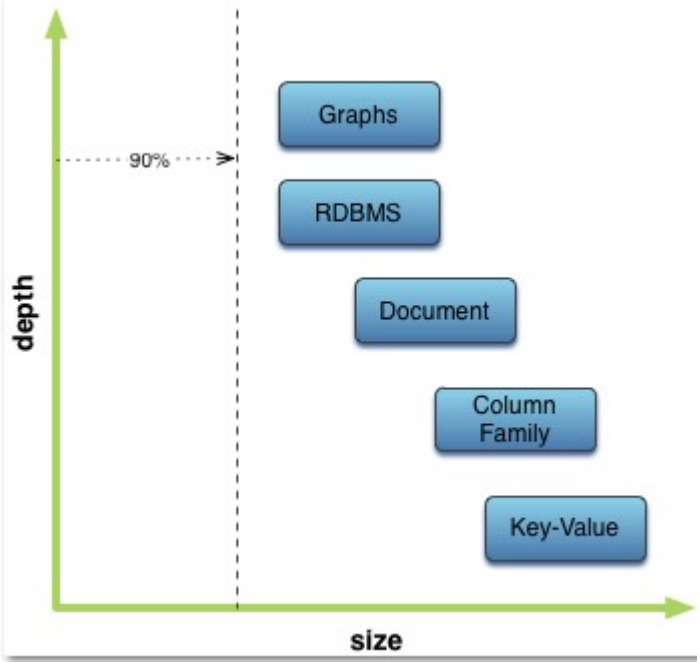

Abbildung 6: Anwendungsgebiet von Graphdatenbanken im Vergleich, entnommen von Neo4j.org

Die Neo4j-Entwickler geben das Anwendungsgebiet von Graph-Datenbanken gemäß Abbildung 6 an. In der Abbildung bezieht sich size dabei auf die Qunatität der gespeicherten Daten und depth auf die Komplexität der Anfragen, also im Wesentlichen darauf, wie viele Entitäten für die Anfrage berührt werden müssen. „Document“, „Column Family“ und „Key-Value“ sind Bezeichner für andere NoSQL-Speichertechnologien. Die 90%-Aussage bezüglich der Größe sagt aus, dass 90% aller Anwendungsfälle ein derartig geringes Volumen aufweisen, dass das Volumen keine Beschränkung darstellt. Andere NoSQL-Technologien sind, gemäß der Graphik, für große Volumina optimiert, während das Einsatzgebiet von Graphdatenbanken sich nicht auf größere Volumina, sondern auf hochgradig vernetzte Daten bezieht.

Was den Reifegrad betrifft, sind die relationalen Datenbanken den Graphdatenbanken klar überlegen. Seit den 1980er-Jahren sind die relationalen Datenbanken vorherrschend [17]. Neo4j hingegen wurde erst 2010 in der Version 1.0 veröffentlicht [18], was zeitlich ungefähr mit dem Beginn der NoSQL-Bewegung übereinstimmt [19]. Die Verwendung der eher unausgereiften Graphdatenbanken bringt einige Nachteile mit sich. Für die relationalen Datenbanken dient SQL als standardisierte Abfragesprache,

welche, mit gewissen anbieterspezifischen Anpassungen, von sämtlichen Datenbank Management Systemen (DBMS) verstanden wird. Für Graphdatenbanken existiert keine derartige standardisierte Abfragesprache. Abfragen werden in Neo4j mittels der Abfragesprache Cypher durchgeführt. Cypher ist allerdings noch konstanten und weitreichenden Veränderungen ausgesetzt, zuletzt mit der Neo4j 2.0-Version [20]. Weiterhin bringt der geringe Reifegrad von Graphdatenbanken, verbunden mit einer fehlenden Standardisierung, das Problem mit sich, dass man sich an einen konkreten Anbieter bindet.

Die Einfachheit der Programmierung hängt von der konkreten Abfrageart ab. Gröger et al. schlagen eine Unterteilung in verschiedene Anfrageklassen vor [3]. Jene wären die beziehungsorientierten Anfragen und die navigationsorientierten Anfragen, unterteilt in Ein-Schritt-Anfragen und Mehr-Schritt-Anfragen. Bei den beziehungsorientierten Anfragen geht es darum herauszufinden, wie die Beziehungen zwischen bestimmten konkreten Elementen sind. Beispielsweise dient die Cypher-Anfrage von Listing 1 aus dem Neo4j-Manual dazu, sämtliche Knoten zu finden, welche ein oder zwei Schritte entfernt vom Knoten „Martin Sheen“ sind.

```
MATCH(martin{name: „Martin Sheen“}) - [:ACTED_IN*1..2]-(x) RETURN x
```

Listing 1: Cypher-Anfrage für nahe Knoten, entnommen von Neo4j.org

Wie ersichtlich, sind dearartige Anfragen mit Neo4j sehr einfach umzusetzen. Auch für das relationale Modell gibt es Ansätze, beziehungsorientierte Anfragen umzusetzen [21], was hier allerdings deutlich schwieriger ist. Die navigationsorientierten Anfragen navigieren zwischen verschiedenen Elementen unter Berücksichtigung von Element-Restriktionen und Link-Restriktionen. Hierbei handelt es sich also um die klassischen Anfragen, wie sie gewöhnlicherweise an relationale Datenbanken gestellt werden. Die Einschritt-Anfragen verbinden dabei lediglich zwei Elemente, während die Mehrschritt-Anfragen eine unbegrenzte Anzahl an Elementen verbinden können.

Unter Flexibiität versteht [15], wie der Datenbank-Typ sich außerhalb des Bereiches, für welchen er konzipiert wurde, verhält. Wie in dieser Diplomarbeit vorgeführt, können Graphstrukturen auch mit relationalen Datenbanken abgebildet werden, während

die Graphdatenbanken für Graphstrukturen optimiert sind. Auch unter das Stichwort der Flexibilität fällt die Möglichkeit, das Datenmodell einfach anpassen zu können. Ein Kennzeichen von Graphdatenbanken ist hier, bedingt durch das Nicht-Vorhandensein eines Schemas, sehr flexibel zu sein. Relationale Tabellen hingegen sind an ein festes Schema gebunden. Im Rahmen dieser Arbeit wird versucht, durch eine generischere Ausgestaltung der Tabellen eine hinreichende Flexibilität für die relationalen Lösungen herzustellen.

Was die Sicherheitskonzepte betrifft, bringen sämtliche gebräuchlichen relationalen Datenbankmanagementsysteme ein ausgefeiltes Berechtigungssystem mit sich. Eine derart eingebaute Security ist in Neo4j nicht vorhanden [22]. Die Security muss damit also vollständig auf Applikationsebene bzw. auf Server-Ebene erfolgen.

3 Verwandte Arbeiten: Graphorientierte Strukturen in relationalen Datenbanken

Eine Literaturrecherche bezüglich einer allgemeinen Speicherung von Graph-Strukturen in im relationalen Modell gab wenig verwertbare Resultate, da diesbezüglich die Themenstellung zu allgemein formuliert ist. Ansätze wie in [15] sagen letzendlich nicht mehr aus, als Knoten und Kanten in zwei verschiedenen Tabellen zu speichern, während in [23] ein DBMS um typische Graph-Operationen erweitert wird. Von daher sollen im Folgenden nicht Graphstrukturen im Allgemeinen, sondern spezielle Graphstrukturen betrachtet werden und untersucht werden, ob sich die Charakteristika der relationalen Speicherung solch einer speziellen Graphstruktur verallgemeinern und insbesondere auf die für diese Diplomarbeit relevanten Property-Graphen übertragen lassen.

In Anlehnung an [12], [24] und [25] identifizieren Gröger et al. drei spezielle Graph-Typen, wie sie in Graphdatenbanken Verwendung finden, nämlich den Simple Graph, den Property Graph und den RDF Graph [3]. Ein Simple Graph wird hierbei als ein gerichteter Graph aufgefasst, bei welchem die Knoten und Kanten jeweils mit Bezeichnern versehen werden können. Ein Property Graph erweitert den Simple Graph dahingehend, dass sowohl Knoten als auch Kanten zusätzlich noch mit Properties versehen werden können. Eine Property ist dabei ein Schlüssel-Wert-Paar.

Anders als für die beiden eben genannten Graphstrukturen, gibt es ernsthafte Untersuchungen für eine relationale Repräsentation von RDF-Graphen. In RDF werden Aussagen getroffen, wobei jede Aussage aus drei Bausteinen, nämlich Subjekt, Prädikat und Objekt besteht [26]. Diesbezüglich spricht man auch vom sogenannten RDF-Tripel. Das Subjekt ist dabei eine Ressource, welche durch das Objekt beschrieben wird. Das Objekt kann hierbei eine weitere Ressource oder ein Prädikat sein. Die Art der Beziehung zwischen Subjekt und Objekt wird dabei durch das Prädikat ausgedrückt, welches ebenfalls eine Ressource darstellt. Für eine eindeutige Bezeichung werden die Ressourcen als URIs augedrückt. Eine Ressource ist dergestalt definiert, dass über sie Aussagen getroffen werden können. Damit können also auch über die Prädikate Aussagen getroffen werden.

Gemäß [27] lassen sich folgende Gründe finden, RDF relational abzubilden:

1. Transaktionsunterstützung relationaler DBMS
2. Ausgereifte Unterstützung gleichzeitigen Zugriffs relationaler DBMS.
3. Ausgereifte Security-Restriktionen relationaler DBMS.

Punkt eins und zwei lassen sich mit Neo4j ebenfalls realisieren, während Punkt drei von Neo4j nicht unterstützt wird.

Demgegenüber werden folgende Herausforderungen einer relationalen Abbildung gegenübergestellt:

1. Sehr viele unterschiedliche Typen benötigen sehr große relationale Schemata.
2. Das Schema ist kontinuierlichen Änderungen unterworfen.
3. Die Daten sind dünn gestreut. Jeder Typ kann also tausende Eigenschaften haben, aber eine konkrete Entität des Typs wird gewöhnlich nur einen Bruchteil dieser Eigenschaften aufweisen.

Die ersten beiden Punkte lassen sich auch auf die Property-Graphen übertragen. In dieser Diplomarbeit werden hierbei als Lösungsansatz sehr allgemeingehaltene, generische Schemavarianten entwickelt. Das Auftreten des dritten Punkts hängt vom konkreten Anwendungsfall ab. Werden sehr konkrete Schemata benutzt, wird dieser Punkt zum Auftreten vieler Null-Werte führen. Auch hier lässt sich durch allgemein gehaltene Schemata dieses Problem jedoch umgehen.

Subjekt	Prädikat	Objekt
Bulldozer	ist ein	Produkt
Kunden	kaufen	Produk
Produkte	haben	Bestandteil

Abbildung 7: Einfachster Ansatz für relationale RDF-Speicherung

Abbildung 7 zeigt den naiven Ansatz, RDF darzustellen. Jede RDF-Aussage wird hier als ein eigenes Tupel in der Tabelle gespeichert. Das Problem der Schema-Flexibilität wird durch diesen Ansatz gelöst, allerdings werden für gewöhnliche Anfragen zu viele

Joins benötigt [28]. Dieser Ansatz stimmt, vom Grundgedanken her, mit dem im Rahmen dieser Diplomarbeit entwickelten Ansatz der generischen Attributstabellen überein.

Bornea et al. stellen in [28] auch den Ansatz vor, die große RDF-Tabelle in kleinere Tabellen aufzuspalten. Als Aufspaltungskriterien werden dabei die Subjekte oder die Prädikate vorgeschlagen. Die Aufnahme neuer Subjekte oder Prädikate führt dabei jedenfalls zu neuen Relationen und damit zu einer Schemaänderung.

entry	*spill*	$pred_1$	val_1	$pred_2$	val_2	$pred_3$	val_3	...	$pred_k$	val_k
Charles Flint	0	died	1934	born	1850	founder	IBM	...	*null*	*null*
Larry Page	0	board	Google	born	1973	founder	Google	...	home	Palo Alto
Android	1	developer	Google	version	4.1	kernel	Linux	...	preceded	4.0
Android	1	*null*	*null*	*null*	*null*	graphics	OpenGL	...	*null*	*null*
Google	0	industry	lid:1	employees	54,604	*null*	*null*	...	HQ	Mtn View
IBM	0	industry	lid:2	employees	433,362	*null*	*null*	...	HQ	Armonk

Abbildung 8: RDF horizontal aufgespannt, entnommen aus [28]

Abbildung 8 zeigt einen Ausschnitt aus der in [28] entwickelten Lösung, die Daten in einer horizontal aufgespannten Weise darzustellen. Es werden mehrere Aussagen über ein Subjekt in einer Zeile zusammengefasst. Dabei ist „entry“ das Subjekt und „val_i“ ist das i-te Objekt. Da die Tabelle so, je nach Anwendungsfall, theoretisch beliebig breit werden kann, werden, wenn es zu viele Aussagen über ein Subjekt gibt, die weiteren Aussagen ins nächste Tupel geschrieben. Dies wird dann in der „spill“-Spalte vermerkt. Die Grundidee davon diente als Ispiration für den Ansatz dieser Arbeit, die Attribute horizontal aufzuspannen. Agrawal et al. lehnen einen horizontalen Ansatz jedoch ab [29]. Grund hierfür ist hauptsächlich die enorme Anzahl an dünn besetzten Spalten. Dies führt damit dann zu Performanceeinbußen, wenn nur ein geringer Teil der enorm vielen Spalten für die Anfrage benötigt wird. Wie in Abbildung 8 ersichtlich, werden, wenn es zu wenig Aussagen über ein Objekt gibt, die verbleibenden Spalten mit Null aufgefüllt. Allerdings dient die Aufteilung der Attribute über mehrere Spalten, wie in [28] praktiziert, dazu, dem Problem der dünn besetzten Spalten entgegenzuwirken

4 Konzeption des relationalen Link Stores

In diesem Kapitel werden relationale Lösungsvarianten für den Link Store entworfen und anhand eines herausgearbeiteten Anforderungskatalogs evaluiert. Die vielversprechendsten Lösungen werden dann auf physischer Ebene verfeinert, um eine Implementierung zu ermöglichen.

4.1 Konzeptionelle Linkstruktur

Sinn dieses Abschnitts ist, die Linkstruktur auf konzeptioneller Ebene zu definieren, um daraus dann mögliche relationale Lösungsvarianten auf logischer Ebene abzuleiten. Eigentlich wird die Linkstruktur bereits in [3] auf konzeptioneller Ebene definiert, aber dort wird von den konkreten Attributen eines Links abstrahiert, sodass Interpretationsspielraum für die logische Ebene besteht. Darum soll im Folgenden aus der logischen Modellierung für Graphdatenbanken, definiert in [3], ein hinreichend genaues konzeptionelles Modell abgeleitet werden. Abbildung 9 stellt das konzeptionelle Link-Modell als ER-Diagramm dar, auf welchem für die logischen relationalen Varianten aufgebaut wird.

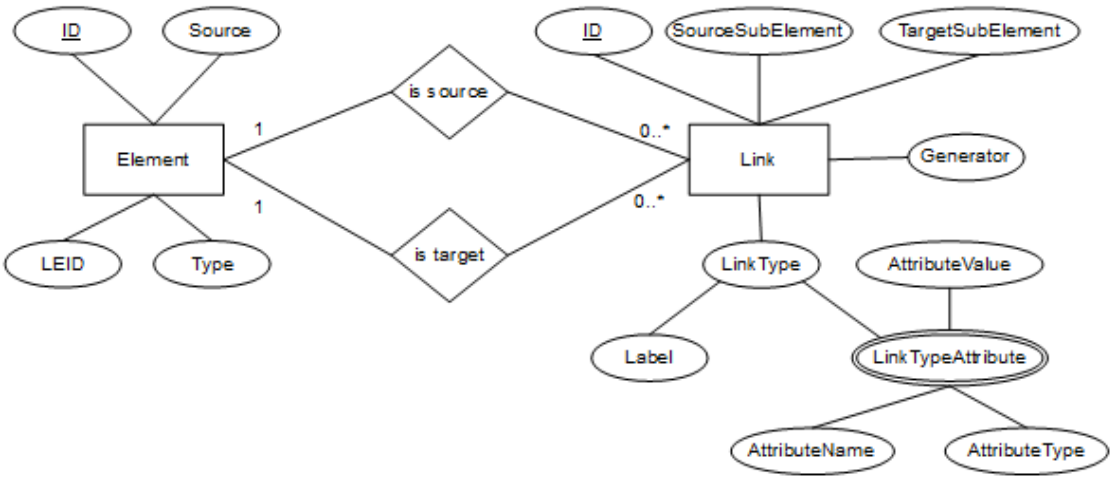

Abbildung 9: Linkstruktur auf konzeptioneller Ebene

Wie in 2.3 beschrieben, gibt es Elemente und Links, welche die Elemente verknüpfen. Sowohl Elemente als auch Links sind um Attribute angereichert. Links sind von einem bestimmten Linktyp, angegeben durch ihr Label und, dem Linktyp entsprechend, enthalten sie eine Menge an Attributen. Hinsichtlich des Generators wird in dieser Arbeit von [3] abgewichen. Eigentlich ist die Struktur eines Generators identisch zu einem

Linktyp aufgebaut, aber um die Lösungen nicht, ohne damit verbundenen Erkenntnisgewinn, aufzublähen, wird der Generator in dieser Arbeit auf ein einzelnes Attribut abgebildet.

4.2 Anforderungskatalog an die Lösungsvarianten

Nachem im Kapitel 3 Gütekriterien bzw. mögliche Schwierigkeiten für die relationale Abbildung von RDF herausgearbeitet wurden, sollen diese Erkenntnisse nun auf die relationale Abbildung für Links übertragen werden. Die Kriterien werden dabei hinreichend detailliert heruntergebrochen, um nachprüfbare Aussagen tätigen zu können. Auf höchster Ebene sollen, in Anlehnung an [30], die Anforderungen Wartbarkeit, Performance, Handhabbarkeit und Speicherplatzverbrauch stehen..

Die Wartbarkeit ist dahingehend relevant, dass, sowohl komplett neue Linktypen einfach neu hinzugefügt werden können sollen, als auch bestehende Linktypen abgeändert werden können sollen, beispielsweise durch Hinzufügung eines neuen Linktyp-Attributs. In diesem Sinne gilt ein Schema also als wartbar, wenn Änderungen am Linktyp-Datenmodell möglichst einfach ins relationale Modell übertragen werden können, ohne dass dort viele Anpassungen vorgenommen werden müssen. Entsprechend den Anforderungen von [3] ist Wartbarkeit damit ein zentrales Kriterium, da einfach neue Analysealgorithmen mit neuartigen Links hinzugefügt werden können sollen.

Als Indizien für die Performance sollen zwei Kriterien herangezogen werden. Jene wären zum einen eventuell benötigte Cast-Funktionen und zum anderen die Anzahl der benötigten Joins bei Anfragen.

Falls ein Schema in einer Produktivumgebung nur sehr schwierig einzusetzen ist, soll dies negativ im Punkt der Handhabbarkeit vermerkt werden. Ein Schema gelte als negativ handhabbar, wenn seine Verwendung unverhältnismäßig komplexe SQL-Anfragen erforderlich mache. Außerdem soll, wenn sich für das aufrufende Programm eine semantische Mehrdeutigkeit ergibt, dies negativ vermerkt werden.

Zur Bewertung des Speicherplatzverbrauchs sollen zwei quantifizierbare Kriterien herangezogen werden, nämlich zum einen die Anzahl der benötigten Null-Werte im Schema und zum anderen mögliche Redundanzen in den gespeicherten Daten. In DB2 kann der Speicherplatzverbrauch durch Verwendung vieler Null-Werte dadurch umgangen werden, dass Value Compression aktiviert wird [31], was für die Performance kaum Nachteile hat. Da die Lösung allerdings nicht derart auf DB2-Interna abgestimmt sein soll, sollen viele Null-Werte trotzdem als negativ vermerkt werden.

4.3 Relationale Lösungsvarianten

Auf logischer Ebene wurden, in Anlehnung an [30] und der gewonnenen Erkenntnisse aus Kapitel 3, verschiedene Schemavarianten entwickelt, welche sich teilweise ausschließen und teilweise zusammen kombinierbar sind.

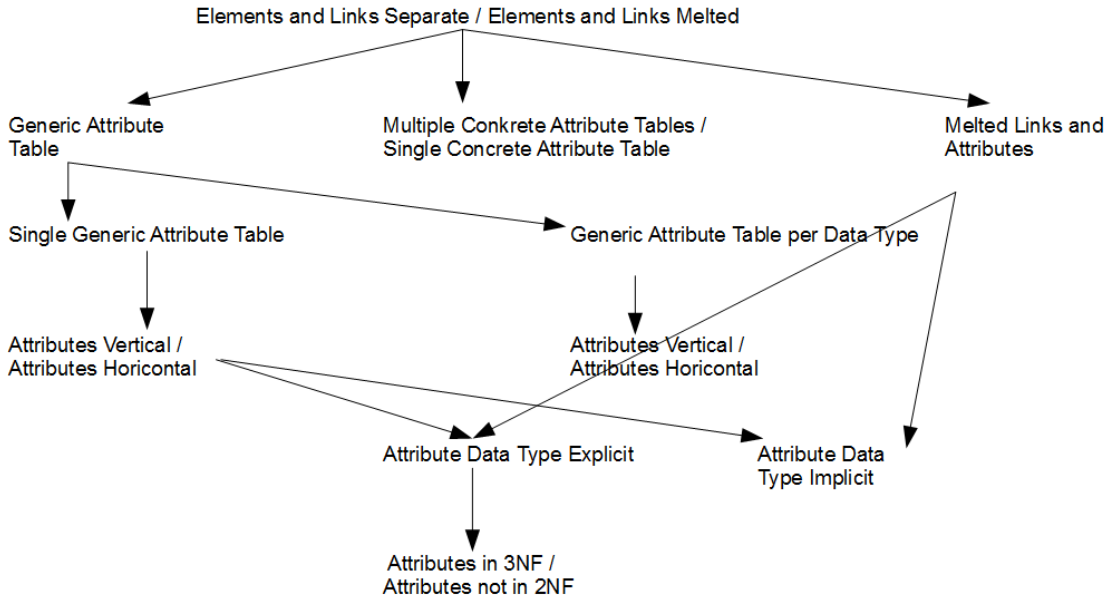

Abbildung 10: Relationale Lösungsvarianten auf logischer Ebene

Die Graphik in Abbildung 11 stellt die Varianten dar. Sie ist so zu lesen, dass entlang eines Pfades beliebig kombiniert werden kann. Man beginnt also bei der Wurzel und trifft dann schrittweise Entscheidungen, bis man an einem Blattknoten angelangt ist. Wo die Entscheidung die weitere Vorgehensweise nicht einschränkt, wurde die Entscheidung in den Knoten geschrieben und wo die Entscheidung die weitere Vorgehensweise einschränkt, wurden verschiedene abzweigende Pfade aufgemacht. Wenn im

Folgenden die Lösungen vorgestellt werden, werden, der Übersichtlichkeit halber, stets konkrete Lösungen vorgestellt, auch wenn die Lösungsvarianten keine Aussage darüber treffen, wie ein anderer Teil des Schemas darzustellen ist.

Um die Lösungsmöglichkeiten zu verdeutlichen, sollen sie anhand eines Fallbeispiels, entnommen aus [30], mit einer einfachen Linkstruktur und einer einfachen Anfrage durchgespielt werden.

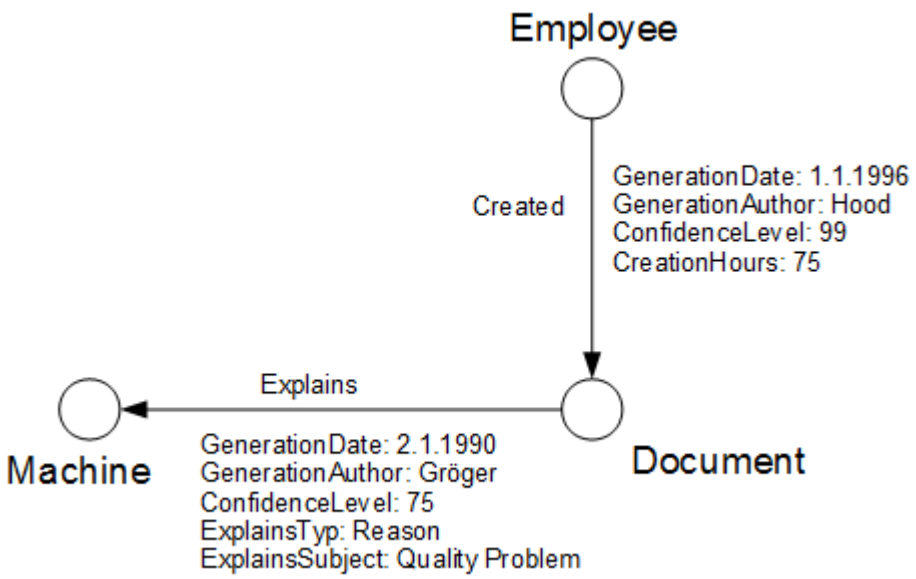

Abbildung 11: Linkstruktur für Fallbeispiel

Abbildung 11 stellt die Linkstruktur des Fallbeispiels dar. Vorhanden ist also ein Dokument, welches von einem Angestellten erzeugt wurde und das gewisse Aussagen über eine Maschine trifft.

Die Beispielanfrage für das Fallbeispiel sei folgendermaßen definiert:
Erhalte alle Dokumente, welche die Maschine Nummer 1 erklären, unter folgenden Einschränkungen:

- ConfidenceLevel >= 50
- ExplainsType = 'Reason'

Gemäß der Klassifikation von Kapitel 2.4 handelt es sich dabei um eine navigationsorientierte Ein-Schritt-Anfrage. Das Durchspielen der Beispielanfrage für sämtliche

entworfenen Lösungsvarianten dient als Machbarkeitsevaluation und der Identifizierung spezieller Lösungsmerkmale. Als Indiz für die Performance soll die Anzahl der benötigten Joins für derartig gestellte Anfragen abgeleitet werden und benötigte Casts vermerkt werden. Weiterhin wird anhand der Beispielanfrage ersichtlich, wie umständlich Anfragen für die einzelnen Lösungsvarianten handhabbar sind.

Eine Kategorisierung der Lösungsvarianten ist nach verschiedenen Kriterien möglich. Im Folgenden soll auf höchster Ebene nach den Kategorien „Generische Attributstabelle“ und „Konkrete Attributstabelle“ kategorisiert werden. Eine Sonderstellung nimmt dabei „Elements and Links Separate / Elements and Links Together“ ein, da jene Entscheidung mit beiden der genannten Kategorien kompatibel ist.

4.3.1 Lösungskategorie Generische Attributstabelle

Sinn der Lösungskategorie „Generische Attributstabelle“ ist eine möglichst generische Darstellung der Attribute, wodurch eine einfache Erweiterbarkeit um neue Linktypen oder um weitere Attribute zu bereits vorhandenen Links bezweckt wird. „Generisch“ bedeutet hierbei, dass das Datenbankschema so allgemeingehalten ist, dass neuartige Attribute in den zu den bereits vorhandenen Attributen hinzugefügt werden können, ohne dass für sie eine extra Tabelle erzeugt werden müsste, oder die bereits vorhandenen Tabellen erweitert werden müssten.

4.3.1.1 Single Generic Attribute Table

Single Generic Attribute Table ist folgendermaßen definiert:

- Element(ID, Source, LEID, Type)
- Link(ID, *SourceElementID*, *TargetElementID*, SourceSubElement, TargetSubElement, Generator, LinkTypeLabel)
- LinkTypeAttribute(*LinkID*, AttributeName, AttributeType, AttributeValue)

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel
1	2	3	*NULL*	*NULL*	manual	Created
2	3	1	*NULL*	*NULL*	manual	Explains

SGAT_LinkTypeAttribute

linkid	attributename	attributetype	attributevalue
1	GenerationDate	Date	1990-01-01
1	GenerationAuthor	String	Hood
1	ConfidenceLevel	Integer	99
1	CreationHours	Integer	75
2	GenerationDate	Date	1990-01-02
2	GenerationAuthor	String	Gröger
2	ConfidenceLevel	Integer	75
2	ExplainsType	String	Reason
2	ExplainsSubject	String	Quality Problem

Abbildung 12: Beispiel Single Generic AttributeTable

Abbildung 12 zeigt das Fallbeispiel für diese Lösungsvariante. Wie in [15] vorgestellt, werden die Elemente und Links in zwei verschiedenen Tabellen gespeichert. Die Attributstabelle wächst besonders schnell an, da sämtliche Attribute in einer einzelnen Tabelle gespeichert werden. Wird ein neuer Linktyp hinzugefügt oder ein bestehender Linktyp erweitert, kann dies komplett ohne eine Schemaänderung vollzogen werden. Dies gilt auch für den Fall, dass ein komplett neuer Attributsdatentyp verwendet wird, da der Datentyp explizit angegeben wird. Diese explizite Speicherung des Attributsdatentyps ist notwendig, da das Feld „AttributeValue“ nicht typisiert ist. Durch die explizite Speicherung des Attributsdatentyps erhöht sich allerdings der Speicherplatzverbrauch.

```
SELECT document.id
FROM element machine
  ,element document
  ,link explains
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND explains.id IN (
    SELECT linkID
    FROM sgat_linkTypeAttribute attributes
    WHERE attributes.linkID = explains.id
      AND (
        CASE
          WHEN attributes.attributename = 'ConfidenceLevel'
            THEN Cast(attributes.attributevalue AS INT) >= 50
          END
        )
      OR (
        attributes.attributeName = 'ExplainsType'
        AND attributes.attributeValue = 'Reason'
        )
    GROUP BY linkID
    HAVING count(*) = 2
    )
```

Listing 2: SQL-Anfrage für SGAT-Variante

Für die Anfrage ergibt sich die Lösung von Listing 2. Um die Anzahl der Joins zu minimieren, wurde hierbei eine Optimierung, entnommen aus [30], angewendet, welche ermöglicht, die Link-Attribute, welche die Lösung einschränken, mit nur einem Join auslesen zu können. Hierzu wird die Abfrage nach den Attributen in einer einzigen WHERE-Klausel vereinigt, welche nur dann zu TRUE auswertet, wenn sämtliche Attribute ihre Bedingungen erfüllen. Diese Optimierung funktioniert allerdings nur bei einer AND-Verknüpfung der Attribute, was allerdings dem Normalfall entsprechen sollte.

Untersuchungen, auch anhand anderer Abfragen, haben gezeigt, dass sich die Anzahl der benötigten Joins, in Abhängigkeit der in der Anfrage beteiligten Elemente und Links, in einer allgemeinen Form angeben lassen. Dies gilt allerdings nur unter der Einschränkung, dass die grundsätzliche Art der Anfrage nicht verändert wird. Jene grundsätzliche Art der Anfrage ist hier „Setze Elemente über die sie verbindenden Links in Beziehung. Auf den Elementen und den Links liegen dabei jeweils Einschrän-

kungen, welche für eine passende Lösung alle erfüllt sein müssen.“ Anders ausgedrückt muss, für die geschlossene Darstellung der Anzahl der Joins, die Art der Anfrage also eine navigationsorientierte Anfrage sein, bei welcher auf allen beiteiligten Elementen und Links Einschränkungen, welche alle erfüllt sein müssen, existieren.

Als Performance-Ergebnis erhält man damit:

- Casts: Benötigt
- Anzahl Joins: |elements| + (2*|links|)

|elements| steht hierbei für die Anzahl der beachteten elements für die Anfrage und gleiches gilt für |links|. 2*|links| ist dadurch bedingt, dass zu jedem Link noch seine zugehörigen Attribute hinzugejoint werden müssen, was die Anzahl der Joins pro Link also verdoppelt. Die Anzahl der benötigten Joins ist damit im unteren Bereich. Allerdings werden, bedingt durch die explizite Speicherung des Attributsdatentyps viele Casts benötigt, was die Performance negativ beeinträchtigt.

Die explizite Speicherung des Attributsdatentyps bringt eine schlechte Handhabbarkeit für das aufrufende Programm mit sich. Im Falle von dynamsichem SQL muss erst der Attributsdatentyp ausgelesen werden und gemäß dem Ergebnis dann der Cast definiert werden.

4.3.1.2 Generic Attribute Table per Data Type

Die Lösung Generic Attribute Table per Data Type stellt eine typsisierte Erweiterung von Single Generic Attribute Table dar. Sie unterscheidet sich von jener Lösung dahingehend, dass pro Attributstyp eine eigene Tabelle angelegt wird, wodurch eine Typisierung der Attribute erreicht wird:

- Element(ID, Source, LEID, Type)
- Link(ID, *SourceElementID*, *TargetElementID*, SourceSubElement, TargetSubElement, Generator, LinkTypeLabel)
- IntegerLinkTypeAttribute(*LinkID*, Attributename, AttributeValue)
- StringLinkTypeAttribute(*LinkID*, Attributename, AttributeValue)
- DateLinkTypeAttribute(*LinkID*, Attributename, AttributeValue)

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel
1	2	3	*NULL*	*NULL*	manual	Created
2	3	1	*NULL*	*NULL*	manual	Explains

GATD_IntegerLinkTypeAttribute

linkid	attributename	attributevalue
1	ConfidenceLevel	99
1	CreationHours	75
2	ConfidenceLevel	75

GATD_StringLinkTypeAttribute

linkid	attributename	attributevalue
1	GenerationAuthor	Hood
2	GenerationAuthor	Gröger
2	ExplainsType	Reason
2	ExplainsSubject	Quality Problem

GATD_DateLinkTypeAttribute

linkid	attributename	attributevalue
1	GenerationDate	1990-01-01
2	GenerationDate	1990-01-02

Abbildung 13: Beispiel Generic Attribute Table per Data Type

Abbildung 13 zeigt das Fallbeispiel für diese Lösungsvariante. Da im Rahmen dieser Diplomarbeit nur die Datentypen String, Integer und Date für die Attribute auftauchen, werden hier nur drei Attributstabellen benötigt. Bei mehr Datentypen müssten entsprechend mehr Attributstabellen erzeugt werden. Die Wartbarkeit ist hier etwas schlechter als bei „Single Generic Attribute Table“, da im Falle, dass ein komplett neuer Datentyp hinzukommt, eine neue Tabelle für diesen Datentyp erzeugt werden muss.

```
SELECT DISTINCT document.id
FROM element machine
  ,element document
  ,link explains
  ,gatd_IntegerLinkTypeAttribute integerAttributes
  ,gatd_stringlinktypeattribute stringAttributes
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND integerAttributes.linkID = explains.id
  AND stringAttributes.linkID = explains.id
  AND CASE
    WHEN integerAttributes.attributename = 'ConfidenceLevel'
      THEN integerAttributes.attributevalue >= 50
    END
  AND CASE
    WHEN stringAttributes.attributename = 'ExplainsType'
      THEN stringAttributes.attributevalue = 'Reason'
    END
```

Listing 3: SQL-Anfrage für GATD-Variante

Für die Anfrage ergibt sich die Lösung von Listing 3, was dann zu folgendem Performanceverhalten führt:

- Casts: Nicht benötigt
- Anzahl Joins: Es muss zwischen dem Worst Case und dem Best Case unterschieden werden, je nachdem wie die Datentypen über die Links verteilt sind.

  Im Worst Case hat jeder Link sämtliche Datentypen der LinkTypeAttribute.

  Im Best Case hat jeder Link genau einen Datentyp der LinkTypeAttribute. In diesem Fall ist der Trick mit der Minimierung der Joins vollständig möglich.

  Anzahl Joins worst case: |elements| + |links| + |links| * |data types|

  Anzahl Joins best case: |elements| + 2* |links|

Da im Gegensatz zu Single Generic Attribute Table der Datentyp hier nicht mehr explizit gespeichert wird, fällt der Nachteil des erhöhten Speicherplatzverbrauchs weg. Hinsichtlich der Performance tritt das Problem der benötigten Casts hier nicht mehr auf. Dafür ist die Anzahl der Joins hier die schlechteste aller Lösungen. Für den Nutzer, welcher nicht mit der Implementierung des Datenmodells vertraut ist, kann nicht ersichtlich sein, weshalb die Anzahl der Joins davon abhängt, welche Datentypen mit einem Link in Verbindung stehen.

4.3.1.3 Single Generic Attribute Table with Implicit Attributes

Wie auch für die Lösung Generic Attribute Table per Data Type gültig, ist es Zweck dieser Lösung eine Typisierung der Attribute zu erreichen. Im Gegensatz zu jener Lösung, werden hier allerdings nicht verschiedene Tabellen erzeugt, sondern die Typisierung findet innerhalb einer einzigen Tabelle statt:

- Element(ID, Source, LEID, Type)
- Link(ID, *SourceElementID*, *TargetElementID*, Generator, LinkTypeLabel)
- LinkTypeAttribute(*LinkID*, AttributeName, IntegerAttributeValue, StringAttributeValue, DateAttributeValue)

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel
1	2	3	*NULL*	*NULL*	manual	Created
2	3	1	*NULL*	*NULL*	manual	Explains

SGATIA_LinkTypeAttribute

linkid	attributename	integerattributevalue	stringattributevalue	dateattributevalue
1	GenerationDate	*NULL*	*NULL*	1990-01-01
1	GenerationAuthor	*NULL*	Hood	*NULL*
1	ConfidenceLevel	99	*NULL*	*NULL*
1	CreationHours	75	*NULL*	*NULL*
2	GenerationDate	*NULL*	*NULL*	1990-01-02
2	GenerationAuthor	*NULL*	Gröger	*NULL*
2	ConfidenceLevel	75	*NULL*	*NULL*
2	ExplainsType	*NULL*	Reason	*NULL*
2	ExplainsSubject	*NULL*	Quality Problem	*NULL*

Abbildung 14: Beispiel Single Generic Attribute Table with Implicit Attributes

Abbildung 15 zeigt das Fallbeispiel für diese Lösungsvariante. Wie ersichtlich, wird hier in den Feldern „IntegerAttributeValue“, „StringAttributeValue“ und „DateAttributeValue“ nur ein einziger Wert, entsprechend dem Datentyp des Attributs, eingetragen, während die übrigen Felder mit Null-Werten aufgefüllt werden. Damit existieren hinsichtlich des Speicherplatzverbrauchs zwei gegenteilige Effekte. Einerseits bewirkt die

nunmehr nicht mehr explizite Speicherung des Datentyps einen geringeren Speicherplatzverbrauch, aber andererseits bewirkt die erhöhte Anzahl an Null-Werten einen erhöhten Speicherplatzverbrauch.

```
SELECT DISTINCT document.id
FROM element machine
  ,element document
  ,link explains
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND explains.id IN (
    SELECT linkID
    FROM sgatia_linkTypeAttribute attributes
    WHERE attributes.linkID = explains.id
      AND (
        attributes.attributename = 'ConfidenceLevel'
        AND attributes.integerattributevalue >= 50
        )
      OR (
        attributes.attributename = 'ExplainsType'
        AND attributes.stringattributevalue = 'Reason'
        )
    GROUP BY linkID
    HAVING count(*) = 2
    )
```

Listing 4: SQL-Anfrage für SGATIA-Variante

Für die Anfrage ergibt sich die Lösung von Listing 4. Verallgemeinert lassen sich aus der Lösung dann folgende Performanceindikatoren ableiten:

- Casts: Nicht benötigt
- Anzahl Joins: |elements| + 2*|links|

Die Performance ist hier etwas besser als bei der expliziten Speicherung, da keine Casts mehr benötigt werden. Bezüglich der Joins weist diese Lösung dasselbe Performanceverhalten wie „Single Generic Attribute Table“, beschrieben in 4.3.1.1 auf.

4.3.1.4 Single Generic Attribute Table with Horicontal Attributes

Im Gegensatz zu den bisherigen Lösungen werden die Attribute hier nicht mehr vertikal, sondern horizontal aufgespannt. Enthält ein konkreter Link also mehrere Attribute,

werden jene nicht einzeln in der Attributstabelle gespeichert, sondern sie werden alle in ein einziges Tupel der Attributstabelle eingefügt:

- Element(ID, Source, LEID, Type)
- Link(ID, SourceElementID, TargetElementID, Generator, LinkTypeLabel)
- LinkTypeAttribute(*LinkID*, $AttributeName_1$, $AttributeType_1$, $AttributeValue_1$, $AttributeName_2$, $AttributeType_2$, $AttributeValue_2$,..., $AttributeName_n$, $AttributeType_n$, $AttributeValue_n$)

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel
1	2	3	*NULL*	*NULL*	manual	Created
2	3	1	*NULL*	*NULL*	manual	Explains

SGATHOA_LinkTypeAttribute

linkid	attributename1	attributetype1	attributevalue1	attributename2	attributetype2	attributevalue2	attributename3	...
1	GenerationDate	Date	1990-01-01	GenerationAuthor	String	Hood	ConfidenceLevel	
2	GenerationDate	Date	1990-01-02	GenerationAuthor	String	Gröger	ConfidenceLevel	

attributetype3	attributevalue3	attributename4	attributetype4	attributevalue4	attributename5	attributetype5	attributevalue5
Integer	99	CreationHours	Integer	75	*NULL*	*NULL*	*NULL*
Integer	75	ExplainsType	Integer	Reason	ExplainsSubject	String	Quality Problem

Abbildung 15: Beispiel Single Generic Attribute Table with Horizontal Attributes

Abbildung 15 zeigt das Fallbeispiel für diese Lösungsvariante. Wie ersichtlich, ist die Attributstabelle hier sehr breit. Sie muss breit genug sein, dass sämtliche Attribute des Linktyps mit den meisten Attributen in ihr gespeichert werden können. Enthält ein Linktyp weniger Attribute, werden die überflüssigen Spalten mit Null-Werten aufgefüllt.

Durch diese horizontale Aufspannung der Attribute sinkt die Wartbarkeit. Im Falle, dass eine obere Schranke für die Anzahl der Attribte angegeben werden kann, kann die LinkTypeAttribute-Tabelle breit genug gewählt werden, dass keine Schemaänderungen für das Einfügen neuer Attribute benötigt werden. Sollte diese obere Schranke jedoch überschritten werden, muss die Tabelle entsprechend verbreitert werden. Der Speicher-

platzverbrauch ist, bedingt durch die enorme Anzahl an Null-Werten, als nachteilig anzusehen.

```
SELECT DISTINCT document.id
FROM element machine
  ,element document
  ,link explains
  ,sgathoa_linkTypeAttribute attributes
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND attributes.linkID = explains.id
  AND CASE
    WHEN attributes.attributename1 = 'ConfidenceLevel'
      THEN Cast(attributes.attributevalue1 AS INT) >= 50
    WHEN attributes.attributename2 = 'ConfidenceLevel'
      THEN Cast(attributes.attributevalue2 AS INT) >= 50
    WHEN attributes.attributename3 = 'ConfidenceLevel'
      THEN Cast(attributes.attributevalue3 AS INT) >= 50
    WHEN attributes.attributename4 = 'Trust'
      THEN Cast(attributes.attributevalue4 AS INT) >= 50
    END
  AND CASE
    WHEN attributes.attributename1 = 'ExplainsType'
      THEN attributes.attributevalue1 = 'Reason'
    WHEN attributes.attributename2 = 'ExplainsType'
      THEN attributes.attributevalue2 = 'Reason'
    WHEN attributes.attributename3 = 'ExplainsType'
      THEN attributes.attributevalue3 = 'Reason'
    WHEN attributes.attributename4 = 'ExplainsType'
      THEN attributes.attributevalue4 = 'Reason'
    END
```

Listing 5: SQL-Anfrage für SGATHOA-Variante

Für die Anfrage ergibt sich die Lösung von Listing 5. Verallgemeinert lassen sich aus der Lösung dann folgende Performanceindikatoren ableiten:

- Casts: Benötigt
- Anzahl Joins: |elements| + 2 * |links|

Sinn der horizontalen Aufspannung der Attribute ist die verbesserte Performance dadurch, dass Joins eingespart werden. Allerdings wurde diese Anzahl der Joins durch Verwendung performanter SQL-Anfragen bereits bei vertikaler Aufspannung der Attribute, beispielsweise für Single Generic Attribute Table erreicht. Hier hat man diese

Anzahl der Joins allerdings garantiert, während die Performanceoptimierung bei den vertikal aufgespannten Attributen nur möglich ist, wenn die Attribute mit AND verknüpft werden. Weiterhin werden hier Casts benötigt. Die Handhabbarkeit ist ein großes Problem dieser Lösungsvariante. Über eine komplexe CASE-Anfrage muss zunächst einmal festgestellt werden, in welcher Spalte der gesuchte Eintrag vorhanden ist.

4.3.1.5 Single Generic Attribute Table with Horizontal Implicit Attributes

Diese Lösung erweitert die Lösung Single Generic Attribute Table With Horizontal Attributes um die Typisierung der Attribute:

- Element(ID, Source, LEID, Type)
- Link(ID, *SourceElementID*, *TargetElementID*, Generator, LinkTypeLabel)
- LinkTypeAttribute(*LinkID*, IntegerAttributeName$_1$, IntegerAttributeValue$_1$,..., IntegerAttributeName$_n$, IntegerAttributeValue$_n$, StringAttributeName$_1$, StringAttributeValue$_1$,..., StringAttributeName$_n$, StringAttributeValue$_n$, DateAttributeName$_1$, DateAttributeValue$_n$,..., DateAttributeName$_n$, DateAttributeValue$_n$)

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel
1	2	3	*NULL*	*NULL*	manual	Created
2	3	1	*NULL*	*NULL*	manual	Explains

SGATHIA_LinkTypeAttribute

linkid	integerattributename1	integerattributevalue1	integerattributename2	integerattributevalue2	stringattributename1	...
1	ConfidenceLevel	99	CreationHours	75	GenerationAuthor	
2	ConfidenceLevel	75	*NULL*	*NULL*	GenerationAuthor	

stringattributevalue1	stringattributename2	stringattributevalue2	stringattributename3	stringattributevalue3	...
Hood	*NULL*	*NULL*	*NULL*	*NULL*	
Gröger	ExplainsType	Reason	ExplainsSubject	Quality Problem	

dateattributename1	dateattributevalue1
GenerationDate	1990-01-01
GenerationDate	1990-01-02

Abbildung 16: Beispiel Single Generic Attribute Table with Horizontal Implicit Attributes

Abbildung 20 zeigt das Fallbeispiel für diese Lösungsvariante. Wie ersichtlich, gleicht sie der Lösung Single Generic Attribute Table With Horizontal Attributes bis auf die Einschränkung, dass die Attribute hier typisiert sind. Damit gelten die Bewertungen der impliziten Speicherung von 4.3.1.3 und der horizontalen Aufspannung der Attribute von 4.3.1.4 hier gleichermaßen.

```
SELECT DISTINCT document.id
FROM element machine
  ,element document
  ,link explains
  ,sgathia_linkTypeAttribute attributes
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND attributes.linkID = explains.id
  AND CASE
    WHEN attributes.integerattributename1 = 'ConfidenceLevel'
      THEN attributes.integerattributevalue1 >= 50
    WHEN attributes.integerattributename2 = 'ConfidenceLevel'
      THEN attributes.integerattributevalue2 >= 50
    END
  AND CASE
    WHEN attributes.stringattributename1 = 'ExplainsType'
      THEN attributes.stringattributevalue1 = 'Reason'
    WHEN attributes.stringattributename2 = 'ExplainsType'
      THEN attributes.stringattributevalue2 = 'Reason'
    END
```

Listing 6: SQL-Anfrage für SGATHIA-Variante

Für die Anfrage ergibt sich die Lösung von Listing 6. Verallgemeinert lassen sich aus der Lösung dann folgende Performanceindikatoren ableiten:

- Casts: Nicht benötigt
- Anzahl Joins: |element| + 2*|link|

4.3.1.6 Melted Links and Attributes

Während die anderen Lösungen Links von ihren Attributen trennen, werden sie in Melted Links and Attributes zusammengefügt. Dies impliziert, dass die Attribute horizontal aufgespannt werden:

- Element(ID, Source, LEID, Type)

- Link(ID, *SourceElementID*, *TargetElementID*, SourceSubElement, TargetSubElement, Generator, LinkTypeLabel, AttributeName$_1$, AttributeType$_1$, AttributeValue$_1$, AttributeName$_2$, AttributeType$_2$, AttributeValue$_2$,..., Attributename$_n$, AttributeType$_n$, AttributeValue$_n$)

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

MLAA_Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel	attributename1	...
1	2	3	NULL	NULL	manual	Created	GenerationDate	
2	3	1	NULL	NULL	manual	Explains	GenerationDate	

attributetype1	attributevalue1	attributename2	attributetype2	attributevalue2	attributename3	attributetype3	...
Date	1990-01-01	GenerationAuthor	String	Hood	ConfidenceLevel	Integer	
Date	1990-01-02	GenerationAuthor	String	Gröger	ConfidenceLevel	Integer	

attributevalue3	attributename4	attributetype4	attributevalue4	attributename5	attributetype5	attributevalue5
99	CreationHours	Integer	75	NULL	NULL	NULL
75	ExplainsType	String	Reason	ExplainsSubject	String	Quality Problem

Abbildung 17: Beispiel Melted Links and Attributes

Abbildung 18 zeigt das Fallbeispiel für diese Lösungsvariante. Wie ersichtlich, gibt es hier, neben der Element-Tabelle, nur noch die Tabelle MLAA_Link, in welcher die Links mit ihren Attributen vereinigt werden. Durch die horizontale Aufspannung der Attribute treten bezüglich der Wartbarkeit und des Speicherplatzverbrauchs dieselben Probleme wie in 4.3.1.4 beschrieben auf.

```
SELECT DISTINCT document.id
FROM element machine
  ,element document
  ,mlaa_link explains
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND CASE
    WHEN explains.attributename1 = 'ExplainsType'
      THEN explains.attributevalue1 = 'Reason'
    WHEN explains.attributename2 = 'ExplainsType'
      THEN explains.attributevalue2 = 'Reason'
    WHEN explains.attributename3 = 'ExplainsType'
      THEN explains.attributevalue3 = 'Reason'
    WHEN explains.attributename4 = 'ExplainsType'
      THEN explains.attributevalue4 = 'Reason'
    END
  AND CASE
    WHEN explains.attributename1 = 'ConfidenceLevel'
      THEN Cast(explains.attributevalue1 AS INT) >= 50
    WHEN explains.attributename2 = 'ConfidenceLevel'
      THEN Cast(explains.attributevalue2 AS INT) >= 50
    WHEN explains.attributename3 = 'ConfidenceLevel'
      THEN Cast(explains.attributevalue3 AS INT) >= 50
    WHEN explains.attributename4 = 'ConfidenceLevel'
      THEN Cast(explains.attributevalue4 AS INT) >= 50
    END
```

Listing 7: SQL-Anfrage für MLAA-Variante

Für die Anfrage ergibt sich die Lösung von Listing 7. Verallgemeinert lassen sich aus der Lösung dann folgende Performanceindikatoren ableiten:

- Casts: Keine Aussage
- Anzahl Joins: |element| + |link|

Die Casts werden für das Beispiel deshalb benötigt, da die Attribute explizit gespeichert werden. Allerdings erlaubt dieser Lösungstypus auch die Attribute implizit wie in 4.3.1.5 zu speichern. Bezüglich der Anzahl der Joins wird hier die beste Performance sämtlicher Lösungsvarianten erreicht. Dies ist der Fall, da die Attribute hier nicht mehr explizit zu ihrem Link hinzugejoint werden, da sie bereits im Tupel des Links enthalten sind.

4.3.1.7 Attributes in 3NF

Das Feld AttributeType in der Tabelle LinkTypeAttribute abzuspeichern, entspricht einer Verletzung der zweiten Normalform. Grund ist, dass AttributeType bereits von AttributeName funktional abhängt und damit also nicht vom gesamten Primärschlüssel. Diese Lösung überführt die Attribute in die dritte Normalform:

- Element(ID, Source, LEID, Type)
- Link(ID, *SourceElementID*, *TargetElementID*, SourceSubElement, TargetSubElement, Generator, LinkTypeLabel)
- LinkTypeAttribute(*LinkID*, AttributeName, AttributeValue)
- AttributeType(AttributeName, AttributeType)

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel
1	2	3	*NULL*	*NULL*	manual	Created
2	3	1	*NULL*	*NULL*	manual	Explains

LTA3NF_LinkTypeAttribute

linkid	attributename	attributevalue
1	GenerationDate	1990-01-01
1	GenerationAuthor	Hood
1	ConfidenceLevel	99
1	CreationHours	75
2	GenerationDate	1990-01-02
2	GenerationAuthor	Gröger
2	ConfidenceLevel	75
2	ExplainsType	Reason
2	ExplainsSubject	Quality Problem

LTA3NF_AttributeType

attributename	attributetype
GenerationDate	Date
GenerationAuthor	String
ConfidenceLevel	Integer
CreationHours	Integer
ExplainsType	String
ExplainsSubject	String

Abbildung 18: Beispiel Link Type Attributes in 3NF

Abbildung 18 zeigt das Fallbeispiel für diese Lösungsvariante. Problematisch ist, wenn zwei Attribute gleichnamig benannt sind, während der Datentyp verschieden ist. In diesem Fall kann die Normalisierung nicht vollzogen werden, da die Abhängigkeit von AttributeTyp von AttributeName nicht mehr gegeben ist. Auch wenn der Datentyp

identisch ist, ergibt sich das Problem, dass, wenn an einem Attribut-Namen eines Linktyp-Attributs etwas geändert werden soll, unerwünschte Seiteneffekte auf andere Linktyp-Attribute auftreten können, wenn nicht zuvor geprüft wird, ob ein Attribut-Name in mehreren Linktyp-Attributen vorkommt. In der Summe wird die Wartbarkeit bzw. sogar die semantische Mächtigkeit durch die Ausgliederung in eine spezielle Tabelle für den Attributstyp eingeschränkt. Dafür bringt die Verwendung der dritten Normalform einen geringeren Speicherplatzverbrauch, da der Attribut-Typ nur einmal pro Attribut-Name gespeichert werden muss, anstelle pro Linktyp-Attribut.

```
SELECT DISTINCT document.id
FROM element machine
  ,element document
  ,link explains
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND explains.id IN (
    SELECT linkID
    FROM lta3nf_linkTypeAttribute attributes
    WHERE attributes.linkID = explains.id
      AND (
        CASE
          WHEN attributes.attributename = 'ConfidenceLevel'
            THEN Cast(attributes.attributevalue AS INT) >= 50
          END
        )
      OR (
        attributes.attributename = 'ExplainsType'
        AND attributes.attributevalue = 'Reason'
        )
    GROUP BY linkID
    HAVING count(*) = 2
    )
```

Listing 8: SQL-Anfrage für LTA3NF-Variante

Für die Anfrage ergibt sich die Lösung von Listing 8. Verallgemeinert lassen sich aus der Lösung dann folgende Performanceindikatoren ableiten:

- Casts: Keine Aussage
- Anzahl Joins: |element| + 2*|link|

Wie ersichtlich gleicht die Anfrage hier der Anfrage für Single Generic Attribute Table in 4.3.1.1, was auch ein identisches Performanceverhalten mit sich bringt. Der intuitive Ersteindruck, dass ein weiterer Join für den Datentyp benötigt würde, täuscht, da der zugehörige Datentyp bereits als implizites Wissen in der Anfrage vorhanden sein muss, um eine sinnvolle Anfrage zu ermöglichen. Im Falle von dynamischem SQL muss das Anwendungsprogramm den Datentyp im Vorhinein erfragen. Da es bei Verwendung dieser Lösung dazu auf die kleinere Tabelle LTA3NF_AttributeTyp zugreifen kann, kann der Zugriff hier sogar minimal schneller erfolgen.

4.3.2 Lösungskategorie konkrete Attributstabelle

Im Gegensatz zur Lösungskategorie der Generischen Attributstabelle werden die Attribute in dieser Lösungskategorie auf die klassische Art und Weise abgespeichert, was also bedeutet, dass jedes Attribut in einer extra angelegten Spalte abgespeichert wird. Durch diese „konkrete“ Speicherung der Attribute wird ermöglicht, dass die Anfragen vereinfacht werden, da der Spaltenname für den Zugriff herangezogen werden kann. Weiterhin wird eine Performanceverbesserung durch die Abwesenheit der hier nicht mehr benötigten Casts angestrebt. Der Nachteil der konkreten Varianten liegt darin begründet, dass das Hinzufügen neuer Links bzw. Änderungen an bestehenden Links stets zu einer Änderung des relationalen Modells führen wird.

4.3.2.1 Multiple Concrete Attribute Tables

Im Ansatz Multiple Concrete Attribute Tables wird pro Linktyp eine separate Attributstabelle erzeugt, in welcher nur jene Attribute eingetragen werden, wo der zugehörige Link zum Linktyp gehört:

- Element(<u>ID</u>, Source, LEID, Type)
- Link(<u>ID</u>, *SourceElementID*, *TargetElementID*, Generator, LinkTypeLabel)
- LinkTypeAttribute$_1$(<u>*LinkID*</u>, ConcreteAttribute$_{A1}$, ConcreteAttribute$_{A2}$,..., ConcreteAttribute$_{An}$)
- LinkTypeAttribute$_2$(<u>*LinkID*</u>, ConcreteAttribute$_{B1}$, ConcreteAttribute$_{B2}$,..., ConcreteAttribute$_{Bn}$)
- ...

In der Auflistung stehen Einträge wie $\text{LinkTypeAttribute}_{1}$ für einen konkreten Linktyp. Selbiges gilt für die Einträge der Art $\text{ConcreteAttribute}_{A1}$, wo ein konkreter Attributs-Typ den Spaltennamen stellt.

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel
1	2	3	*NULL*	*NULL*	manual	Created
2	3	1	*NULL*	*NULL*	manual	Explains

MCAT_Created

linkid	generationdate	generationauthor	confidencelevel	creationhours
1	1990-01-01	Hood	99	75

MCAT_Explains

linkid	generationdate	generationauthor	confidencelevel	explainstype	explainssubject
2	1990-01-02	Gröger	75	Reason	Quality Problem

Abbildung 19: Beispiel Multiple Concrete Attribute Tables

Abbildung 19 zeigt das Fallbeispiel für diese Lösungsvariante. Wie ersichtlich wurden im Beispiel Bezeichnungen des Typs $\text{LinkTypeAttribute}_{i}$ durch den konkreten Namen der Attributstabelle substituiert. Die Anzahl der Attributstabellen entspricht der Anzahl der Linktypen und die konkrete Struktur einer Attributstabelle ergibt sich aus den in ihr gespeicherten Attributen.

```
SELECT DISTINCT document.id
FROM element machine
   ,element document
   ,link explains
   ,mcat_explains attributes
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND attributes.linkID = explains.id
  AND attributes.confidenceLevel >= 50
  AND attributes.explainstype = 'Reason'
```

Listing 9: SQL-Anfrage für MCAT-Variante

Für die Anfrage ergibt sich die Lösung von Listing 9. Verallgemeinert lassen sich aus der Lösung dann folgende Performanceindikatoren ableiten:

- Casts: Nicht benötigt
- Anzahl Joins: |elements| + 2*|links|

Die Anzahl der Joins bewegt sich damit im unteren Bereich. Zwar wurde diese Anzahl auch beispielsweise für „Single Generic Attribute Table“ erreicht, allerdings galt dort die Einschränkung, dass die Attribute alle mit AND verknüpft werden. Weiterhin ist die Anfrage hier einfacher als bei den generischen Varianten, da der Spaltenname verwendet werden kann. In diesem Fall kann beispielsweise mit „Attributes.ConfidenceLevel >= 50“ abgefragt werden, während die Anfrage für Single Generic Attribute Table in 4.3.1.1 mit „CASE WHEN Attributes.AttributeName = 'ConfidenceLevel' THEN...“ erstmals, anhand des Attribut-Namens, feststellen musste, ob das richtige Tupel vorliegt.

4.3.2.2 Single Concrete Attribute Table

Im Gegensatz zu Multiple Concrete Attribute Tables wird hier nicht pro Linktyp eine neue Tabelle angelegt, sondern sämtliche Attribute werden in eine einzige Attributs-Tabelle eingetragen:

- Element(ID, Source, LEID, Type)
- Link(ID, *SourceElementID*, *TargetElementID*, Generator, LinkTypeLabel)

- LinkTypeAttributes(*LinkID*, $ConcreteAttribute_{A1}$, $ConcreteAttribute_{A2}$,..., $ConcreteAttribute_{An}$, $ConcreteAttribute_{B1}$, $ConcreteAttribute_{B2}$,..., $ConcreteAttribute_{Bn}$)

Element

id	source	leid	type
1	DW	pir.machine.ID=100	machine
2	DW	pir.employee.ID=20	employee
3	CMS	SpaceStore/12345	document

Link

id	sourceelementid	targetelementid	sourcesubelement	targetsubelement	generator	linktypelabel
1	2	3	*NULL*	*NULL*	manual	Created
2	3	1	*NULL*	*NULL*	manual	Explains

SCAT_LinkTypeAttribute

linkid	generationdate	generationauthor	confidencelevel	creationhours	explainstype	explainssubject
1	1990-01-01	Hood	99	75	*NULL*	*NULL*
2	1990-01-02	Gröger	75	*NULL*	Reason	Quality Problem

Abbildung 20: Beispiel Single Concrete Attribute Table

Abbildung 20 zeigt das Fallbeispiel für diese Lösungsvariante. Wie ersichtlich, wächst die Attributstabelle mit dem Hinzukommen weiterer Linktypen horizontal in die Breite. Wenn ein Attribut nicht zu einem Linktyp gehört, wird der entsprechende Eintrag mit einem Null-Wert aufgefüllt. Es muss im Falle eines komplett neuen Linktyps keine komplett neue Tabelle erezgut werden, aber die alte Tabelle muss entsprechend erweitert werden. Problematisch ist, wenn zwei verschiedene Linktypen ein gleichnamig benanntes Linktyp-Attribut haben. In diesem Fall darf es in der Tabelle LinkTypeAttributes nur ein einziges mal gespeichert werden. Wenn nun ein Linktyp gelöscht werden soll, kann bei Multiple Concrete Attribute Tables einfach die entsprechende Linktyp-Tabelle gelöscht werden. Mit dieser Lösung hingegen müssen die zugehörigen Spalten in LinkTypeAttributes gelöscht werden. Dabei ist jedoch darauf zu achten, ob eventuell ein anderer Linktyp ein gleichnamig benanntes Linktyp-Attribut besitzt, in wessen Fall die Spalte nicht gelöscht werden darf. Aus diesem Grund ergibt sich, dass die Wartbarkeit schlechter als bei Multiple Concrete Attribute Tables ist. Weiterhin erhöht sich hier durch die Verwendung vieler Null-Werte der Speicherplatzverbrauch.

```
SELECT DISTINCT document.id
FROM element machine
   ,element document
   ,link explains
   ,scat_linkTypeAttribute attributes
WHERE machine.type = 'machine'
  AND machine.id = 1
  AND document.type = 'document'
  AND explains.sourceElementID = document.id
  AND explains.targetElementID = machine.id
  AND explains.linktypelabel = 'Explains'
  AND attributes.linkID = explains.id
  AND attributes.confidenceLevel >= 50
  AND attributes.explainstype = 'Reason'
```

Listing 10: SQL-Anfrage für SCAT-Variante

Für die Anfrage ergibt sich die Lösung von Listing 10. Verallgemeinert lassen sich aus der Lösung dann folgende Performanceindikatoren ableiten:

- Casts: Nicht benötigt
- Anzahl Joins: |element| + 2*|link|

Die SQL-Anfrage gleicht strukturell der SQL-Anfrage von Multiple Concrete Attribute Tables, womit auch das Performanceverhalten gleich wie in 4.3.2.1 ist.

4.3.3 Elements and Links Melted

Da die Entscheidung Knoten und Kanten zu verschmelzen unabhängig von den Attributen ist, gehört Melted Elements and Links weder zur Kategorie Generische Attributstabelle noch zur Kategorie Konkrete Attributstabelle. Wie die Elemente und Links verschmolzen werden, wird im Folgenden aufgeführt:

- LinkWithElementAnnotation(ID, SourceElementID, SourceElementSource, SourceElementLEID, SourceElementType, TargetElementID, TargetElementSource, TargetElementLEID, TargetElementType, SourceSubElement, TargetSubElement, Generator, LinkTypeLabel)
- LinkTypeAttribute(*LinkID*, AttributeName, AttributeType, AttributeValue)

MEA_LinkWithElementAnnotation

id	sourceelementid	sourceelementsource	sourceelementleid	sourceelementtype	targetelementid	targetelementsource	...
1	2	DW	pir.employee.ID=20	employee	3	CMS	
2	3	CMS	SpaceStore/12345	document	1	DW	

targetelementleid	targetelementtype	sourcesubelement	targetsubelement	generator	linktypelabel
SpaceStore/12345	document	*NULL*	*NULL*	manual	Created
pir.machine.ID=100	machine	*NULL*	*NULL*	manual	Explains

LinkTypeAttribute

linkid	attributename	attributetype	attributevalue
1	GenerationDate	Date	1990-01-01
1	GenerationAuthor	String	Hood
1	ConfidenceLevel	Integer	99
1	CreationHours	Integer	75
2	GenerationDate	Date	1990-01-02
2	GenerationAuthor	String	Gröger
2	ConfidenceLevel	Integer	75
2	ExplainsType	String	Reason
2	ExplainsSubject	String	Quality Problem

Abbildung 21: Beispiel Melted Elements and Links

Abbildung 2 zeigt das Fallbeispiel für diese Lösungsvariante. Wie ersichtlich findet bezüglich der Elemente eine Redundanz statt, da jeder Link die beiden mit ihm verbundenen Elemente separat speichert. Weiterhin können Elemente nicht mehr ohne Links existieren. Kombiniert man diese Lösung mit Melted Links and Attributes, kommt man mit nur einer einzigen Tabelle für die gesamte Linkstruktur aus. Hinsichtlich der Linktyp-Atrribute ist die Wartbarkeit nicht betroffen, da die Verschmelzung der Elemente und Links mit Lösungen, welche eine gute Wartbarkeit bezwecken, kombinierbar ist. Trotzdem ist die Wartbarkeit als negativ anzumerken, da Elemente hier nicht mehr ohne Links existieren können. Soll also ein Link gelöscht werden, kann dadurch mehr Information als erwünscht verloren gehen. Weiterhin führt die Redundanz, ein Element in jedem damit verbundenen Link zu speichern, zu einer Erhöhung des Speicherplatzverbrauchs.

```
SELECT DISTINCT explains.sourceElementID
FROM mea_linkWithElementAnnotation explains
WHERE explains.sourceElementType = 'document'
  AND explains.targetElementType = 'machine'
  AND explains.targetElementID = 1
  AND explains.id IN (
    SELECT linkID
    FROM sgat_linkTypeAttribute attributes
    WHERE attributes.linkID = explains.id
      AND (
        CASE
          WHEN attributes.attributename = 'ConfidenceLevel'
            THEN Cast(attributes.attributevalue AS INT) >= 50
          END
        )
      OR (
        attributes.attributename = 'ExplainsType'
        AND attributes.attributevalue = 'Reason'
        )
    GROUP BY linkID
    HAVING count(*) = 2
    )
```

Listing 11: SQL-Anfrage für MEA-Variante

Für die Anfrage ergibt sich die Lösung von Listing 11. Verallgemeinert lassen sich aus der Lösung dann folgende Performanceindikatoren ableiten:

- Casts: Keine Aussage
- Anzahl Joins: 2* |link|

Neben der Lösung Melted Links and Attributes wird hier damit also die geringste Anzahl an Joins benötigt. Ob Casts wirklich benötigt werden, hängt von der Kombination ab, mit der diese Variante verknüpft wird.

4.4 Abschließende Bewertung und Auswahl weiter zu verfolgender Lösungsvarianten

Tabelle 1 gibt eine Übersicht über die Vorteile und Nachteile der einzelnen Lösungsvarianten im relativen Vergleich zu den anderen Lösungsvarianten. Um eine Scheinobjektivität zu vermeiden, werden dabei lediglich die Bewertungen „+“ und „-“ vergeben. Dabei symbolisiert „+“ eine gute Bewertung bzw. eine Unbedenklichkeit, während „-“ auf Nachteile oder potentielle Nachteile der Lösung hinweist.

	Wartbarkeit	Performance	Handhabbarkeit	Speicherplatzverbrauch
Single Generic Attribute Table	+	-	-	-
Generic Attribute Table per Data Type	+	-	-	+
Explicit Attribute Data Type	+	-	-	?
Attributes horicontal	-	+	-	-
Melted Links and Attributes	-	+	-	-
Attributes in 3NF	-	+	-	+
Multiple Concrete Attribute Tables	-	+	+	+
Single Concrete Attribute Table	-	+	+	-
Elements and Links Melted	-	+	-	-

Tabelle 1: Bewertung der relationalen Lösungsvarianten im Überblick

Wie gezeigt, bringen nahezu sämtliche Lösungen ihre individuellen Vorteile mit sich. Als komplett zwecklos hat sich lediglich die Lösung Single Concrete Attribute Table präsentiert, da die Lösung Multiple Concrete Attribute Tables ihr nirgends unterlegen, aber dafür in einigen Punkten überlegen ist. Weiterhin soll der Ansatz, die Linktyp-Attribute horizontal aufzuspannen, nicht weiter verfolgt werden. Dieser Ansatz wurde entworfen, um die Anzahl der Joins zu minimieren. Da man, mit performanten SQL-Anfragen, bei einer AND-Verknüpfung der Attribute, gleichartige Ergebnisse bei Verwendung vertikal aufgespannter Attribute erhält, fällt dieser Vorteil weg. Der Ansatz, Knoten und Kanten zusammenzustecken, erniedrigt zwar die Anzahl der Joins, macht die Tabellen aber insgesamt zu schwer wartbar, um praktikabel zu sein und soll damit auch wegfallen. Außerdem soll der Ansatz, die Attribute in dritter Normalform zu speichern, aufgrund der erwähnten semantischen Probleme, nicht weiter verfolgt werden.

Übrig bleiben damit folgende vier Lösungen, welche genauer untersucht und implementiert werden sollen:

- Multiple Concrete Attribute Tables (MCAT)
- Single Generic Attribute Table (SGAT)
- Generic Attribute Table per Data Type (GATD)
- Single Generic Attribute Table with Implicit Attribute Data Types (SGATIA)

Sämtliche dieser Lösungen haben jeweils ihre eigenen Vorteile und Nachteile. MCAT ist die naive Standard-Lösung, welche hinsichtlich der Performance vermutlich gut abschneiden sollte und zu einfachen SQL-Anfragen führt, dafür aber schlecht wartbar ist. Die generischen Attributstabellen sind dahingehend auf Wartbarkeit ausgerichtet, während die Performance ein potentielles Problem darstellt. Inwiefern die benötigten Casts die Performance beeinträchtigen, kann durch Vergleich der Lösung SGAT mit den beiden anderen generischen Attributstabellen-Lösungen herausgefunden werden. Die Lösung GATD hat hinsichtlich der Joins das schlechteste und unvorhersehbarste Verhalten, aber dafür keine Null-Werte im Gegensatz zu SGATIA.

4.5 Spezifikation der ausgewählten Lösungsvarianten auf physikalischer Ebene

In diesem Kapitel werden die vier ausgewählten relationalen Lösungsvarianten hinreichend genau spezifiziert, um eine Implementierung zu ermöglichen. In Abschnitt 4.5.1 werden die genauen Datentypen festgelegt. Eine Beschreibung der Indexe erfolgt in Abschnitt 4.5.2.

4.5.1 Datentypen

Die auf logischer Ebene ausgewählten Datentypen können teilweise auf verschiedene physikalische Datentypen abgebildet werden [32]. Auf logischer Ebene gibt es im Prototyp die Datentypen String, Date, Integer und Boolean.

Für Ganzzahlen und Date ergibt sich der physikalische Datentyp direkt aus der Modellierung, also INTEGER und DATE. Da DB2 keine Boolean-Datentypen kennt, wird hierbei auf SMALLINT ausgewichen, wobei 0 False und 1 True repräsentiert. Bei String besteht Spielraum. Als ernsthafte Alternativen kommen Varchar und Char in Frage. Da die Char-Verarbeitung eine potentielle Fehlerquelle für den Java-Code darstellt und eventuell kleine Performanceeinbußen in SQL zu teuren Stringoperationen in Java führen würden, wurde Varchar genommen.

Einige der Fallbeispiele enthalten Spalten variablen Datentyps. Dies wirft zwei Fragen auf, nämlich:

- Welcher SQL bzw. DB2-Datentyp eignet sich für die unbestimmten Typen und

- wie können die unbestimmten Typen in ihr korrektes Format übertragen werden?

Eine praktikable Lösung ist, die unbestimmten Typen als Varchar abzuspeichern. DB2 bietet zwei Möglichkeiten Varchars nach Integer zu casten, nämlich die Funktionen INTEGER und CAST. Da CAST eine Funktion des SQL-Standards ist, empfiehlt die DB2-Spezifikation aus Portabilitätsgründen diese Funktion [33].

4.5.2 Indexstrukturen

Im Folgenden soll eine Indizierung für die vier ausgewählten Lösungskombinationen entworfen werden. Hilfestellungen für den Entwurf einer sinnvollen Indizierung boten Quellen, wie insbesondere [34], und der DB2 Query Optimizer, welcher sehr konkret und halbwegs zuverlässig angab, welche Indexe er für konkrete Anfragen vermisste. Einige der im Folgenden aufgelisteten Indexe werden nicht explizit erzeugt, sondern durch den Primärschlüssel oder durch UNIQUE-Constraints automatisch erstellt.

Die vier Lösungsvarianten unterscheiden sich lediglich hinsichtlich der Linktyp-Attribute. Damit ist es naheliegend, für Element und Link stets dieselben Indexe zu verwenden, welche da wären:

- Element: (ID), (LEID)
- Link: (ID), (SourceElementID, TargetElementID, LinkTypeLabel), (TargetElementID, SourceElementID, LinkTypeLabel), (TargetElementID, LinkTypeLabel, SourceElementID), (SourceElementID, LinkTypeLabel, TargetElementID)

Die Indizierung für Element ist simpel. Über LEID wird der Sprung zum Data Warehouse vorgenommen. Für die anderen Element-Felder kommen Indexe nicht in Frage, da deren Kardinalität dazu zu beschränkt ist.

Die Indizierung für Link hingegen ist sehr komplex und wurde nach einem intensivem Studium der Ausführungspläne vorgenommen. Die allgemeine Semantik eines Links ist, ein Startelement mit einem Zielelement zu verbinden. Hinsichtlich des Ausführungsplans stellt sich nun die Frage, in welcher Reihenfolge die drei beteiligten Komponenten eines Links berechnet werden. Hierfür existieren folgende praktisch mögliche Kombinationen:

1. Zuerst Link, dann Startelement und Zielelement (beliebige Reihenfolge)
2. Zuerst Startelement und Zielelement (beliebige Reihenfolge) und dann Link
3. Erst Startelement, dann Link, dann Zielement
4. Erst Zielelement, dann Link, dann Startelement

Die Indizierung muss nun in einer Art und Weise erstellt werden, dass sämtliche dieser vier Kombinationen unterstützt werden. Die erste Variante wird gewählt, wenn Startelement und Zielelement kaum eingeschränkt sind und die Einschränkungen auf dem Link liegen. Hinsichtlich der Indizierung lässt sich diese Variante in der Link-Tabelle nicht unterstützen, da die zugehörige Attributstabelle zum Link die Einschränkungen vorgeben wird. Ein Index auf LinkTypeLabel macht wegen der geringen Kardinalität keinen Sinn. Die zweite Variante wird beispielsweise durch den Index (SourceElementID, TargetElementID, LinkTypeLabel) unterstützt. Etwas komplexer sind die verbleibenden symmetrischen Varianten drei und vier. Hier wird der Link sozusagen als Sprungbrett genommen, um das Startelement bzw. Zielelement zu berechnen. Ob der Ausführungsplan zuerst das Startelement oder zuerst das Zielelement berechnet, kann enorme Auswirkungen auf die Laufzeit haben, je nachdem wo mehr Einschränkungen gemacht werden. Indexe wie (SourceElementID, LinkTypeLabel, TargetElementID) unterstützen die Variante drei und wenn bezüglich dieses Indexes SourceElementID und TargetElementID vertauscht werden, wird die Variante vier unterstützt.

Im Folgenden werden die für die relationalen Lösungsvarianten spezifischen Indexe auf die jeweiligen Attributstabellen vorgestellt:
Indizierung für MCAT:

- (LinkID, AttributeName)

Indizierung für SGAT:

- (LinkID, AttributeName), (AttributeName, AttributeValue)

Indizierung für SGATIA:

- (LinkID, AttributeName), (AttributeName, IntegerAttributeValue) und dasselbe dann auch für die anderen Datentypen

Indizierung für GATD:

- (LinkID, AttributeName), (AttributeName, AttributeValue). Für alle Datentyp-Tabellen identisch

Die Indizierungen der Art (AttributeName, AttributeValue) sollten zu keinen großen Laufzeitgewinnen führen. Grund ist, dass hierfür die Kardinalität von AttributeName zu eingeschränkt ist. Da auf dem Attributwert oft Bereichsanfragen durchgeführt werden, wäre eine Indizierung der Art (AttributeValue, AttributeName) nicht sinnvoll.

5 Implementierung und Evaluation

In diesem Kapitel werden die graphbasierte Variante und die ausgewählten relationalen Varianten implementiert und evaluiert. In den Abschnitten 5.1 und 5.2 wird der dazu entwickelte Prototyp vorgestellt, im Abschnitt 5.3 wird die entworfene Linkstruktur und das Mengengerüst vorgestellt und in Abschnitt 5.4 wird die Performance der Lösungsvarianten, anhand von definierten Use Cases, evaluiert.

5.1 Architektonischer Aufbau des Prototyps

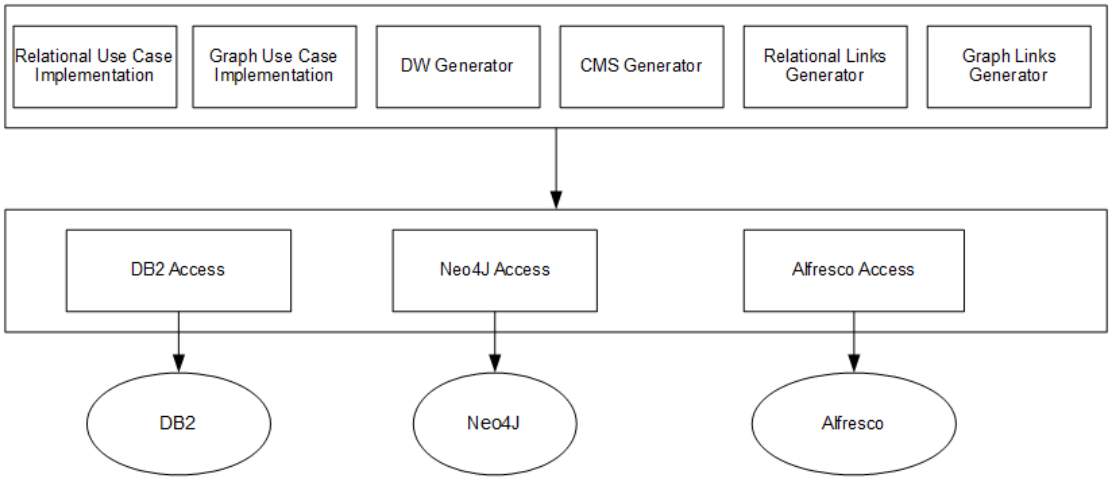

Abbildung 22: Architektur des Prototyps

Abbildung 22 skizziert die Architektur des Prototyps. Da der Prototyp nicht in ein bereits vorhandenes System eingebettet wird, sondern für sich alleine steht, müssen sämtliche benötigten äußeren Abhängigkeiten selbst erzeugt werden. Vorbedingung für die weiteren Schritte ist, dass das DW und das CMS mit Daten befüllt sind. Hierfür sind die Komponenten „DW Generator“ bzw. „CMS Generator“ zuständig. Sind die Quelldaten im DW und CMS vorhanden, kann die Komponente „Relational Links Generator“ die Links für die relationale Umsetzung und die Komponente „Graph Links Generator“ die Links für die graphbasierte Umsetzung erzeugen. Nach der Erzeugung der Links sind sämtliche benötigten Daten vorhanden, um Anfragen starten zu können. Die Implementierung dieser Anfragen findet sich in den Komponenten „Relational Use Case Implementation“ bzw. „Graph Use Case Implementation“.

Die Architektur gliedert sich in zwei Schichten. In der oberen Schicht ist die Anwendungslogik implementiert, während in der unteren Schicht der Zugriff auf die Fremdsysteme gekapselt ist. Die ovalen Kästchen in der Abbildung stellen dabei die drei Fremdsysteme dar und befinden sich außerhalb des Systems. In der oberen Schicht wurden die Komponenten so gewählt, dass aus Sicht der Anwendungslogik zusammenhängender Code gekapselt wurde, was auch als Strukturierung nach Feature bezeichnet wird [35]. Die Pfeile stellen die einzig erlaubten Zugriffe dar. Es ist also lediglich ein Zugriff von Komponenten der oberen Schicht auf Komponenten der unteren Schicht erlaubt, während Komponenten der oberen Schicht sich nicht austauschen dürfen. Diese strenge Kapselung der Komponenten zusammen mit der Strukturierung nach Feature bewirkt, dass einzelne Komponenten einfach herausgenommen werden können. Sollte beispielsweise die graphbasierte Link Store-Variante später verworfen werden, genügt es einfach, die entsprechenden Ordner zu löschen, ohne dass Seiteneffekte auf andere Programmteile bestehen würden.

5.1.1 Überblick über die Module zur Erzeugung der Graphtestdaten und der relationalen Testdaten

Hinsichtlich dem Relational Links Generator-Modul und dem Graph Links Generator-Modul bestand die Anforderung, dass die Graph-Datenbank und die relationale Datenbank mit identischen Testdaten befüllt werden. Aus designtechnischen Günden ist allerdings eine Abhängigkeit zwischen diesen beiden Modulen unerwünscht. Sollte beispielsweise später nur eine der beiden Lösungen weiter verfolgt werden, ist es nicht nachvollziehbar, wenn hinsichtlich des anderen Datengenerators Abhängigkeiten bestehen.

Als Lösung für das Problem, dass die Testdaten gleich sein sollen, ohne dass direkte Abhängigkeiten zwischen den Generatoren bestehen sollen, kamen deterministische Zufallszahlen für die randomisierte Erzeugung der Links zum Einsatz [1]. Der deterministische Zufallszahl-Generator von Java läuft so ab, dass manuell ein Startwert (Samen) gesetzt wird und aus diesem Samen dann eine unendlich lange Folge von zufällig erscheinenden Zahlen generiert wird [36]. Setzt man die Folge zurück auf die Ausgangsposition, wird wieder dieselbe Folge wie zuvor generiert.

[1] http://www.random.org/randomness/

Insgesamt lassen sich für den Prototyp folgende Vorteile deterministischer Zufallszahlen identifizieren:

- Gleiche Daten für den Graph-Link Store und relationalen Link Store.
  - Fehlerhafte Anfragen können anhand unterschiedlicher Ergebnisse identifiziert werden.
  - Aussagekraft der gemessenen Zeiten steigt durch identische Daten
- Die Use Case-Funktionsparameter können (für ein bestimmtes Volumen) nur einmal gesetzt werden und müssen nicht angepasst werden, wenn neue Testdaten generiert werden.
- Spätere Testdurchläufe können mit denselben Daten durchgeführt werden, auch wenn die gespeicherten Testdaten nicht mehr vorhanden sind. Tunt man nun die Anwendung, ist sichergestellt, dass die gemessenen Performanceverbesserungen nicht durch andersartige Testdaten determiniert sind.

Verkompliziert wurde das gleiche Erzeugen der Testdaten dadurch, dass der relationale Testdatengenerator und der Graphtestdatengenerator unterschiedliche Abläufe haben. Verantwortlich dafür sind die Link-Attribute, welche beim Graphtestdatengenerator zusammen mit ihrem Link erzeugt werden müssen. Bei den untersuchten relationalen Lösungen werden die Links und die Link-Attribute allerdings in unterschiedlichen Tabellen gespeichert, was als natürliche Lösung eine sequentielle Erzeugung, zuerst der Links und dann der Link-Attribute, mit sich bringt.

Der genaue Ablaufplan der Graph-Erzeugung ist folgender:

1. Erzeuge die Elemente
2. Erzeuge die Links zusammen mit ihren Link-Attributen

Wohingegen der Ablaufplan der relationalen Erzeugung folgender ist:

1. Erzeuge die Elemente
2. Erzeuge die Links
3. Erzeuge die Link-Attribute

Als Lösung kommen unterschiedliche Instanzen des Zufallszahlgenerators für die Link-Erzeugung und die Link-Attribut-Erzeugung zum Einsatz. Dadurch, dass für die Links und Link-Attribute unterschiedliche Zufallszahlfolgen generiert werden, wird erreicht, dass es keine Rolle mehr spielt, ob die Link-Attribute während oder nach den Links erzeugt werden.

Zu beachten ist allerdings, dass die interne Reihenfolge, in welcher die Links bzw. Link-Attribute erzeugt werden, für den Graph-Testdatengenerator und den relationalen Testdatengenerator identisch sein muss. Tauscht man beispielsweise die Reihenfolge um, sodass nunmehr zuerst die OccursIn-Links und dann die IsExpertOn-Links erzeugt werden, wird dies zur einer komplett anderen verbleibenden Linkstruktur führen.

5.1.2 Überblick über die Module zur Implementierung der Use Cases

Die Module, in welchen die Use Cases implementiert wurden, weisen pro Use Case eine passende Schnittstelle auf, welche von außen mit den benötigten Parametern aufgerufen werden kann. Da sowohl das Data Warehouse als auch der Link Store in der relationalen Implementierung in einer DB2-Datenbank laufen, wurde die relationale Implementierung auf zwei verschiedene Arten umgesetzt. Jene wären einmal mit separaten Anfragen für Data Warehouse und Link Store und einmal mit verschmolzenen Anfragen für Data Warehouse und Link Store. Die verschmolzenen Anfragen funktionieren allerdings nur, wenn das Data Warehouse und der Link Store sich in derselben Datenbank befinden, was eine Einschränkung an die Umgebung darstellt. Da die relationale Implementierung jeweils für die vier ausgewählten Schemavarianten umgesetzt wurde, ergibt dies also insgesamt bis zu neun Kombinationen pro Use Case.

Der strukturelle Aufbau des Codes ist für die vier separaten Varianten und die graphbasierte Variante identisch. Der einzige Unterschied liegt in unterschiedlichen SQL-Anfragen bzw. der Cypher-Anfrage. Von daher wird hier, was die relationalen Varianten betrifft, keine Codeduplizierung vorgenommen, sondern es wird nur die SQL-Anfrage ausgetauscht. Damit keine Abhängigkeit zwischen der graphbasierten Variante und den relationalen Varianten besteht, wird der Code für die graphbasierte Variante dupliziert. Die Varianten mit den verschmolzenen Anfragen folgen auch grundsätzlich

dem Ablauf der separaten Varianten, nur dass mehrere Schritte zusammengefasst werden. Notwendige Bedingung für die Korrektheit ist, dass sämtliche Lösungsvarianten, für dieselben Testdaten, identische Ergebnisse liefern.

5.2 Setup und verwendete Technologien für den Prototyp

Für die einzelnen Komponenten des Prototyps kamen verschiedene Technologien zum Einsatz, welche im Folgenden kurz vorgestellt werden sollen.

Der Prototyp wurde in **Java**[1] Version 1.7 implementiert. Java hat den Vorteil, dass Neo4j selbst eine Java-Datenbank darstellt und damit Java umfassend unterstützt wird. Herausragendes Merkmal von Java ist, dass der Java-Code zuerst in maschinenunabhängigen Bytecode compiliert wird und jener dann zur Ausführungszeit interpretiert wird [37]. Dadurch wird eine Plattformunabhängigkeit erreicht.

Das Data Warehouse und der relationale Link Store wurden mit **DB2**[2] Version 10.5 umgesetzt. DB2 ist ein ausgereiftes kommerzielles relationales DBMS von IBM.

Die graphbasierte Lösungsvariante des Link Stores wurde mit **Neo4j**[3] Version 2.1.2 Community implementiert. Die Testdaten-Erzeugung wurde hierbei mit Hilfe der Java-API erzeugt, während die Anfragen für die Use Cases als Cypher-Abfragen implementiert wurden.

Für das Content Management System kam Alfresco[4] in der Version 4.2.c zum Einsatz. Auf Alfresco wurde zurückgegriffen, weil es Open Source ist und von Java aus mittels CMIS (Content Management Interoperability Services)[5] angesprochen werden kann.

Für die Verbindung zwischen dem Java-Code und den relationalen Datenbanken wurde die **Java Database Connectivity (JDBC)**[6] verwendet. JDBC ermöglicht es, Datenbanken unterschiedlicher Hersteller über eine gemeinsame Schnittstelle anzusprechen.

1 https://www.java.com
2 http://www-01.ibm.com/software/data/db2/
3 http://www.neo4j.org/
4 http://www.alfresco.com/
5 https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cmis
6 http://www.oracle.com/technetwork/java/javase/jdbc/index.html

Hierbei kümmert sich JDBC darum, die Datenbankverbindungen aufzubauen, SQL-Anfragen an die Datenbank weiterzuleiten und die Datenbank-Ergebnisse zurück nach Java zu senden.

Für die Messungen wurden drei verschiedene Server verwendet, im Folgenden als „Server 1“, „Server 2“ und „Server 3“ bezeichnet, welche jeweils für eigenständige Aufgaben zuständig waren. Die Unterteilung in mehrere Server diente einer realistischen Annäherung an die Realität, wo die unterschiedlichen Teilsysteme gewöhnlicherweise ebenfalls auf unterschiedlichen Maschinen laufen.

Das Programm wurde folgendermaßen auf die Server verteilt:

- Server 1: Data Warehouse + Link Store für verschmolzene Lösung
- Server 2: CMS
- Server 3: Java + Link Store für separate Lösung

Sämtliche der drei Server wiesen folgende technische Spezifikation auf:

- Betriebssystem: Winsows Server 2008 R2 Standard
- Prozessor: Intel Xeon E312xx (Sandy Bridge) 2.0 GHz
- Arbeitsspeicher: 32 GB
- Systemtyp: 64 Bit-Betriebssystem

5.3 Entworfene Linkstruktur und Mengengerüst

Im Rahmen der Diplomarbeit wurde die bislang nur rudimentäre Linkstruktur um neue Linktypen erweitert. Abbildung 23 stellt die entworfene Linkstruktur dar.

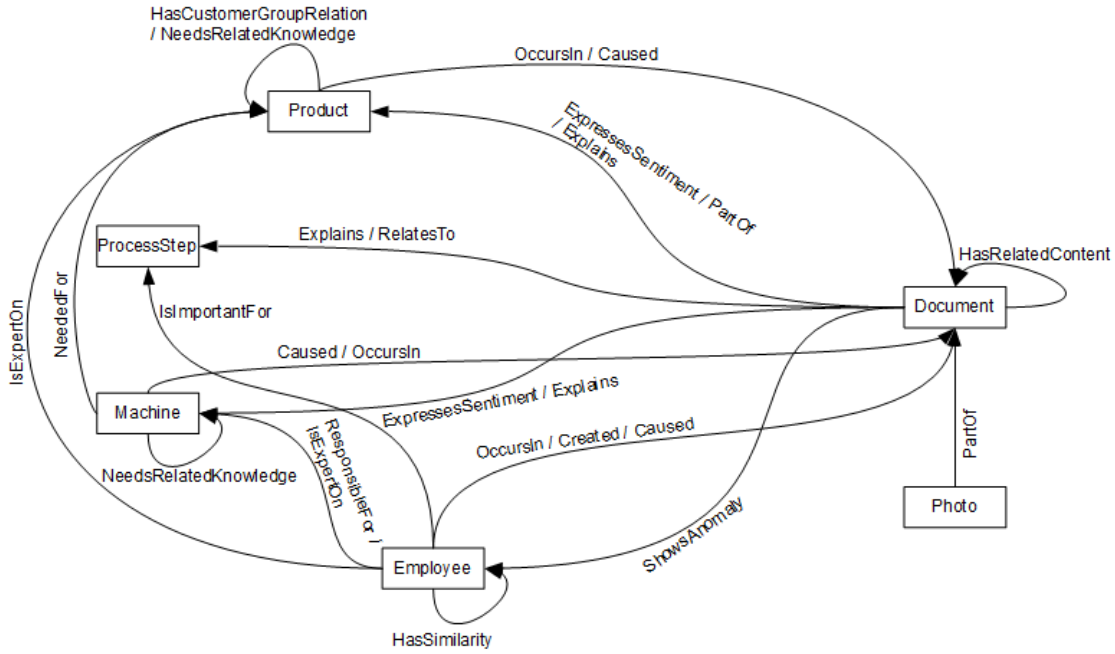

Abbildung 23: Linkstruktur für Prototyp

Pro Link existieren noch im Schnitt ungefähr sieben Attribute, von welchen, der Übersichtlichkeit halber, in der Abbildung abstrahiert wird. Für die verschiedenen Linktypen wurde eine möglichst große Masse angestrebt, um ein großes Volumen ins Mengengerüst zu bekommen.

Hinsichtlich des Mengengerüsts wurde lediglich das Link Store-Volumen variiert, während die Daten im Data Warehouse und im CMS konstant gehalten wurden. Im Data Warehouse wurden hierbei 1 Million Tupel pro Tabelle erzeugt, was insgesamt zu 4 Millionen Einträgen führt. Data Warehouse-Tabellen, welche eine Gruppierung darstellen, wie beispielsweise Employee-Group oder Product-Group, wurden mit einer weit geringeren Menge an Datensätzen generiert, was in Hinsicht auf die entworfenen Anfragen dem Zweck diente, die Ergebnismengen hochzuhalten. Für das CMS wurden lediglich 10000 PDF-Dokumente und 3333 Photos generiert. Diese verhältnismäßig geringe Anzahl war durch Limitierungen von Alfresco bzw. der Alfresco-Schnittstelle bedingt. Geplant wäre ansonsten ein ähnliches Volumen wie für das Data Warehouse gewesen. Für den Link Store wurden verschiedene Volumenstufen gemessen, um das Laufzeitverhalten genauer spezifizieren und erklären zu können. Was die Obergrenze betrifft, gaben allerdings wieder technische Einschränkungen das Limit vor, hier in

Form der Laufzeit des relationalen Testdatengenerators. Eine Erhöhung der Links pro Linktyp um 1 bewirkt bereits eine 410 fache Erhöhung der Tupel für die Linktyp-Attribute, was für Link-Volumina im unteren Millionenbereich schnell zu Milliarden an Tupeln insgesamt führt. Der Testdatengenerator fügt momentan 10 Millionen Tupel in etwas unter einer Stunde in DB2 ein. Erzeugt wurden letzlich die Volumina wie in Tabelle 2 dargestellt. Grau markierte Einträge weisen darauf hin, dass ohne die Beschränkung des CMS hier mehr Elemente erzeugt worden wären.

Volumen pro Element-Typ	**25**	**50**	**500**	**5000**	**50000**	**100000**
Volumen pro Linktyp	**50**	**100**	**1000**	**10000**	**100000**	**200000**
Anzahl Links	1300	2600	26000	260000	2600000	5200000
Anzahl Elemente	200	400	4000	33333	213333	413333
Anzahl Linktyp-Attribute	6400	12800	128000	1280000	12800000	25600000

Tabelle 2: Erzeugte Link Store-Volumina

Um die Anzahl der Ergebnisse hochzuhalten, wurden doppelt so viele Links pro Linktyp wie Elemente pro Element-Typ erzeugt. Eine verhältnismäßig hohe Anzahl von Links pro Linktyp im Vergleich zu den Elementen pro Element-Typ führt bei den meisten Use Cases zu mehr Ergebnissen. Zu beachten ist, dass hinsichtlich der Elemente nicht mehr Elemente pro Element-Typ erzeugt werden können, wie es zugehörige Data Warehouse bzw. CMS-Elemente gibt. Hinsichtlich des Data Warehouse ist dies unproblematisch, da das Data Warehouse-Volumen das Link Store-Volumen übersteigt. Dafür sind die Photo-Elemente und Dokument-Elemente allerdings auf 3333 bzw. 10000 Einträge gedeckelt. Diese Deckelung hat Einfluss auf die Dichte der Links. Für die Elemente, welche Data Warehouse-Gegenstücke haben, gilt, dass sie im Schnitt mit zwei Links eines konkreten Linktyps verbunden sind. Dieser lineare Zusammenhang gilt unabhängig von der gewählten Volumenstufe. Die Links auf CMS-Elemente (Dokument, Photo) werden mit zunehmenden Volumenstufen allerdings zunehmend dichter pro Element. Im Schnitt ist ein Dokument-Element pro Linktyp mit „Volumen pro Element-Typ“ * 2 / 10000 Links verbunden. Dies bedeutet beispielsweise für ein Volumen von 100000 Elementen pro Element-Typ, dass ein Dokument-Element mit 20

Links eines konkreten Linktyps verbunden ist. Für Photos ist der Nenner nicht mehr 10000 sondern 3333.

5.4 Entwurf und Durchführung der Use Cases

In diesem Abschnitt werden die entworfenen Anfragen zum Zugriff auf den Link Store spezifiziert und durchgeführt. Um realistische Zahlen zu erzielen, wurden sämtliche Use Cases fünf mal hintereinander ausgeführt und dann wurde der Schnitt aus den letzten drei Messungen genommen. Insbesondere Neo4j muss eine Warm-Up-Phase zugestanden werden, um die vorhandenen Caches zu füllen [38]. Wenn einzelne Anfragen innerhalb von zehn Minuten keine Ergebnisse lieferten, wurden sie abgebrochen.

Die relationalen Varianten mit separatem Linksture und DW-Zugriff und die graphbasierte Variante folgen demselben Ablaufplan, im Wesentlichen wird also nur an den betreffenden Stellen eine unterschiedliche SQL-Anfrage bzw. eine Cypher-Anfrage gestellt. Die Nummern unter „Verarbeitung“ in den Tabellen drücken den jeweiligen Schritt aus. Für die Messungen werden die Zeiten für DW, Link Store, Alfresco, Java und Gesamt unterschieden, wobei die Zeit für Java keinem speziellen Schritt zugeordnet wird, sondern der Differenz der Gesamtzeit und der Summe der Einzelzeiten entspricht. Da der Ablauf pro Use Case, unabhängig von der Variante, identisch ist, bedingt dies, dass für das CMS und das DW stets identische Zeiten gemessen werden. Auch die Zeit für die Java-Berechnungen ist im Normalfall konstant über die Varianten, allerdings wurden für einige Use Cases, in der graphbasierten Variante, Optimierungen vorgenommen, welche Berechnungen, welche eigentlich in der Graphdatenbank stattfinden, nach Java verschieben. In diesem Fall kann die Java-Zeit für die graphbasierte Variante größer als für die relationalen Varianten sein.

Aufgrund der gewöhnlicherweise sehr geringen Ergebnismengen für die Use Cases mussten pro Volumenstufe die Parameter der Anfragen angepasst werden, um Ergebnisse zu erzielen. Wenn damit dann beispielsweise in den Use Case-Beschreibungen Aussagen wie „Erhalte alle Failure Reports für Produkt XYZ“ stehen, kann es sein, dass für eine bestimmte Volumenstufe nicht die Failure Reports sondern die Improvement Suggestions ausgegeben werden.

Nicht gemessen wurden die Zeiten für die verschmolzenen Varianten. Da in element.LEID Einträge wie „pir.machine.ID=1“ stehen und nur der Teil rechts vom „=“-Zeichen dem Primärschlüssel entspricht, muss mittels der CONCAT-Funktion von SQL die Beziehung zwischen Element.LEID und dem Primärschlüssel hergestellt werden, also „WHERE element.LEID = CONCAT('pir.machine.ID=', dwMachine.machine_ID)“. Die Notwendigkeit der Berechnung über CONCAT führt zu einer Aufhebelung der Indizierung und damit letzlich zu einem inakzeptablem Laufzeitverhalten.

Entsprechend der Klassifikation von Abschnitt 2.4 handelt es sich bei den Use Cases 1, 2, 3 und 4 um Einschritt-Anfragen, während die Use Cases 5,6,7,8 und 12 Mehrschritt-Anfragen darstellen. Die beziehungsorientierten Anfragen der Use Cases 9,10 und 11 wurden nur für die Graphdatenbank implementiert. Sämtliche Use Cases wurden aus [4] entnommen, mit Ausnahme von Use Case 12, welcher in Anlehnung an [30] erstellt wurde.

5.4.1 Use Case 1

Beschreibung	Erhalte alle „Failure Reports“, von welchen das Produkt „Bulldozer PR1“ Bestandteil ist, unter folgenden Einschränkungen: • Der Link darf spätestens am „1.1.1990“ erzeugt worden sein.	
Eingabeparameter	• Dokumententyp: Failure Report • Produktname: Bulldozer PR1 • Entstehungsdatum: 1.1.1990	
Ausgabe	Liste gefilterter Dokumente	
Verarbeitung	1 - DW	Erhalte das WH-Produkt für „Bulldozer PR1“
	2 - Link Store	Erhalte das zum WH-Produkt zugehörige Produkt-Element
	3 - Link Store	Erhalte sämtliche Dokument-LEIDs mit (Dokument) - [PartOff] - (Produkt-Element von Schritt 2)
	4 - CMS	Filtere aus den Dokument-LEIDs von Schritt 3 jene LEIDs heraus, welche Failure Reports repräsentieren.
	5 - CMS	Erhalte zu den Failure Report-LEIDs von Schritt 4 die passenden Failure Reports

Tabelle 3: Use Case 1

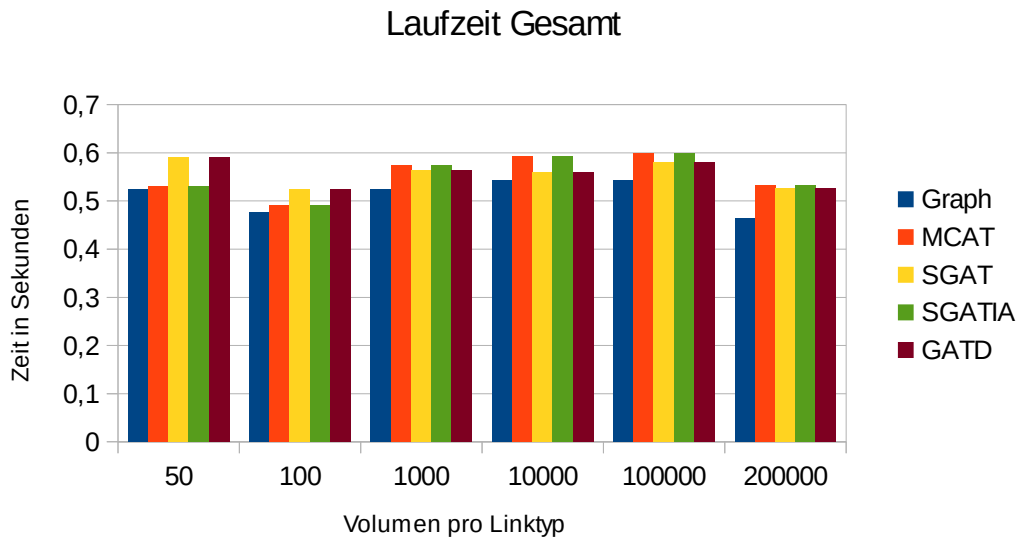

Abbildung 24: Gesamtlaufzeit Use Case 1

Tabelle 3 stellt den Use Case dar und Tabelle 15 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte. Wie aus Abbildung 24 ersichtlich, liegen die festgehaltenen Zeiten für sämtliche Volumenstufen und für sämtliche Lösungsvarianten konstant um 0,5 Sekunden herum. Dieses Ergebnis ist durch die geringe Menge der Ergebnisse in den einzelnen Schritten bedingt.

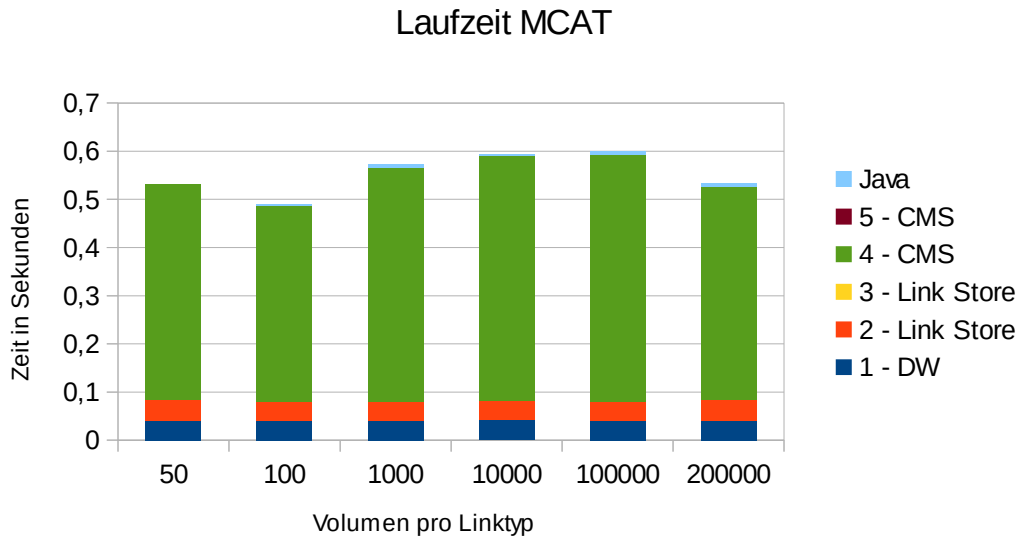

Abbildung 25: Laufzeit MCAT Use Case 1 im Detail

Abbildung 25 zeigt die Laufzeit für die einzelnen Schritte von MCAT, stellvertretend für sämtliche Lösungsvarianten. Bereits im ersten Schritt erfolgt eine Eingrenzung auf ein einziges Produkt und die Anzahl der Dokumente für solch ein einzelnes Produkt wird sich, unabhängig von der Volumenstufe, erwartungsgemäß im einstelligen Bereich bewegen. In den Schritten 2 und 3 verhindert die Indizierung des Link Stores eine messbare Abhängigkeit von den darin gespeicherten Elementen. Da die Zwischenergebnis-Menge hier nicht von der Volumenstufe abhängt und das Alfresco-Volumen konstant ist, ist die Laufzeit der CMS-Zugriffe der Schritte 4 und 5 ebenfalls unabhängig von der Volumengröße.

Verallgemeinert kann also folgende Erkenntnis aus diesem Use Case sowohl für die graphbasierte als auch für die relationalen Varianten fesgehalten werden: Wenn die Anfragen keine komplexen Bedingungen beinhalten und die Ergebnismenge sehr gering ist, ist unabhängig von dem gewählten Volumen eine sehr gute Laufzeit zu erwarten.

5.4.2 Use Case 2

Beschreibung	Erhalte alle Maschinen, welche Teil der Maschinengruppe „G1“ sind.	
Eingabeparameter	• Maschinengruppenname: G1	
Ausgabe	Liste gefilterter Maschinen	
Verarbeitung	1 - DW	Erhalte alle Maschinen, welche Teil der Maschinengruppe „G1“ sind.

Tabelle 4: Use Case 2

Tabelle 4 stellt den Use Case dar und Tabelle 16 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte. Da der Use Case lediglich auf das Data Warehouse zugreift, ist die Ausführungszeit damit unabhängig vom Link Store-Volumen und liegt bei ungefähr einer halben Sekunde. Der Ausführungsplan von Abbildung 26 birgt keine Überraschungen und ist grob folgendermaßen zu lesen:
Zuerst wird die Maschinengruppe Namens G1 identifiziert, wobei der zugehörige UNIQUE-Index auf dem Maschinengruppennamen benutzt wird. Anschließend werden diejenigen Einträge der Maschinen-Tabelle, wo die Maschinengruppen-ID zur erhaltenen Maschinengruppe passt, mit dieser Maschinengruppe gejoint. Da auf Machine.Machine_Group_ID kein Index liegt, muss hierfür per Full-Table-Scan die gesamte Ma-

chine-Tabelle durchgeführt werden. Hier auf einen Index zu verzichten ist aufgrund der geringen Kardinalität der Zielspalte vertretbar und sollte keine deutlichen Performancenachteile mit sich bringen.

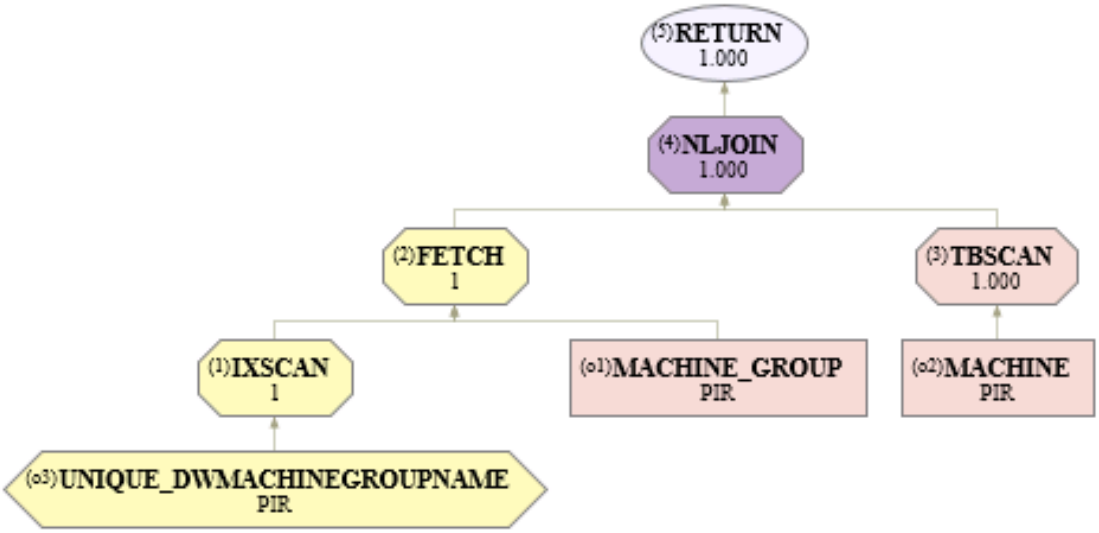

Abbildung 26: Ausführungsplan Use Case 2

5.4.3 Use Case 3

Beschreibung	Erhalte alle Angestellten, welche den „Failure Report“ namens „Dokument 3“ verursachten. Falls „Dokument 3“ kein „Failure Report“ ist soll eine Exception geworfen werden.	
Eingabeparameter	• Dokumententyp: Failure Report • Dokumentenname: Dokument 3	
Ausgabe	Liste gefilterter Angestellter	
Verarbeitung	1 - CMS	Erhalte das CMS-Dokument Namens „Dokument 3“
	2 - CMS	Überprüfe, ob es sich beim Dokument von Schritt 1 um einen Failure Report handelt.
	3 - Link Store	Berechne alle Employee-LEIDs, welche den besagten Failure Report verursachten.
	4 - DW	Berechne die DW-Employees zu den Employee-LEIDs von Schritt 3

Tabelle 5: Use Case 3

Tabelle 5 stellt den Use Case und Tabelle 17 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte. Für Use Case 1 wurde als Ergebnis festgehalten, dass wenn, die Anfrage einfach gestrickt ist und die Anzahl der Zwischenergebnisse klein

ist, eine gute Laufzeit, unabhängig von der Volumenstufe, zu erwarten ist. Von den Voraussetzungen her gleicht Use Case 3 hier dem Use Case 1 und tatsächlich ist die Laufzeit, ersichtlich anhand Abbildung 27, unabhängig von der Volumenstufe und der gewählten Lösungsvariante, sehr gering.

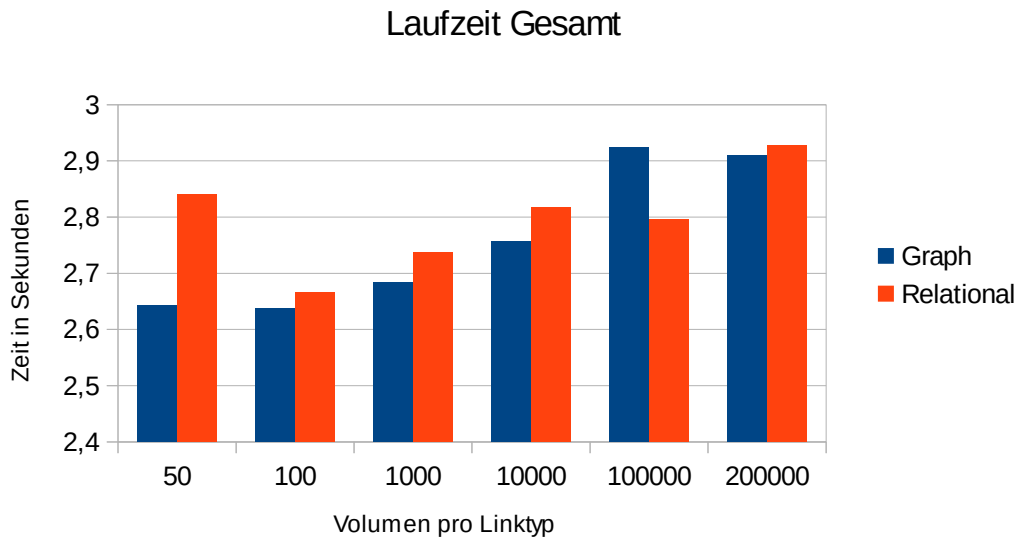

Abbildung 27: Gesamtlaufzeit Use Case 2

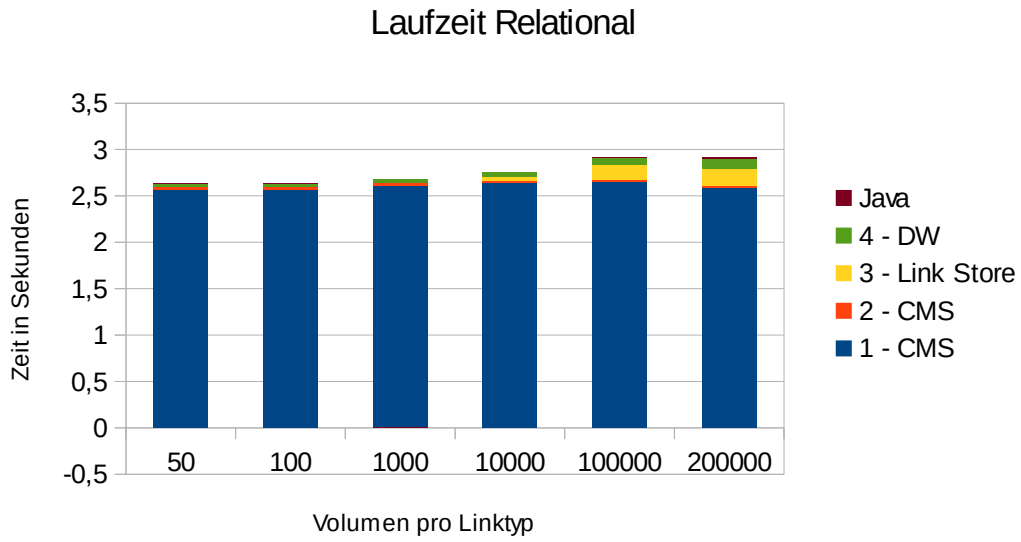

Abbildung 28: Laufzeit relationale Variante Use Case 2 im Detail

Abbildung 28 zeigt die Laufzeit für die relationale Lösungsvariante, welche sich nicht prinzipiell von der graphbasierten Laufzeit unterscheidet. In Schritt 1 erfolgt die Re-

striktion auf ein einzelnes Dokument und aufgrund des Verhältnisses der Elemente zu den Links, ist mit ungefähr zwei Angestellten zu rechnen, welche dieses Dokument verursachten. In Schritt 1 fällt auch der weit größte Anteil der Laufzeit an, wo das entsprechende Dokument im CMS gesucht wird. Dies ist durch die CMS-Schnittstelle bedingt, wo über sämtliche Dokumente iteriert wird, bis das passende Dokument gefunden wird.

Da keine Restriktionen auf den Link-Attributen bestehen, wird zwischen den vier relationalen Lösungsvarianten nicht unterschieden. Hinsichtlich des Link Stores ist für kleine Volumina die Laufzeit der graphbasierten Variante etwas schneller. Für die geringe Laufzeit der kleinen Volumina im Bereich von Hundertstelsekunden könnte ein geringerer Overhead des graphbasierten Link Stores ausschlaggebend sein.

5.4.4 Use Case 4

Beschreibung	Erhalte alle Angestellten, welche noch nie einen „Failure Report“ verursacht haben.	
Eingabeparameter	• Dokumententyp: Failure Report	
Ausgabe	Liste gefilterter Angestellter	
Verarbeitung	1 - CMS	Erhalte alle Failure Reports
	2 - Link Store	Erhalte alle Employee-LEIDs, welche nie Failure Reports verursachten
	3 - DW	Berechne die DW-Employees zu den Employee-LEIDs von Schritt 2

Tabelle 6: Use Case 4

Tabelle 6 stellt den Use Case dar und Tabelle 18 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte.

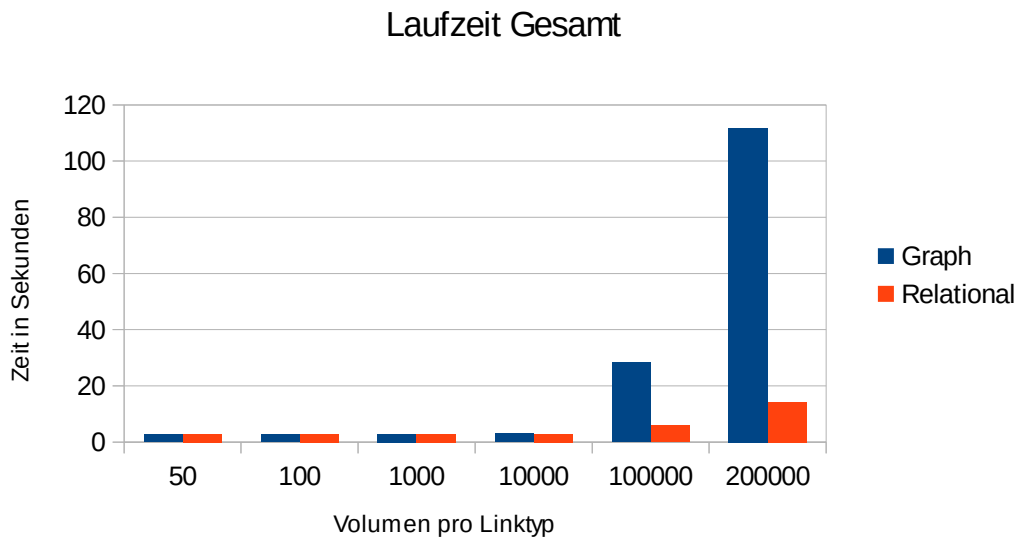

Abbildung 29: Gesamtlaufzeit Use Case 4

Wie anhand Abbildung 29 ersichtlich, gibt es bei diesem Use Case größere Laufzeitunterschiede zwischen der relationalen und der graphbasierten Lösungsvariante. Mit zunehmendem Volumen wird die Laufzeit der graphbasierten Lösungsvariante zunehmend schlechter.

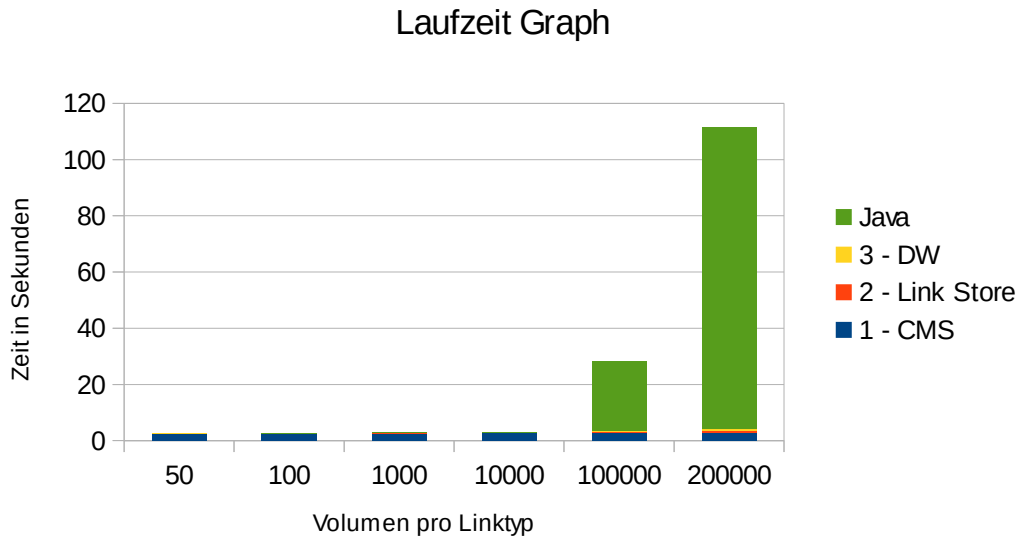

Abbildung 30: Laufzeit graphbasierte Variante Use Case 4 im Detail

Abbildung 30 zeigt die Laufzeit für die einzelnen Schritte für die graphbasierte Variante. In Schritt 1 wird über sämtliche Alfresco-Dokumente iteriert um die Failure Reports zu erhalten. Demgemäß ist diese Laufzeit über sämtliche Lösungsvarianten konstant. Die Laufzeit des DW-Zugriffs von Schritt 3 steigt mit zunehmender Volumenstufe an. Dies ist der Fall, da mit zunehmender Volumenstufe zunehmend mehr Angestellte keine Failure Reports verursachten. Jene Angestellten-IDs werden in dieser Implementierung in der WHERE-Klausel aufgelistet, also „WHERE employee_ID IN ('E1','E2','E3',...)“. Mit zunehmender Länge wird die Laufzeit hier zunehmend schlechter. Es ist ein großer Nachteil der separaten gegenüber der verschmolzenen Lösungsvariante, dass hier eine lange Liste von Objekten in der WHERE-Klausel übergeben wird. Deutlich tritt die Diskrepanz zwischen den Laufzeiten der einzelnen Schritte und der Gesamtlaufzeit der relationalen Lösungsvariante mit zunehmender Volumenstufe hervor. Bei höchster Volumenstufe sind dies bereits ungefähr 9 Sekunden und damit mehr als für die einzelnen Schritte benötigt wird. Verursacht wird diese Diskrepanz durch Java-Berechnungen zwischen Schritt 2 und 3, um die Indirektion zwischen Element.Leid und Employee.Employee_ID aufzulösen. Ist die Employee_ID beispielsweise 10, steht in Element.Leid 'Pir.Employee.ID=10'. Um die Employee_ID zu erhalten, muss also alles, was vor dem '='-Zeichen steht, weggeschnitten werden und wenn diese teure String-Operation für einige zehntausend Employees stattfindet, erklärt dies die fehlenden Sekunden.

Die schlechte Laufzeit der graphbasierten Lösungsvariante bedarf einer genaueren Untersuchung. Verantwortlich für die aufwendigen Java-Berechnungen ist der Link Store-Zugriff von Schritt 2. Wie anhand der relationalen Lösungsvariante ersichtlich, sollte eine akzeptable Laufzeit für Schritt 2 unterhalb einer Sekunde liegen. Um überhaupt in die Nähe einer akzeptablen Laufzeit zu gelangen, wurde die Anfrage nach den Employees, welche nie Failure Reports verursachten, in zwei Teilanfragen aufgespalten und die Verbindung per Java-Code hergestellt. Zuerst werden die Employees berechnet, welche Failure Reports erzeugten und diese Menge an Employees wird dann in Java von der Menge sämtlicher Employees abgezogen. Diese Optimierung auf Neo4j-Seite führt zwar zu deutlich schnelleren Cypher-Abfragen, aber dafür wird das Problem nach Java übertragen, wo dann für größere Volumenstufen der Großteil der Lauf-

zeit anfällt. Trotzdem wird so eine insgesamte Laufzeitverbesserung um Größenordnungen erreicht.

5.4.5 Use Case 5

Beschreibung	Erhalte alle Manager aus der Abteilung „G1“, welche für einen Angestellten verantwortlich sind, welcher Dokumente verursachte, unter folgenden Einschränkungen: • Die Sicherheit des Links muss mindestens „79“ sein.	
Eingabeparameter	• Abteilungsname: G1 • Sicherheit: 79	
Ausgabe	Liste gefilterter Angestellter	
Verarbeitung	1 - Link Store	Erhalte alle Employees, welche Dokumente verursachten (unter den gegebenen Einschränkungen)
	2 - DW	Erhalte alle Manager der Abteilung „G1“, welche für einen Employee von Schritt 1 verantwortlich sind.

Tabelle 7: Use Case 5

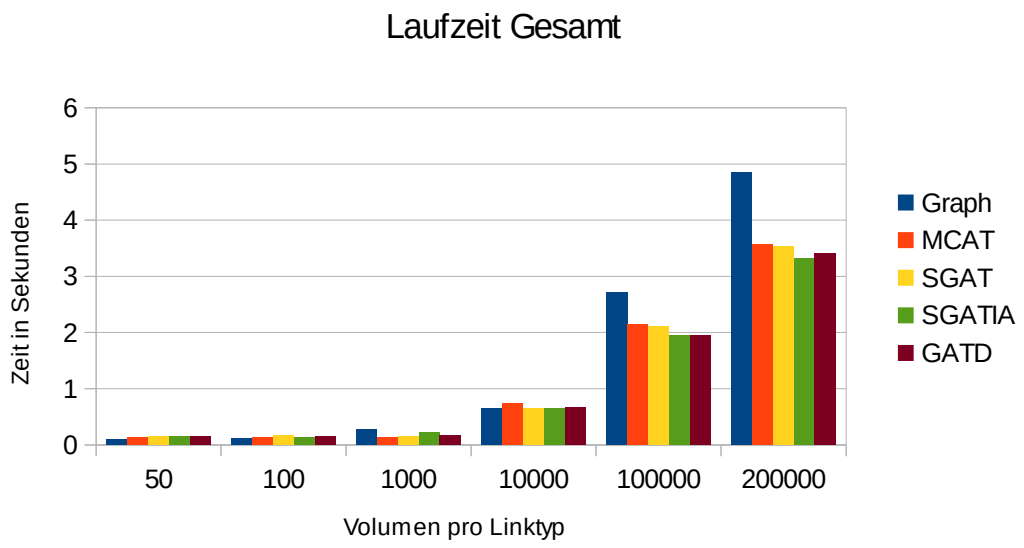

Abbildung 31: Gesamtlaufzeit Use Case 5

Tabelle 7 stellt den Use Case dar und Tabelle 19 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte. Wie anhand von Abbildung 31 ersichtlich, bewegt sich der Performance für sämtliche Lösungsvarianten stets innerhalb derselben Größenordnung. Höhrere Volumina führen zu einer etwas größeren Laufzeit, wobei die Laufzeit über alle Volumenstufen in einem akzeptablen Rahmen bleibt.

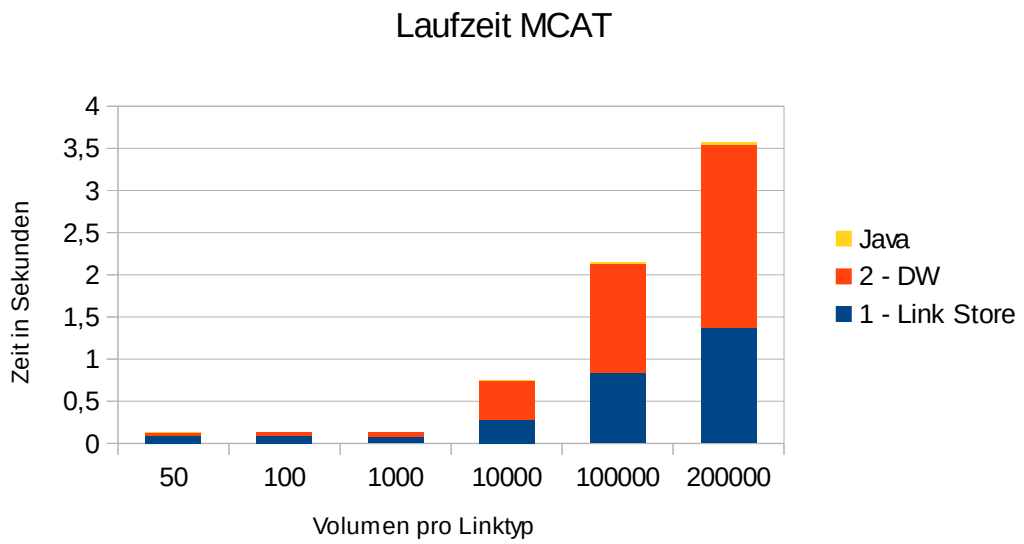

Abbildung 32: Laufzeit MCAT Use Case 5 im Detail

Abbildung 32 zeigt die Laufzeit für MCAT, stellvertretend für die graphbasierte und die anderen relationalen Varianten, welche ein ähnliches Laufzeitverhalten zeigen. Interessant ist der Laufzeitanstieg vom Data Warehouse von einigen hundertstel Sekunden auf über zwei Sekunden für das größte Volumen. Der Data Warehouse-Generator wurde in einer Art und Weise geschrieben, dass jede Employee-Group, unabhängig von der Volumenstufe, genau zwei Manager enthält. Die Anfragen für größere Volumenstufen unterscheiden sich von denen für kleinere Volumenstufen dadurch, dass für größere Volumenstufen mehr passende Employees in der IN-Liste der WHERE-Klausel übergeben werden. Tatsächlich umfasst diese Liste für die höchste Volumenstufe 86528 Employees. Abbildung 33 zeigt den Ausschnitt aus dem Ablaufdiagramm, wo die Filterung stattfindet. Das Kästchen GENROW ist dabei so zu interpretieren, dass DB2 hier für die Filter-Elemente eine temporäre Tabelle anlegt und diese temporäre Tabelle wird dann mit den Employees gejoint, um die passenden Employees zu erhalten. Die temporäre Tabelle enthält allerdings keine Indexe, was zu einem eher langsamen Full Table Scan führt. Weiterhin fällt im DBMS entsprechend mehr Zeit an, um diese lange Anfrage zu parsen. Allgemein kann festgehalten werden, dass die separaten Lösungsvarianten typischerweise mit einer mehr oder weniger langen IN-Liste als Filter für das Data Warehouse einhergehen. Damit stehen die separaten Lösungsvarianten

im Widerspruch zu der Best Practice-Regel, derartige IN-Listen, wo möglich, zu vermeiden [39] und falls die IN-Liste stark ansteigt, führt dies zu großen Performanceverlusten.

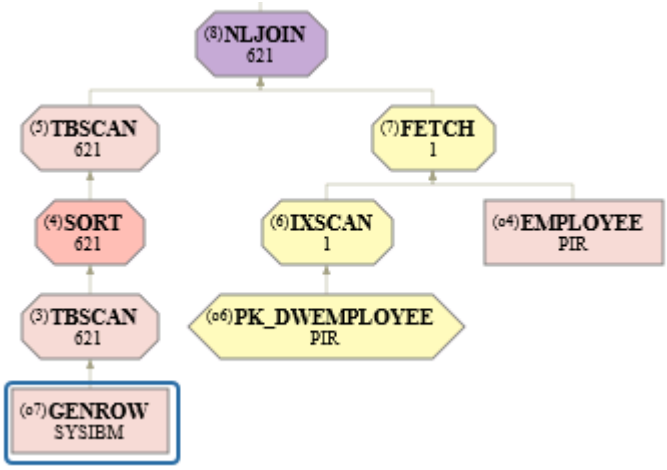

Abbildung 33: Filter-Ausschnitt aus Ablaufdiagramm für Use Case 5

5.4.6 Use Case 6

Beschreibung	Erhalte alle Maschinen, welche Teil der Maschinengruppe „G6“ sind und in bestimmten „Failure Reports“ auftauchen. Für die „Failure Reports“ muss dabei gelten, dass sie eine Beziehung zu einem Prozessschritt, gehörend zum „Prozess 1“, haben. Weiterhin müssen folgende Bedingungen erfüllt sein: • Die Sicherheit des OccursIn-Links muss mindestens „20“ sein.	
Eingabeparameter	• Maschinengruppenname: G6 • Dokumententyp: Failure Report • Prozess-Name: Prozess 1 • Sicherheit: 20	
Ausgabe	Liste gefilterter Maschinen	
Verarbeitung	1 - DW	Erhalte alle Prozess-Schritte gehörend zum „Prozess 1“.
	2 - Link Store	Erhalte alle Dokument-LEIDs, welche mit den den Prozess-Schritten von Schritt 1 verknüpft sind.
	3 - CMS	Filtere aus den Dokument-LEIDs die Failure Report-LEIDs heraus.
	4 - Link Store	Berechne die Maschinen-LEIDs, welche in den Failure Reports auftauchen.
	5 - DW	Berechne die zu den Maschinen-LEIDs zugehörigen DW-Maschinen und filtere jene heraus, welche zur Maschinengruppe „G2“ gehören.

Tabelle 8: Use Case 6

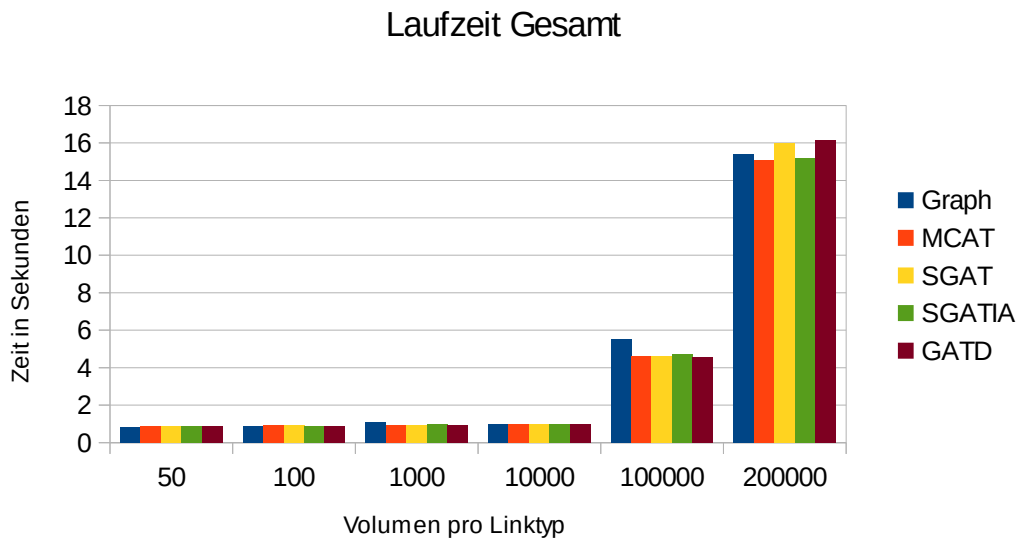

Abbildung 34: Gesamtlaufzeit Use Case 6

Tabelle 8 stellt den Use Case dar und Tabelle 20 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte. Wie aus Abbildung 34 ersichtlich, steigt die Laufzeit dieses Use Cases von ungefähr einer Sekunde für die kleinen Volumenstufen auf ungefähr 15 Sekunden für die höchste Volumenstufe an. Die vier relationalen Lösungsvarianten und die graphbasierte Lösungsvariante haben dabei annähernd gleiche Laufzeiteigenschaften.

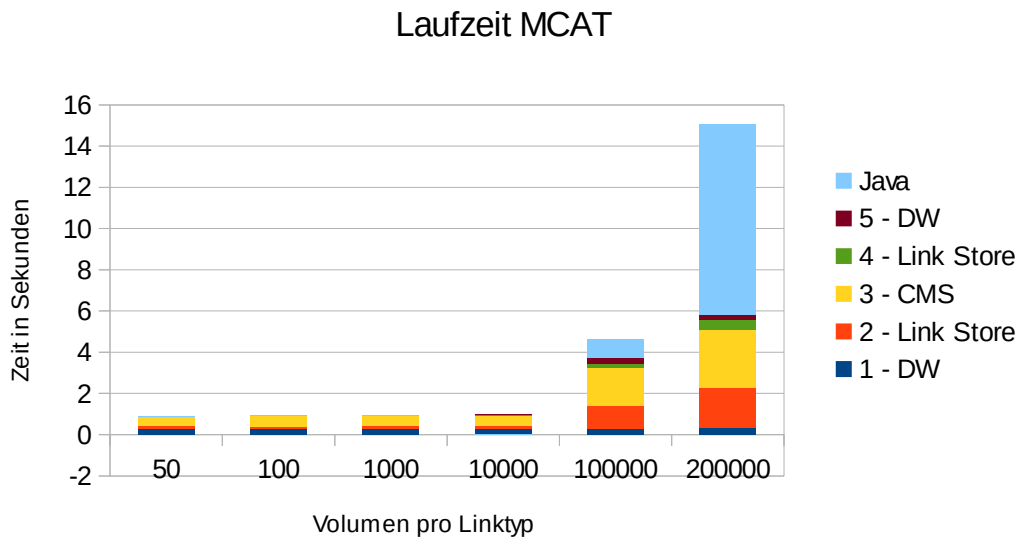

Abbildung 35: Laufzeit MCAT Use Case 6 im Detail

Abbildung 35 schlüsselt das Laufzeitverhalten für MCAT, stellvertretend für sämtliche Lösungsvarianten, genauer auf. Schritt 1 operiert auf dem Data Warehouse und ist unabhängig vom gewählten Volumen. Die Laufzeit von Schritt 2 ist zwar nicht herausragend groß, aber trotzdem wird hier wieder das Problem großer Listen in der IN-Klausel verdeutlicht. Diese Liste trägt die Verantwortung dafür, dass die relationale Laufzeit hier, für größere Volumenstufen, über eine Sekunde beträgt. Die Laufzeit für die graphbasierte Variante war für Schritt 2 ursprünglich nicht akzeptierbar lang, was aber durch eine Änderung in der Anfrage umgangen werden konnte. Die ursprüngliche Anfrage war wie in Listing 12 definiert.

```
match(document : document) - [relatesTo : RelatesTo] -> (prod : production_step)
where prod.leid in {processStepElementLEIDs} return distinct document.leid
```

Listing 12: Unperformante Cypher-Anfrage Use Case 6

Als Performanceproblem wurde dabei die lange Parameterliste processStepElementLEIDs identifiziert. Für Performancezwecke wurde die Anfrage dann ohne diese Einschränkung gestellt und die Ergebnisse, welche dann zu viel erscheinen, wurden mit reinem Java Code rausgefiltert. Der Java-Code arbeitet dabei im Bruchteil einer Sekunde. Es kann also festgehalten werden, dass lange IN-Listen nicht nur für die relatio-

nalen Lösungsvarianten schlecht sind, sonden für die graphbasierten Lösungsvarianten noch schlechtere Auswirkungen haben können.

Schritt 3 iteriert über die identifizierten Dokumente und stellt für jedes Element eine Alfresco-Anfrage nach dem Dokumententyp. Demgemäß hängt die Laufzeit hier von der Anzahl der Zwischenergebnisse ab.

Ersichtlich an der sprunghaft ansteigenden Java-Laufzeit für die größte Volumenstufe wird wieder das Problem der Indirektion zwischen Element.LEID und dem Primärschlüssel des Data Warehouse-Elements. Dieses Problem muss in Java aufgelöst werden und macht für die größeren Volumenstufen über die Hälfte der Laufzeit aus. Für die größte Volumenstufe muss hier die teure String-Operation, einen Substring zu finden, 45111 mal ausgeführt werden. Ohne diese String-Operation würde die Java-Laufzeit keinen nennenswerten Teil der Gesamtlaufzeit mehr ausmachen.

5.4.7 Use Case 7

Beschreibung	Erhalte alle „Manuals“, in welchen bestimmte Maschinen der Maschinengruppe „G1“ auftauchen. Für die Maschinen muss dabei gelten, dass sie in „Failure Reports“ auftauchen, unter der Einschränkung, dass die „Failure Reports“ von solchen Angestellten erzeugt sein müssen, welche auch „Improvement Suggestions“ für die Maschine erzeugten. Weiterhin müssen folgende Bedingungen erfüllt sein: • Die Sicherheit des OccursIn-Links muss mindestens „20“ sein. • Der Link darf spätestens am „1.1.1990“ erzeugt worden sein.	
Eingabeparameter	• Maschinengruppenname: G1 • Dokumententyp 1: Failure Report • Dokumententyp 2: Improvement Suggestion • Dokumententyp 3: Manual • Sicherheit: 20 • Enstehungsdatum: 1.1.1990	
Ausgabe	Liste gefilterter Dokumente	
Verarbeitung	1 - CMS	Erhalte sämtliche Failure Reports
	2 - CMS	Erhalte sämtliche Improvement Suggestions
	3 - CMS	Erhalte sämtliche Manuals
	4 - DW	Erhalte sämtliche Maschinen, gehörend zur Maschinengruppe „G1“

	5 - Link Store	Berechne die gefilterten Manual-LEIDs gemäß den Einschränkungen
	6 - CMS	Berechne die zugehörigen Manuals zu den Manual-LEIDs von Schritt 5

Tabelle 9: Use Case 7

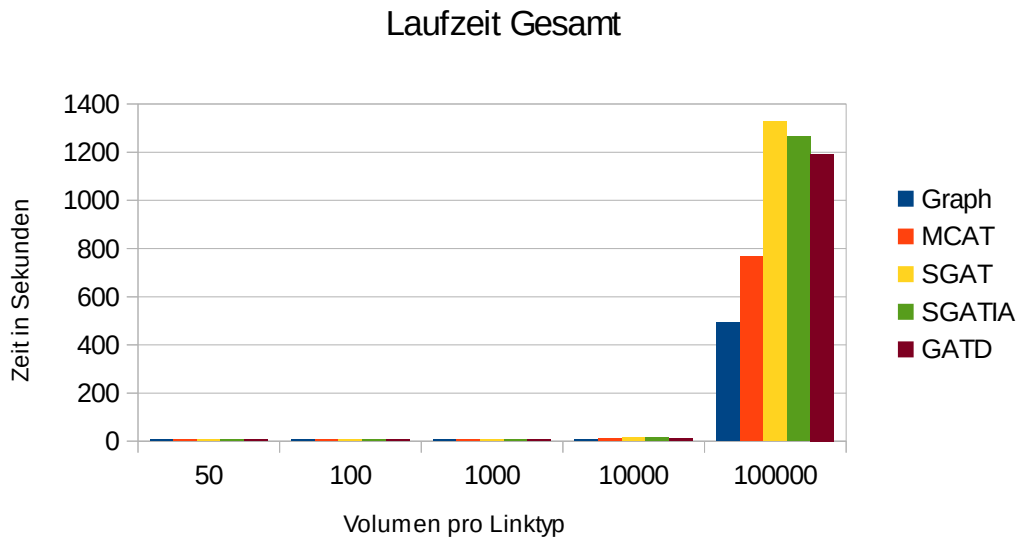

Abbildung 36: Gesamtlaufzeit Use Case 7

Tabelle 9 stellt den Use Case dar und Tabelle 21 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte. Wie aus Abbildung 36 hervorgeht, steigt die Gesamtlaufzeit für größere Volumenstufen stark an und erstmals ist die Laufzeit der graphbasierten Variante besser.

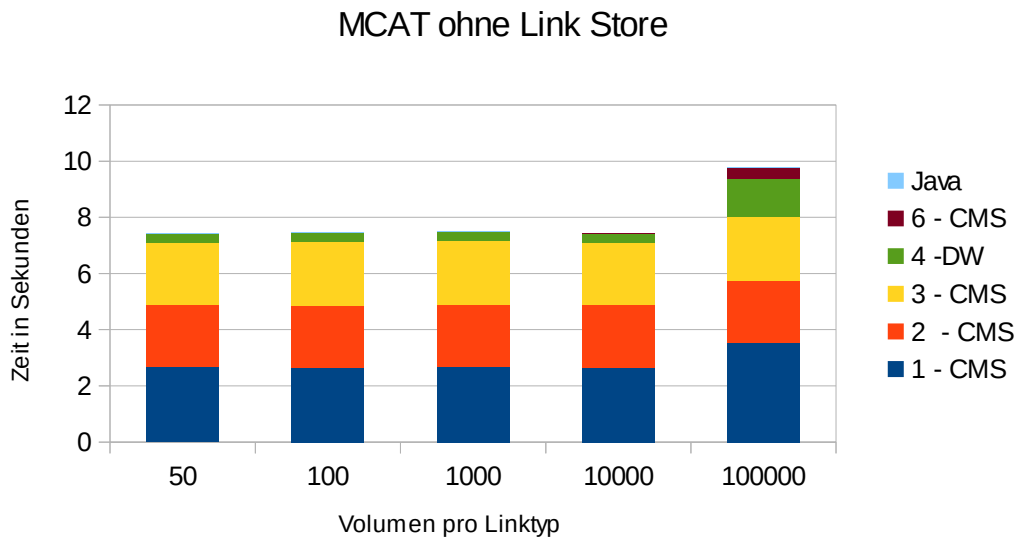

Abbildung 37: Laufzeit MCAT ohne Linkstore Use Case 7 im Detail

Da der langsame Linkstore-Zugriff für größere Volumina die anderen Werte in der Graphik überdecken würde, enthält Abbildung 37 die Zugriffszeiten ohne den Linkstore-Zugriff. Lediglich der nicht dargestellte Linkstore-Zugriff im Schritt 5 und der CMS-Zugriff von Schritt 6 weisen eine Abhängigkeit von der Volumenstufe auf, während die Laufzeit für die anderen Schritte weitgehend konstant bleibt.

In den ersten drei Schritten wird pro Schritt jeweils über sämtliche CMS-Dokumente iteriert und, wenn der Typ übereinstimmt, wird das Dokument zuürckgegeben. Die Alfresco-Schnittstelle wurde im Rahmen dieser Diplomarbeit nicht angepasst, hier wäre allerdings eine einfache Anpassung möglich, dass nur einmal über die Alfresco-Dokumente iteriert werden müsste, was für die ersten drei Schritte die Laufzeit durch drei teilen würde.

Der Data Warehouse-Zugriff von Schritt 4 entspricht dem von Use Case 2, beschrieben in 5.4.2 und demgemäß entspricht auch die Verarbeitungszeit der Verarbeitungszeit von Use Case 2 von Tabelle 16.

Für Schritt 5 versagt sowohl der graphbasierte als auch der relationale Zugriff darin, für größere Volumenstufen in ansprechender Zeit Ergebnisse zu liefern. Hinsichtlich des relationalen Zugriffs bestehen zwei Probleme. Zum Einen ist die Anfrage mit 13 Tabellen, welche hinzugejoint werden, sehr komplex und zum Anderen gibt es wieder relativ lange IN-Listen. Für die Joins ist es sehr wichtig, dass vor den Joins stark gefiltert wird, damit die Zwischenergebnis-Menge klein bleibt. Eine Analyse mithilfe des DB2 Query Optimizers deutet darauf hin, dass es pro Join hier zu viele Freiheitsgrade gibt, was dazu führt, dass die Zwischenergebnis-Menge zu groß ist. Ursprünglich war die graphbasierte Variante noch langsamer als die relationale Variante, aber es ließ sich eine Optimierung in der Abfrage vornehmen, um einen besseren Ausführungsplan zu erhalten. Entsprechend den Cypher Performance Tunning-Tipps, wurde die Anfrage ausgeflacht, um die Reihenfolge der Ausführung zu bestimmen. In diesem Fall half das Konstrukt von Listing 13, Neo4j zu zwingen, zuerst die Einschränkungen auf der Maschine vorzunehmen, was die Anzahl der Zwischenergebnisse deutlich minimierte und hier Performancegewinne um 1-2 Größenordnungen brachte.

```
match (machine : machine) where mahcine.leid in {machineLEIDsBelongingToG1}
with machine match [Rest der Anfrage]
```

Listing 13: Optimierte Cypher-Anfrage Use Case 7

Für SQL wurde sich darauf verlassen, dass das DBMS von selbst den besten Ausführungsplan findet.

5.4.8 Use Case 8

Beschreibung	Erhalte bestimmte Angestellte aus der Abteilung, welche vom Abteilungsleiter Namens „E7“ geleitet wird. Für die Angestellten muss dabei gelten, dass sie „Failure Reports“ über die Maschine „M3“ erzeugten, in welchen Photos, genommen vom Angestellten „E17“ auftauchen.	
Eingabeparameter	• Abteilungsleiter-Name: E7 • Dokumententyp: Failure Report • Maschinen-Name: M3 • Angestellten-Name: E17	
Ausgabe	Liste gefilterter Angestellter	
Verarbeitung	1 - Link Store	Erhalte sämtliche Photo-LEIDs
	2 - CMS	Filtere die Photos auf die Autorenschaft vom Angestellten „E17“
	3 - CMS	Erhalte sämtliche Failure Reports
	4 - DW	Erhalte die WH-Maschine Namens „M3“
	5 - Link Store	Berechne die gefilterten Employee-LEIDs gemäß den gestellten Bedingungen.
	6 - DW	Berechne jene DW-Employees, welche zu Schritt 5 gehören und die zusätzlich noch zur selben Abteilung wie der Abteilungsleiter „E7“ gehören.

Tabelle 10: Use Case 8

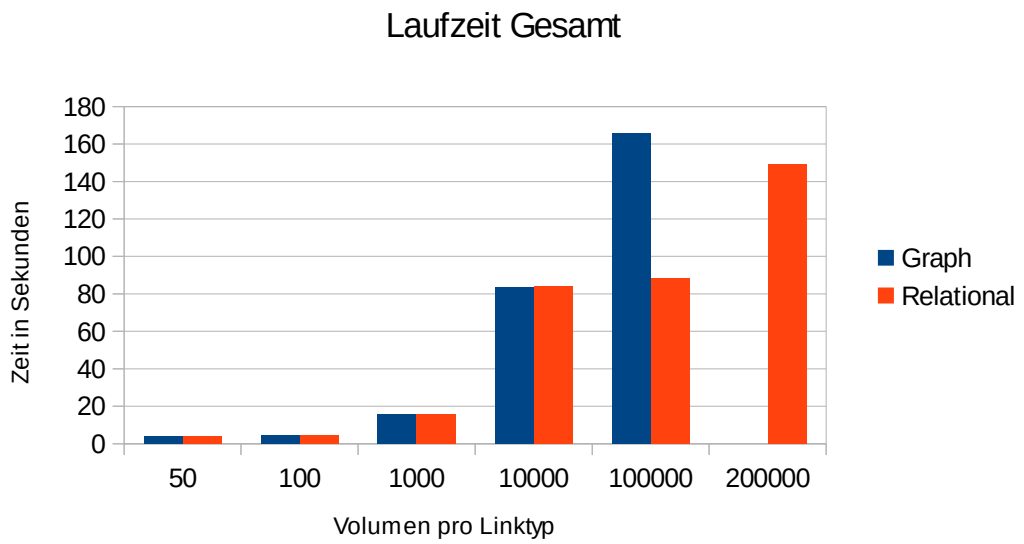

Abbildung 38: Gesamtlaufzeit Use Case 8

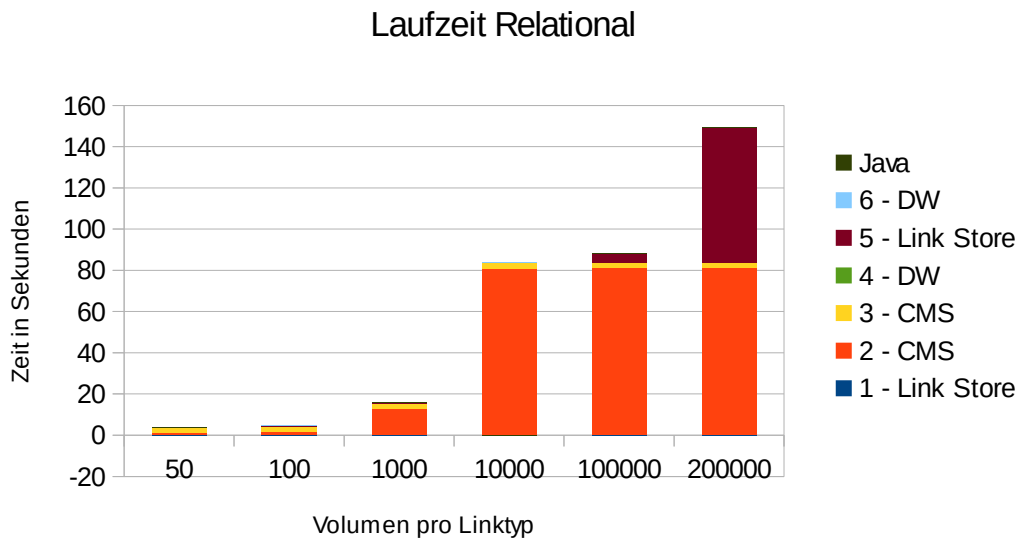

Abbildung 39: Laufzeit relationale Variante Use Case 8 im Detail

Tabelle 10 stellt den Use Case dar und Tabelle 22 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte. Wie aus Abbildung 39 ersichtlich, bereitet hier erstmals der CMS-Zugriff von Schritt 5 ernsthafte Performanceprobleme. Verantwortlich dafür, dass die graphbasierte Variante in Abbildung 38 schlechter als die relationale Variante abschneidet, ist die Zugriffszeit auf den Link Store, welcher, insbesondere für die graphbasierte Variante, mit zunehmendem Volumen rapide ansteigt.

Die ersten beiden Schritte dienen dazu, Photos mit dem Angestellten namens E17 als Autor herauszubekommen. Für den CMS-Zugriff wird dabei über alle im Link Store befindlichen Photos iteriert und der Autor des Photos herausgelesen. Der Zugriff auf Photos ist allerdings im Vergleich zum Zugriff auf gewöhnliche Dokumente unverhältnismäßig langsam, was zu dieser großen Laufzeit von bis zu über einer Minute führt. Die Schritte 3 und 4 hängen nicht vom gewählten Volumen ab. Der Link Store-Zugriff von Schritt 5 ist durch die vielen Freiheitsgrade in der Anfrage beeinträchtigt, was dazu führt, dass für größere Volumina zu viele Vergleiche stattfinden müssen, um eine ansprechende Performance zu erhalten.

5.4.9 Use Cases 9-11

Beschreibung	Erhalte den kürzesten Pfad zwischen einem gegebenen Document und einem gegebenen Employee. Falls mehrere kürzeste Pfade existieren, wähle willkürlich einen aus.	
Eingabeparameter	- Dokumenten-ID: 123 - Angestellten-ID: 50	
Ausgabe	Einzelner kürzester Pfad	
Verarbeitung	1 - Link Store	Berechne den kürzesten Pfad zwischen dem Dokument und dem Employee

Tabelle 11: Use Case 9

Beschreibung	Erhalte alle kürzesten Pfade zwischen zwei konkreten Maschinen.	
Eingabeparameter	- Maschinen-ID 1: 10 - Maschinen- ID 2: 11	
Ausgabe	Liste sämtlicher kürzester Pfade	
Verarbeitung	1 - Link Store	Berechne sämtliche kürzesten Pfade zwischen den beiden Maschinen

Tabelle 12: Use Case 10

Beschreibung	Erhalte sämtliche Pfade zwischen zwei konkreten Maschinen.	
Eingabeparameter	- Maschinen-ID 1: 10 - Maschinen-ID 2: 11	
Ausgabe	Liste sämtlicher Pfade	
Verarbeitung	1 - Link Store	Berechne sämtliche Pfade zwischen den beiden Maschinen

Tabelle 13: Use Case 11

Da die beziehungsorientierten Use Cases 9-11 alle demselben Ablauf folgen, werden sie hier zusammengefasst. Die Tabellen 11, 12 und 13 stellen die Use Cases dar und die gemessenen Zeiten finden sich in den Tabellen 23, 24 und 25. Für jeden dieser strukturell gleich aufgebauten Use Cases gibt es jeweils in Neo4j vordefinierte Funktionen, welche lediglich aufgerufen werden müssen, um das erwünschte Ergebnis zu erhalten. Demgemäß ist der Ablaufplan denkbar einfach. Jene beziehungsorientierten Anfragen sind gewissermaßen das Aushängeschild der Graphdatenbanken, da hier eine Anfrage, welche mit SQL nicht direkt erfüllbar ist, mit einer Zeile Code gelöst wird.

Wie die Messungen in 23 und 24 zeigen, wird die Anfrage nach einem simplen kürzesten Pfad bzw. sämtlichen kürzesten Pfaden im Bereich von Hundertstelsekunden gelöst. Eine genaue Zeitangabe in diesem Bereich wäre unseriös, da für derart geringe Zeiten der Aufwand für die Zeitmessung das Ergebnis determinieren würde. Dem Charakteristikum der Graphdatenbanken entsprechend, hängt die gemessene Zeit nicht vom Volumen des Graphen ab. Allerdings sollte die enorm schnelle Zeit nicht unreflektiert akzeptiert werden. Da für einen Festplattenzugriff mindestens 10 Millisekunden eingeplant werden müssen [40], muss der benötigte Teil des Graphen im Arbeitsspeicher gehalten werden. Weiterhin hält Neo4j hier nicht nur den Graphen, sondern auch den Ausführungsplan für die Anfrage im Cache. Damit ist dann allerdings nicht mehr klar, inwiefern während der Anfrage tatsächlich kürzeste Pfade berechnet werden und inwiefern auf vorkalkulierte Ergebnisse zugegriffen wird.

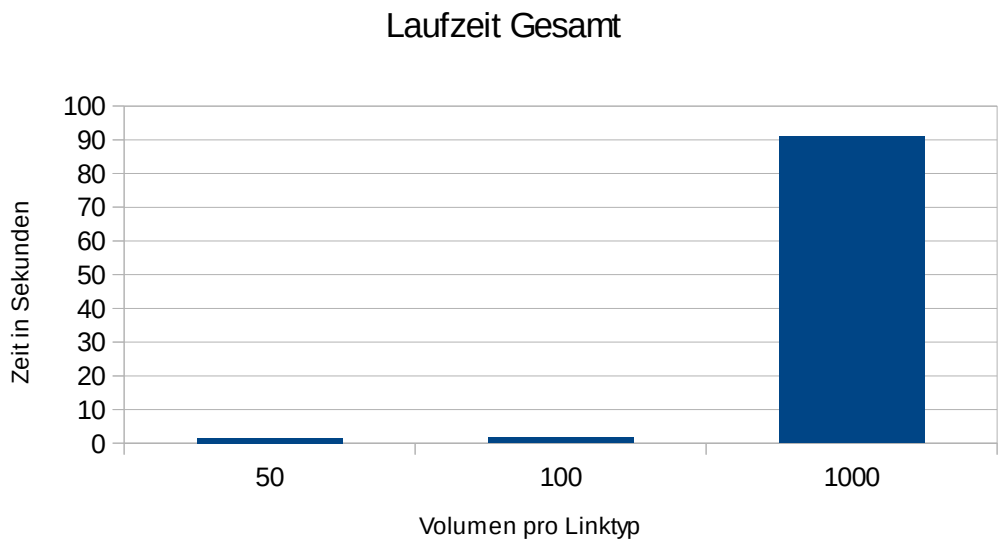

Abbildung 40: Gesamtlaufzeit Use Case 11

Wie Abbildung 40 zeigt, ist die Laufzeit für die Berechnung sämtlicher Pfade deutlich schlechter als für die Berechnung der kürzesten Pfade. Dieses Ergebnis muss vor dem Hintergrund der kombinatorischen Explosion der möglichen Pfade in Abhängigkeit von der Volumenstufe interpretiert werden. Um die Ergebnismenge einigermaßen kleinzuhalten, wurde in den Anfragen die maximale Pfadlänge auf 4 begrenzt. Trotzdem kann eine einzige neu hinzukommende Kante die Ergebnismenge theoretisch ma-

ximal bis zu verdoppeln, indem einmal die Pfade ohne diese Kante und einmal die Pfade mit dieser Kante gezählt werden. Für ein Volumen von 50 liegen bereits 649 Ergebnisse in der Lösungsmenge.

5.4.10 Use Case 12

Beschreibung	Dies ist ein großer Use Case um die Performance des Link Stores bei einer längeren Pfadlänge in der Anfrage zu testen. Getestet werden die Pfadlängen 10, 20 und 30, was hierbei so zu verstehen ist, dass beispielsweise bei einer Pfadlänge von 10 nur die ersten 10 Bedingungen erfüllt sein müssen. Weiterhin wird neben MCAT, SGAT, SGATIA und GATD noch die Konfiguration NONE, wobei NONE für das Nicht-Vorhandensein von Einschränkungen steht. Da die Semantik bei diesem Use Case nicht im Vordergrund steht, werden ansonsten die Einschränkungen in einer Art formuliert, das keine echte Einschränkung vorliegt, also z. B. der Form „WHERE confidenceLevel >= 0“. 1. Erhalte alle Employees (Employee-IDs) unter gewissen Einschränkungen: 2. Hinsichtlich der Employees von Schritt 1 muss gelten, dass X - [IsExpertOn] -> Product 3. Hinsichtlich der Products von Schritt 2 muss gelten, dass X - [OccursIn] -> Document 4. Hinsichtlich der Documents von Schritt 3 muss gelten, dass Photo - [PartOf] -> X 5. Hinsichtich der Documents von Schritt 3 muss außerdem gelten dass X - [Explains] -> Machine 6. Hinsichtlich der Machines von Schritt 5 muss gelten, dass X - [NeedsRelatedKnowledge] -> Machine 7. Hinsichtlich der Machines von Schritt 6 muss gelten, dass Document - [ExpressesSentiment] -> X 8. Hinsichtlich der Documents von Schritt 7 muss gelten, dass X - [ShowsAnomaly] -> Employee 9. Hinsichtlich der Employees von Schritt 8 muss gelten, dass X - [HasSimilarity] -> Employee 10. Hinsichtlich der Employees von Schritt 9 muss gelten, dass X - [IsImportantFor] -> ProcessStep 11. Hinsichtlich der ProcessSteps von Schritt 10 muss gelten, dass Document - [Explains] -> X 12. Hinsichtlich der Documents von Schritt 11 muss gelten, dass X - [Explains] -> Product 13. Hinsichtlich der Products von Schritt 12 muss gelten, dass X - [HasCustomerGroupRelation] -> Product 14. Hinsichtlich der Products von Schritt 13 muss gelten, dass Machine - [NeededFor] -> X 15. Hinsichtlich der Machines von Schritt 14 muss gelten, dass X - [Causes] -> Document

	16. Hinsichtlich der Documents von Schritt 15 muss gelten, dass Employee - [OccursIn] -> X 17. Hinsichtlich der Employees von Schritt 16 muss gelten, dass X - [ResponsibleFor] -> Machine 18. Hinsichtlich der Machines von Schritt 17 muss gelten, dass X - [OccursIn] -> Document 19. Hinsichtlich der Documents von Schritt 18 muss gelten, dass X - [RelatesTo] -> ProcessStep 20. Hinsichtlich der ProcessSteps von Schritt 19 muss gelten, dass Employee - [IsImportantFor] -> X 21. Hinsichtlich der Employees von Schritt 20 muss gelten, dass X - [Created] -> Document 22. Hinsichtlich der Documents von Schritt 21 muss gelten, dass X - [ExpressesSentiment] -> Product 23. Hinsichtlich der Products von Schritt 22 muss gelten, dass X - [NeedsRelatedKnowledge] -> Product 24. Hinsichtlich der Products von Schritt 23 muss gelten, dass Document - [PartOf] -> X 25. Hinsichtlich der Documents von Schritt 24 muss gelten, dass Employee - [Caused] -> X 26. Hinsichtlich der Employees von Schritt 25 muss gelten, dass X - [IsExpertOn] -> Machine 27. Hinsichtlich der Machines von Schritt 26 muss gelten, dass Document - [Explains] -> X 28. Hinsichtlich der Documents von Schritt 27 muss gelten, dass Employee - [Caused] -> X 29. Hinsichtlich der Employees von Schritt 28 muss gelten, dass X - [HasSimilarity] -> Employee 30. Hinsichtlich der Employees von Schritt 29 muss gelten, dass X - [Created] -> Document	
Eingabeparameter	-	
Ausgabe	Liste mit gefilterten Employee-IDs	
Verarbeitung	1 - Link Store	Berechne alles in einer großen Anfrage

Tabelle 14: Use Case 12

Tabelle 14 stellt den Use Case dar und Tabelle 26 zeigt die gemessenen Zeiten für die einzelnen Verarbeitungsschritte. Die gemessenen Zeiten sind bereits bei kleinen Volumina inakzeptabel groß. Der interne Programmablauf ist denkbar einfach, besteht also nur aus einer großen SQL-Anfrage bzw. einer großen Cypher-Anfrage.

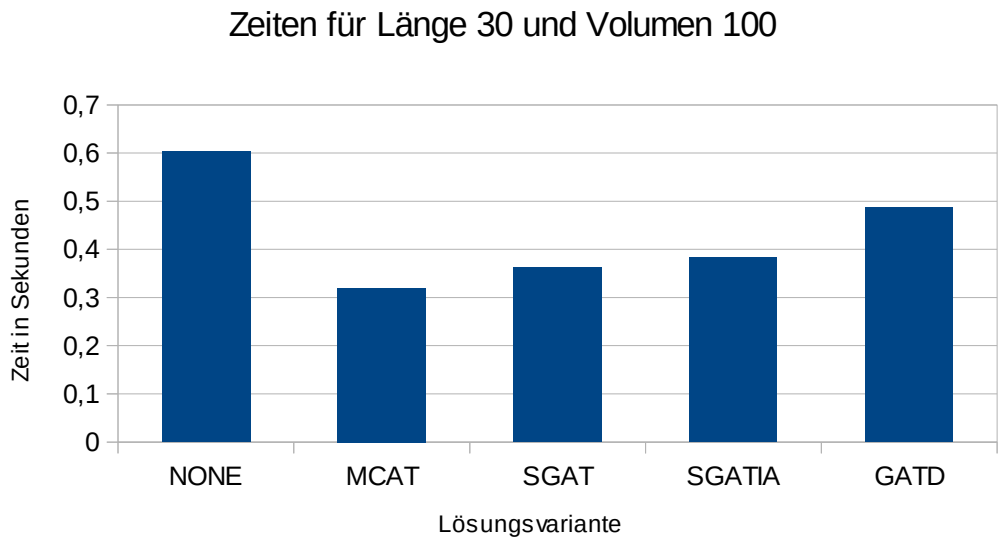

Abbildung 41: Gemessene Zeiten Use Case 12 für Länge 30 und Volumen 100

Interessant sind die starken Schwankungen innerhalb einer Volumenstufe für die relationalen Lösungsvarianten, wie in Abbildung 41 für die Anfragelänge 30 und ein Volumen von 100 dargestellt. Da diese Zeiten auch bei mehrmaligen Messungen auftraten, kann eine nichtdeterministische Ursache, wie das Einschalten des Garbage Collectors, ausgeschlossen werden. Die einzige verbleibende Ursache ist, dass ein ungünstiger Ausführungsplan für die Ausreißer nach oben verantwortlich ist. Da es sich bei den Einschränkungen nicht um echte Einschränkungen handelt, besteht kein äußerer Grund dafür, dass die Lösung ohne Einschränkungen schlechter als eine Lösung mit Einschränkungen abschneiden sollte. Wie in der Abbildung ersichtlich, dauert die Lösung ohne Einschränkung für Länge 30 ungefähr 0,6 Sekunden, während die MCAT-Lösung für Länge 30 lediglich 0,3 Sekunden benötigt. Derartige Beobachtungen legen die Vermutung nahe, dass DB2 bei zu vielen beteiligten Joins Probleme damit hat, den besten Ausführungsplan zu finden und manchmal auf suboptimale Ausführungspläne zurückgreift.

5.5 Diskussion der gemessenen Zeiten

Lediglich in den Use Cases 1, 5, 6, 7 und 12 werden die relationalen Lösungsvarianten unterschieden, da die anderen Use Cases, im Falle der beziehungsorientierten Anfragen, überhaupt nicht auf dem relationalen Link Store laufen, oder ansonsten ohne Ein-

schränkungen auf den Attributen formuliert sind. Die fünffache Durchführung der Messungen zeigte, dass die Ausführungszeit des Link Stores Schwankungen ausgesetzt ist, wobei die Schwankungen bei den sehr kleinen Ausführungszeiten prozentual größer als bei den großen Ausführungszeiten ausfallen. Durch die mehrmalige Ausführung wurde dieser Effekt etwas abgefedert, trotzdem müssen für Ausführungszeiten von ungefähr einer Sekunde ca. 10 Prozent Toleranz eingerechnet werden. Für Use Case 1 sind die gemessenen Zeiten von einigen Hunderstel-Sekunden zu gering um eine seriöse Aussage treffen zu können. Die Link Store-Zeiten von Use Case 5 und 6 liegen alle im Bereich einer natürlichen Schwankungsbreite. Sehr deutliche Unterschiede treten bei Use Case 7 auf, wo MCAT, für die höchste gemessene Volumenstufe, mit 757 Sekunden schneller als die anderen Varianten im Bereich von 1200 Sekunden abschneidet. Ein anderes Bild ergibt sich für Use Case 12, wo SGAT und SGATIA die besten Ergebnisse lieferten. Allerdings müssen die gemessenen Zeiten für Use Case 12 vor dem Hintergrund schlechter Ausführungspläne interpretiert werden, sodass dieses Ergebnis nicht repräsentativ sein sollte. Auch einen Performance-Tip für MCAT, aufgrund der Zeiten von Use Case 7, zu geben, ist schwierig. MCAT schnitt hier zwar noch am wenigsten schlecht ab, aber dies deckt sich nicht mit den Messungen für die anderen Use Cases.

Wie gezeigt schneidet die relationale Umsetzung, mit Ausnahme für Use Case 7, besser als die graphbasierte Umsetzung ab und Neo4j kann, je nach Anfrage, für höhere Volumina komplett einbrechen. Dieses Ergebnis widerspricht zwar dem Selbstverständnis von Neo4j , aber nicht unbedingt den Erfahrungen anderer Anwender [41]. Gemäß Vicknair et al. sollte Neo4j bei Verwendung der Java-API deutlich besser als bei Verwendung von Cypher abschneiden [15]. Da die schlechte Neo4j-Performance vermutlich zu einem nicht geringen Teil auf schlechte Ausführungspläne zurückging, erscheint diese Annahme realistisch. Woran Neo4j auch oft gescheitert ist, war eine zu lange Liste an Parametern. Bezüglich der schlechten Ausführungspläne wurde beispielsweise in Listing 13 gezeigt, wie sich Cypher-Anfragen für bessere Ausführungspläne umschreiben lassen. Anhand dieser einfachen Optimierung wird ersichtlich, dass Cypher nicht wie SQL verwendet werden darf, wo die Anfrage einfach so geschrieben werden kann, wie der Nutzer sich den Sachverhalt vorstellt. Für performante Cypher-

Anfragen muss dahingegen der erwünschte Ausführungsplan beachtet werden. Es ist nicht der Fall, dass Neo4j seit den paar Jahren des Bestehens bereits eine Technologiereife wie die relationalen DBMS aufweist. Normale Anwender mit durchschnittlichen Sprachkenntnissen werden sich mit SQL weit einfacher als mit Cypher tun. Warum Neo4j ausgerechnet bei Use Case 7 besser als die relationalen Lösungsvarianten abschnitt, ist schwierig zu begründen. Das bessere Laufzeitverhalten von Neo4j war hier jedenfalls nur durch eine Umschreibung der Cypher-Anfragen möglich.

Wie insbesondere das Laufzeitverhalten für Use Case 6, beschrieben in 5.4.6, zeigt, führt die Indirektion zwischen Element.LEID und dem Primärschlüssel des zugehörigen Data Warehouse-Elements auch für die separaten Anfragen zu großen Laufzeiteinbußen, welche über die Hälfte der Gesamtlaufzeit ausmachen können. Es gibt mehrere Lösungsmöglichkeiten das Schema hier anzupassen und die simpelste Möglichkeit wäre einfach, die Indirektion zu entfernen, also in Element.LEID den Primärschlüssel des Data Warehouse-Elements reinzuschreiben. Auf unterster Ebene sollte im Hinblick auf Performance designt werden, ohne das Designentscheidungen auf der konzeptionellen Ebene hier die Vorgaben treffen.

Ein Problem stellten die langen IN-Listen in den SQL-Anfragen bzw. das Äquivalent langer Funktionsparameterlisten in den Cypher-Anfragen dar. Für SQL war die Performance hier zwar noch beherrschbar, aber für anders gestellte Anfragen bzw. noch größere Volumina, welche zu noch längeren IN-Listen führen, wird die Performance noch weiter rapide absinken. Durch die Verwendung der verschmolzenen Anfragen kann dieses Problem in Bezug auf lange IN-Listen für Data Warehouse-Elemente umgangen werden, aber die passenden CMS-Elemente müssen weiterhin über solch eine IN-Liste übergeben werden. Eine Lösungsmöglichkeit für dieses Problem wäre, selbständig eine voll indizierte Tabelle zu erstellen und jene Tabelle während der Abfragen mit den zutreffenden Werten der IN-Liste zu befüllen, um dann, anstelle der Übergabe der IN-Liste, mit dieser Tabelle zu joinen.

6 Fazit und Ausblick

Da, mit Ausnahme von Use Case 12, für sämtliche Use Cases die Schwankungsbreite mit maximal 50% für Use Case 7 überschaubar war, bleibt festzuhalten, dass sämtliche relationalen Lösungsvarianten ein ähnliches Performanceverhalten aufweisen. Von daher sollte die Lösungsauswahl hier nicht anhand der Performanceeigenschaften, sondern aufgrund der anderen Kriterien von Abschnitt 4.2 erfolgen. Ein wichtiges Kriterium ist die Schemaflexibilität welches MCAT zu der am wenigsten zu bevorzugenden Variante macht. Die simpelste der verbleibenden drei Varianten ist SGAT. Hier müssen, im Gegensatz zu den anderen Varianten, zwar Casts stattfinden, aber jene machen keinen großen Teil der Gesamtlaufzeit aus. Darum soll an dieser Stelle eine Empfehlung für die SGAT-Variante ausgesprochen werden.

Daran, dass, trotz eines gewissen Aufwands für Optimierungen, nicht sämtliche Messergebnisse befriedigende Resultate aufwiesen, wird ersichtlich, dass die eingeschlagene Vorgehensweise für die Implementierung der Verlinkungen für die höchsten Volumina und die komplexesten Anfragen bereits die Grenze des Machbaren erreichte. Da für einfach gestrickte Anfragen, auch bei größeren Volumina, die Link Store-Performance unproblematisch war, deutet dies darauf hin, dass das Anwendungsgebiet des Link Stores sich auf Anfragen normaler Komplexität beziehen sollte. Komplexere Anfragen, wo fünf oder mehr Entitäten zueinander in Beziehung gesetzt werden, sind für den menschlichen Nutzer auch sehr schwer zu durchschauen, sodass Anfragen normaler Komplexität der Realität näher kommen sollten.

Nachdem in dieser Arbeit die Machbarkeit der Link Store-Umsetzung, unter den herausgearbeiteten Einschränkungen, gezeigt wurde, wäre der nächste Schnitt die Entwicklung einer Anfrageschnittstelle, an welche der Nutzer seine Anfragen stellen kann, statt der hartcodiert implementierten Use Cases.

Literaturverzeichnis

[1] C. Gröger, F. Niedermann, H. Schwarz und B. Mitschang. "Supporting Manufacturing Design by Analytics. Continuous Collaborative Process Improvement enabled by the Advanced Manufacturing Analytics Platform" in Proceedings of the 2012 16th IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD), May 23-25, 2012, Wuhan, China, 2012.

[2] C. Gröger, H. Schwarz und B. Mitschang. "The Manufacturing Knowledge Repository. Consolidating Knowledge to Enable Holistic Process Knowledge Management in Manufacturing" in Proceedings of the 16th International Conference on Enterprise Information Systems (ICEIS), 27-30 April, 2014, Lisbon, Portugal, 2014.

[3] C. Gröger, H. Schwarz und B. Mitschang. "The Deep Data Warehouse: Link based Integration and Enrichment of Warehouse Data and Unstructured Content" in Proceedings of the 18th IEEE International Enterprise Distributed Object Computing Conference (EDOC), 01-05 September, 2014, Ulm, Germany, 2014.

[4] P. Stuhlmüller. "Integration strukturierter und unstrukturierter Daten zur Prozessoptimierung", Institut für Parallele und Verteilte Systeme - Abteilung Anwendersoftware, Universität Stuttgart, 2013.

[5] C. Gröger, F. Niedermann und B. Mitschang. "Data Mining-driven Manufacturing Process Optimization" in Ao, S. I. (Hrsg); Gelman, L. (Hrsg); Hukins, D. W. L. (Hrsg); Hunter, A. (Hrsg); Korsunsky, A. M. (Hrsg): Proceedings of the World Congress on Engineering 2012 Vol III, WCE 2012, 4 – 6 July, 2012, London, U.K, 2012.

[6] C. Gröger und C. Stach. "The Mobile Manufacturing Dashboard" in Proceedings of the 2014 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 24-28 March, 2014, Budapest, Hungary, 2014.

[7] K. Kruczynski. "DW 2.0 - das neue Konzept für Data Warehousing", abgerufen am 28.8.2014, http://www.wi.htwk-leipzig.de/fileadmin/fbwiwi/Wirtschaftsinfo/Publikationen/2006/DW2.0_Architektur.pdf.

[8] S. Busse, R. Kutsche, U. Lesser und Herbert Weber. " Federated Information Systems: Concepts, Terminology and Architectures", Fachbereich 13 Informatik Computergestützte InformationssystemeCIS , Technische Universität Berlin, 1999.

[9] C. Goble und R. Stevens. "State of the nation in data integration for bioinformatics" in Journal of Biomedical Informatics, vol. 41 (5), pp. 687-693, 2008.

[10] E. Rahm, A. Thor, D. Aumueller, H. Do, N. Golovin und T. Kirsten. "iFuice – Information Fusion utilizing Instance Correspondences and Peer Mappings " in Proc. 8th Intl. Workshop on the Web and Databases (WebDB), 2005.

[11] C. Cargill. "Standards & Technology A Look Behind the Scenes Creating and Implementing Database Standards", abgerufen am 28.8.2014, http://www.uniforum.org/publications/ufm/may96/standards.html.

[12] R. Angles. "A Comparison of Current Graph Database Models" in Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on, 2012.

[13] M. Chapple. "Abandoning ACID in Favor of BASE", abgerufen am 18.8.2014, http://databases.about.com/od/otherdatabases/a/Abandoning-Acid-In-Favor-Of-Base.htm.

[14] Neo4j. "Chapter 16. Transaction Management", abgerufen am 18.8.2014, http://docs.neo4j.org/chunked/stable/transactions.html.

[15] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen und D. Wilkins. "A Comparison of a Graph Database and a Relational Database" in ProceedingACM SE '10 Proceedings of the 48th Annual Southeast Regional Conference Article No. 42, 2010.

[16] Neo4j. "New: Labels and Indexes", abgerufen am 28.8.2014, http://www.neo4j.org/develop/labels.

[17] R. Melworm. "The History of Database", abgerufen am 18.8.2014, http://www.kean.edu/~rmelworm/3040-00/LuoDatabaseTimeLine.html.

[18] Neo4j. "The top 10 ways to get to know Neo4j", abgerufen am 28.8.2014, http://neo4j.com/blog/the-top-10-ways-to-get-to-know-neo4j/.

[19] K. Haugen. "A Brief History of NoSQL", abgerufen am 28.8.2014, http://blog.knuthaugen.no/2010/03/a-brief-history-of-nosql.html.

[20] Neo4j. "Release Notes: Neo4j 2.0", abgerufen am 28.8.2014, http://neo4j.com/release-notes/neo4j-2-0/.

[21] J. Gao, R. Jin, J. Zhou, J. Xu Yu, X. Jiang und T. Wang. "Relational Approach for Shortest Path Discovery over Large Graphs" in Proceedings of the VLDB Endowment, Volume 5 Issue 4, Pages 358-369, 2011.

[22] Neo4j. "Chapter 25. Security", abgerufen am 28.8.2014, http://docs.neo4j.org/chunked/stable/operations-security.html.

[23] M. Rudolf, M. Paradies, C. Bornhövd und W. Lehner. "The Graph Story of the SAP HANA Database" in BTW'13, 15. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web, March 11 - March 15, Magdeburg, 2013.

[24] R. Angles und C. Gutierrez. "Survey of graph database models", ACM Computing Surveys, vol. 40 (1), pp. 1–39, 2008.

[25] M. Rodriguez und P. Neubauer. "Constructions from Dots and Lines", Bulletin of the American Society for Information Science and Technology, vol. 36 (6), pp. 35–41, 2010.

[26] J. Tauberer. "What is RDF and what is it good for?", abgerufen am 28.8.2014, https://github.com/JoshData/rdfabout/blob/gh-pages/intro-to-rdf.md.

[27] D. Srinivas. "The case for storing RDF in a relational database", abgerufen am 28.8.2014, http://files.meetup.com/274991/NYSemanticWebDB2RDF.pdf.

[28] M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea und B. Bhattacharjee. "Building an Efficient RDF Store Over a Relational Database" in Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, Pages 121-132, 2013.

[29] R. Agrawal, A. Somani und Y. Xu "Storage and Querying of E-Commerce Data" in Proceedings of the 27th International Conference on Very Large Data Bases, Pages 149-158, 2001.

[30] C. Gröger. "Konzeption eines relationalen Link Store", unveröffentlichter technischer Bericht, Universität Stuttgart, 2014.

[31] J. Seifert. "Shrink your DB2 for Linux, UNIX, and Windows database using value compression", IBM, 2008, http://www.ibm.com/developerworks/data/library/techarticle/dm-0806seifert/dm-0806seifert-pdf.pdf.

[32] o.A. "Data types that map to SQL data types in JDBC applications", abgerufen am 29.8.2014, http://poincare.matf.bg.ac.rs/~aspasic//pbp/vezbe/SQLJ%20host%20variable%20types.pdf.

[33] IBM. "INTEGER or INT", abgerufen am 29.8.2014, http://www-01.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/com.ibm.db2z10.doc.sqlref/src/tpc/db2z_bif_integer.dita.

[34] M. Winand. "Online-Buch zu indexbasiertem SQL Tuning", abgerufen am 29.8.2014, http://use-the-index-luke.com/de.

[35] S. Childers. "Project Packaging: by-layer vs. by-feature", abgerufen am 29.8.2014, http://shaunchilders.com/node/15.

[36] Oracle. "Random", abgerufen am 29.8.2014, http://docs.oracle.com/javase/6/docs/api/java/util/Random.html.

[37] B. Venners. "Bytecode basics A first look at the bytecodes of the Java virtual machine", abgerufen am 29.8.2014, http://www.javaworld.com/article/2077233/core-java/bytecode-basics.html.

[38] F. Holzschuher und R. Peinl. "Performance of Graph Query Languages - Comparison of Cypher, Gremlin and Native Access in Neo4j" in Proceedings of the Joint EDBT/ICDT 2013 WorkshopsPages 195-204, 2013.

[39] E. Crooks. "Fun with IN-lists and GENROW", abgerufen am 3.9.2014, http://db2commerce.com/2010/06/03/fun-with-in-lists-and-genrow/.

[40] Microsoft. "Performance data from Average Disk sec/Read counter of the PhysicalDisk performance object", abgerufen am 13.8.2014, http://technet.microsoft.com/en-us/library/aa996220(v=exchg.80).aspx.

[41] J. Baach. "neo4j performance compared to mysql", aberufen am 2.9.2014, https://baach.de/Members/jhb/neo4j-performance-compared-to-mysql.

Anhang A - Tabellarisches Verzeichnis gemessener Zeiten

Im Folgenden werden die während der Durchführung der Use Cases gemessenen Zeiten tabellarisch protokolliert. Die hier erwähnten Schritte entsprechen dabei den Schritten, welche in Kapitel 5.4 für den internen Programmablauf der Use Cases dokumentiert wurden. Die Volumen-Angabe bezieht sich auf die Anzahl der Links pro Linktyp. Wie hierbei die Element-Volumina und Link-Attribut-Volumina sind, ergibt sich aus Tabelle 2. Sämtliche Zeitangaben erfolgen in Sekunden und auf zwei Nachkomastellen gerundet.

Volumen	Schritt	Graph	MCAT	SGAT	SGATIA	GATD
50	1 DW	0,04	0,04	0,04	0,04	0,04
	2 Link Store	0	0,04	0,04	0,04	0,05
	3 Link Store		0	0,06	0,07	0,03
	4 CMS	0,48	0,45	0,45	0,48	0,45
	5 CMS	0	0	0	0	0
	Java	0,01	0	0	0,01	0
	Gesamt	0,52	0,53	0,59	0,64	0,58
100	1 DW	0,04	0,04	0,04	0,04	0,04
	2 Link Store	0	0,04	0,04	0,04	0,03
	3 Link Store		0	0	0,06	0,04
	4 CMS	0,43	0,41	0,44	0,41	0,43
	5 CMS	0	0	0	0	0
	Java	0	0	0,01	0	0
	Gesamt	0,48	0,49	0,52	0,56	0,54
1000	1 DW	0,04	0,04	0,04	0,04	0,04
	2 Link Store	0	0,04	0,04	0,04	0,04
	3 Link Store		0	0	0	0
	4 CMS	0,48	0,49	0,47	0,45	0,49
	5 CMS	0	0	0	0	0
	Java	0,01	0,01	0,01	0	0,01
	Gesamt	0,52	0,57	0,56	0,54	0,58
10000	1 DW	0,04	0,04	0,04	0,04	0,04
	2 Link Store	0,01	0,04	0,04	0,04	0,05
	3 Link Store		0	0	0	0
	4 CMS	0,5	0,51	0,48	0,52	0,51
	5 CMS	0	0	0	0	0
	Java	0	0	0,01	0,01	0
	Gesamt	0,54	0,59	0,56	0,61	0,6
100000	1 DW	0,04	0,04	0,04	0,04	0,04
	2 Link Store	0	0,04	0,04	0,04	0,04
	3 Link Store		0	0	0	0
	4 CMS	0,5	0,51	0,5	0,49	0,5
	5 CMS	0	0	0	0	0
	Java	0	0,01	0	0,01	0,01
	Gesamt	0,54	0,6	0,58	0,58	0,58
200000	1 DW	0,04	0,04	0,04	0,04	0,04
	2 Link Store	0	0	0,04	0,04	0,04
	3 Link Store		0	0	0	0
	4 CMS	0,42	0,42	0,44	0,44	0,44

5 CMS	0	0	0	0	0
Java	0,01	0,01	0	0,01	0,01
Gesamt	0,46	0,53	0,53	0,53	0,53

Tabelle 15: Gemessene Zeiten Use Case 1

Volumen	Schritt	
beliebig	1 DW	0,28
	Gesamt	0,28

Tabelle 16: Gemessene Zeiten Use Case 2

Volumen	Schritt	Graph	Relational
50	1 CMS	2,57	2,73
	2 CMS	0,03	0,02
	3 Link Store	0	0,04
	4 DW	0,04	0,04
	Java	0,01	0
	Gesamt	2,64	2,84
100	1 CMS	2,57	2,57
	2 CMS	0,02	0,02
	3 Link Store	0	0,04
	4 DW	0,04	0,04
	Java	0	0
	Gesamt	2,64	2,67
1000	1 CMS	2,62	2,63
	2 CMS	0,02	0,02
	3 Link Store	0	0,04
	4 DW	0,04	0,04
	Java	0	0,01
	Gesamt	2,68	2,74
10000	1 CMS	2,64	2,69
	2 CMS	0,03	0,02
	3 Link Store	0,04	0,06
	4 DW	0,04	0,05
	Java	0	0
	Gesamt	2,76	2,82
100000	1 CMS	2,65	2,64
	2 CMS	0,02	0,02
	3 Link Store	0,17	0,06
	4 DW	0,08	0,07
	Java	0,01	0
	Gesamt	2,92	2,8
200000	1 CMS	2,59	2,7
	2 CMS	0,02	0,02
	3 Link Store	0,19	0,09
	4 DW	0,1	0,09
	Java	0,01	0,02
	Gesamt	2,91	2,93

Tabelle 17: Gemessene Zeiten Use Case 3

Volumen	Schritt	Graph	Relational
50	1 CMS	2,66	2,74
	2 Link Store	0,02	0,1
	3 DW	0,04	0,04
	Java	0	0
	Gesamt	2,73	2,89
100	1 CMS	2,6	2,63
	2 Link Store	0,04	0,12
	3 DW	0,04	0,04
	Java	0,01	0
	Gesamt	2,69	2,79
1000	1 CMS	2,61	2,71
	2 Link Store	0,17	0,06
	3 DW	0,05	0,06
	Java	0	0,01
	Gesamt	2,83	2,84
10000	1 CMS	2,67	2,66
	2 Link Store	0,12	0,09
	3 DW	0,09	0,13
	Java	0,16	0,02
	Gesamt	3,03	2,9
100000	1 CMS	2,66	2,64
	2 Link Store	0,55	0,37
	3 DW	0,37	0,85
	Java	24,64	1,96
	Gesamt	28,22	5,83
200000	1 CMS	2,57	2,6
	2 Link Store	1,08	0,64
	3 DW	0,67	1,88
	Java	107,43	9,14
	Gesamt	111,75	14,26

Tabelle 18: Gemessene Zeiten Use Case 4

Volumen	Schritt	Graph	MCAT	SGAT	SGATIA	GATD
50	1 Link Store	0,05	0,09	0,11	0,11	0,1
	2 DW	0,04	0,04	0,04	0,04	0,04
	Java	0	0	0	0	0
	Gesamt	0,09	0,13	0,15	0,15	0,14
100	1 Link Store	0,07	0,1	0,13	0,09	0,1
	2 DW	0,04	0,04	0,04	0,04	0,04
	Java	0	0	0	0	0
	Gesamt	0,11	0,14	0,17	0,13	0,15
1000	1 Link Store	0,22	0,08	0,1	0,16	0,11
	2 DW	0,05	0,05	0,05	0,06	0,05
	Java	0	0	0	0	0
	Gesamt	0,27	0,14	0,16	0,22	0,16
10000	1 Link Store	0,15	0,28	0,21	0,19	0,19
	2 DW	0,47	0,46	0,43	0,44	0,47
	Java	0,01	0	0	0,01	0,01
	Gesamt	0,64	0,74	0,64	0,64	0,66
100000	1 Link Store	1,41	0,84	0,82	0,66	0,68
	2 DW	1,29	1,29	1,27	1,28	1,26
	Java	0,01	0,01	0,01	0,01	0,01
	Gesamt	2,71	2,15	2,11	1,95	1,95
200000	1 Link Store	2,58	1,37	1,35	1,17	1,16

	2 DW	2,24	2,17	2,15	2,13	2,22
	Java	0,02	0,03	0,03	0,03	0,03
	Gesamt	4,84	3,57	3,53	3,33	3,41

Tabelle 19: Gemessene Zeiten Use Case 5

Volumen	Schritt	Graph	MCAT	SGAT	SGATIA	GATD
50	1 DW	0,31	0,3	0,29	0,31	0,3
	2 Link Store	0,03	0,12	0,11	0,1	0,1
	3 CMS	0,46	0,46	0,46	0,46	0,45
	4 Link Store	0	0	0	0	0
	5 DW	0	0	0	0	0
	Java	0,01	0,01	0,01	0,01	0,01
	Gesamt	0,81	0,89	0,87	0,88	0,87
100	1 DW	0,31	0,3	0,3	0,29	0,3
	2 Link Store	0,05	0,1	0,09	0,09	0,09
	3 CMS	0,52	0,52	0,52	0,49	0,49
	4 Link Store	0	0,01	0,01	0,01	0
	5 DW	0	0	0	0	0
	Java	0,01	0	0,01	0,01	0,01
	Gesamt	0,89	0,92	0,93	0,88	0,89
1000	1 DW	0,3	0,29	0,3	0,34	0,29
	2 Link Store	0,25	0,11	0,12	0,13	0,11
	3 CMS	0,51	0,53	0,51	0,49	0,5
	4 Link Store	0,01	0	0	0	0
	5 DW	0,01	0,01	0,01	0,01	0
	Java	0	0,01	0	0	0,01
	Gesamt	1,07	0,94	0,94	0,97	0,91
10000	1 DW	0,3	0,27	0,26	0,31	0,29
	2 Link Store	0,14	0,14	0,15	0,14	0,14
	3 CMS	0,5	0,52	0,5	0,49	0,52
	4 Link Store	0,03	0,02	0,02	0,02	0,02
	5 DW	0,02	0,02	0,02	0,02	0,02
	Java	0	0	0,01	0,01	0
	Gesamt	1	0,98	0,96	1	1
100000	1 DW	0,3	0,28	0,27	0,29	0,28
	2 Link Store	1,38	1,12	1,09	1,12	1,08
	3 CMS	2,08	1,87	1,89	1,92	1,9
	4 Link Store	0,28	0,2	0,21	0,26	0,19
	5 DW	0,27	0,26	0,25	0,23	0,24
	Java	1,2	0,91	0,91	0,9	0,89
	Gesamt	5,5	4,63	4,61	4,73	4,59
200000	1 DW	0,31	0,32	0,28	0,29	0,27
	2 Link Store	2,48	1,97	2,05	2,02	2,06
	3 CMS	2,75	2,91	2,75	2,79	2,78
	4 Link Store	0,87	0,5	0,52	0,48	0,46
	5 DW	0,26	0,25	0,27	0,22	0,25
	Java	8,75	9,24	10,11	9,36	10,33
	Gesamt	15,41	15,08	15,98	15,16	16,15

Tabelle 20: Gemessene Zeiten Use Case 6

Volumen	Schritt	Graph	MCAT	SGAT	SGATIA	GATD
50	1 CMS	2,66	2,65	2,69	2,69	2,63
	2 CMS	2,26	2,24	2,11	2,26	2,12

	3 CMS	2,27	2,2	2,22	2,26	2,13
	4 DW	0,32	0,31	0,33	0,31	0,31
	5 Link Store	0,11	0,09	0,1	0,11	0,1
	6 CMS	0,02	0,02	0,02	0,02	0,02
	Java	0	0,01	0,01	0	0,01
	Gesamt	7,64	7,52	7,47	7,66	7,32
100	1 CMS	2,65	2,63	2,55	2,61	2,6
	2 CMS	2,17	2,22	2,21	2,25	2,16
	3 CMS	2,26	2,28	2,21	2,21	2,21
	4 DW	0,3	0,31	0,34	0,34	0,29
	5 Link Store	0,08	0,1	0,1	0,14	0,09
	Java	0	0	0	0	0
	6 CMS	0,01	0,01	0	0	0,01
	Gesamt	7,47	7,56	7,42	7,55	7,35
1000	1 CMS	2,64	2,68	2,64	2,64	2,61
	2 CMS	2,21	2,23	2,27	2,18	2,32
	3 CMS	2,28	2,27	2,29	2,27	2,27
	4 DW	0,31	0,3	0,32	0,29	0,31
	5 Link Store	0,13	0,24	0,18	0,18	0,16
	6 CMS	0,02	0,02	0,02	0,02	0,02
	Java	0	0	0	0,01	0,01
	Gesamt	7,59	7,74	7,73	7,59	7,7
10000	1 CMS	2,65	2,64	2,61	2,59	2,68
	2 CMS	2,25	2,24	2,27	2,2	2,19
	3 CMS	2,26	2,22	2,27	2,26	2,25
	4 DW	0,34	0,31	0,32	0,3	0,3
	5 Link Store	0,6	5,47	7,94	7,63	5,07
	6 CMS	0,02	0,03	0,02	0,03	0,03
	Java	0	0,01	0,01	0,01	0,01
	Gesamt	8,12	12,92	15,44	15,02	12,52
100000	1 CMS	3,43	3,53	2,69	2,64	2,62
	2 CMS	2,23	2,2	2,27	2,32	2,16
	3 CMS	2,29	2,3	2,24	2,3	2,35
	4 DW	1	1,35	0,38	0,33	0,4
	5 Link Store	482,27	757,07	1318,47	1257,76	1184,09
	6 CMS	0,39	0,37	0,37	0,37	0,37
	Java	0,01	0,01	0,01	0	0,02
	Gesamt	491,62	766,83	1326,43	1265,72	1192,01
200000	1 CMS	-	-	-	-	-
	2 CMS	-	-	-	-	-
	3 CMS	-	-	-	-	-
	4 DW	-	-	-	-	-
	5 Link Store	-	-	-	-	-
	6 CMS	-	-	-	-	-
	Java	-	-	-	-	-
	Gesamt	-	-	-	-	-

Tabelle 21: Gemessene Zeiten Use Case 7

Volumen	Schritt	Graph	Relational
50	1 Link Store	0	0,07
	2 CMS	1,05	1,03
	3 CMS	2,62	2,57
	4 DW	0,04	0,05
	5 Link Store	0,01	0,04
	6 DW	0	0

	Java	0,01	0,01
	Gesamt	3,73	3,77
100	1 Link Store	0,01	0,08
	2 CMS	1,68	1,65
	3 CMS	2,63	2,66
	4 DW	0,04	0,04
	5 Link Store	0,02	0,04
	6 DW	0	0,01
	Java	0	0
	Gesamt	4,38	4,48
1000	1 Link Store	0,07	0,06
	2 CMS	12,88	12,81
	3 CMS	2,63	2,62
	4 DW	0,04	0,04
	5 Link Store	0,04	0,04
	6 DW	0	0
	Java	0,01	0,01
	Gesamt	15,68	15,59
10000	1 Link Store	0,08	0,09
	2 CMS	80,46	80,91
	3 CMS	2,65	2,55
	4 DW	0,05	0,04
	5 Link Store	0,16	0,24
	6 DW	0,02	0,1
	Java	0	0
	Gesamt	83,42	83,85
100000	1 Link Store	0,07	0,11
	2 CMS	81,45	81,33
	3 CMS	2,57	2,5
	4 DW	0,04	0,04
	5 Link Store	81,57	4,43
	6 DW	0,04	0,02
	Java	0,01	0,02
	Gesamt	165,75	88,44
200000	1 Link Store	-	0,11
	2 CMS	-	81,11
	3 CMS	-	2,51
	4 DW	-	0,05
	5 Link Store	-	62,25
	6 DW	-	0,07
	Java	-	0,09
	Gesamt	-	149,19

Tabelle 22: Gemessene Zeiten Use Case 8

Volumen	Schritt	Graph
50	1 Link Store	0
100	1 Link Store	0
1000	1 Link Store	0
10000	1 Link Store	0,03
100000	1 Link Store	0,02
200000	1 Link Store	0,02

Tabelle 23: Gemessene Zeiten Use Case 9

Volumen	Schritt	Graph
50	1 Link Store	0
100	1 Link Store	0,01
1000	1 Link Store	0,03
10000	1 Link Store	0,03
100000	1 Link Store	0,28
200000	1 Link Store	0,48

Tabelle 24: Gemessene Zeiten Use Case 10

Volumen	Schritt	Graph
50	1 Link Store	1,53
100	1 Link Store	1,65
1000	1 Link Store	90,96
10000	1 Link Store	-
100000	1 Link Store	-
200000	1 Link Store	-

Tabelle 25: Gemessene Zeiten Use Case 11

		Graph						Relational														
Volumen	Schritt	No restrictions			Restrictions			No restrictions			MCAT			SGAT			SGATIA			GATD		
		10	20	30	10	20	30	10	20	30	10	20	30	10	20	30	10	20	30	10	20	30
50	1	0,36	2,7	-	0,45	8,51	-	0,15	0,25	0,35	0,09	0,12	0,14	0,16	0,2	0,33	0,18	0,29	0,3	0,17	0,23	0,35
100	1	0,41	17,71	-	0,58	52,67	-	0,2	0,3	0,6	23,94	0,17	0,32	0,14	0,3	0,36	0,16	0,27	0,38	0,17	0,3	0,49
1000	1	532,75	-	-	-	-	-	-	-	-	-	-	-	93,8	217,03	-	82	200,83	-	-	-	-
10000	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
100000	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
200000	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Tabelle 26: Gemessene Zeiten Use Case 12

Erklärung

Ich versichere, diese Arbeit selbständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

_____

Ort, Datum, Unterschrift