JAVACREAM

Training
Consulting
Projectmanagement

Neo4J

Vorstellungsrunde

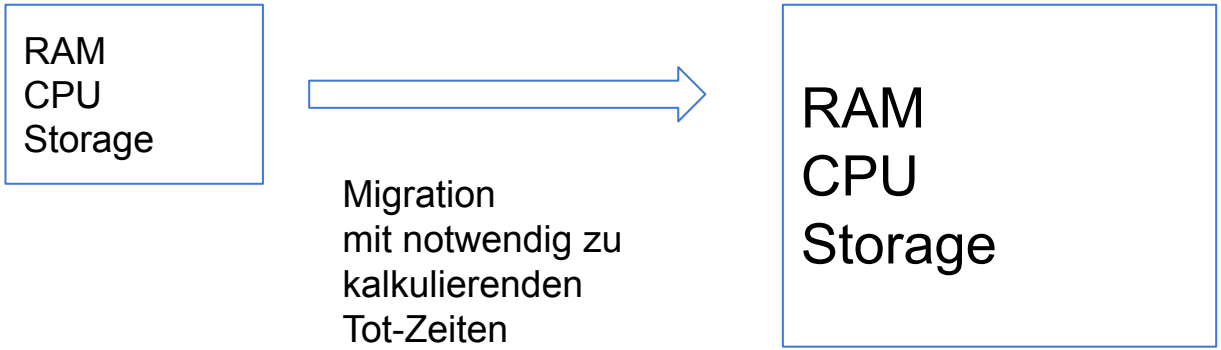
JAVACREAM
Training
Consulting
Projectmanagement

- Name, Rolle im Unternehmen
- Konkrete Problemstellung
- Themenbezogene Vorkenntnisse
- Konkrete individuelle Zielsetzung

Ausgangssituation

Ursprüngliche Problemstellung

- Big&Fast Data
  - 2006: Die Grenze der Ablage von Daten in relationalen Datenbanksystemen war erreicht
    - Storage kann nicht beliebig erhöht werden, weil relationale DBMS vertikal skalieren



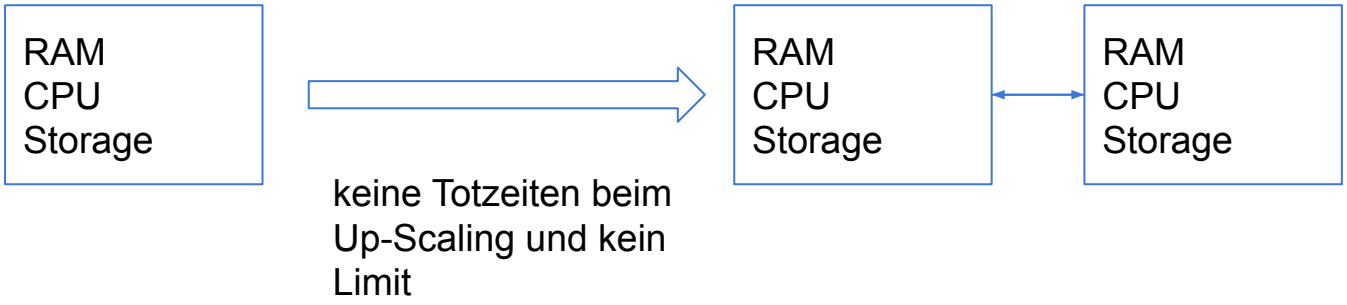
    - Daten-Analyse übersteigt durch die Komplexität der Daten-Zusammenhänge CPU- und RAM-Limits

NoSQL-Bewegung

- NoSQL als Begriff ist eher unglücklich gewählt
  - Besser: NoRelational
  - “No” ist nicht eine Ablehnung sondern eine Abkürzung für “not only”
- Bei der Umsetzung eines Entity-Modells sollen auch alternative Modelle gleichberechtigt in Erwägung gezogen werden
- https://java.integrata-cegos.de/nosql-eine-einfuehrung/

Konkrete Ideen

- “Wie modelliere ich meine Datenhaltung, um diese in einem horizontal skalierenden System halten zu können?”



- **Key-Value-Store**
  - select value from store where key = aKey
- **Column-oriented Databases**
  - Das sind die eigentlichen Big Data-Datenbanken

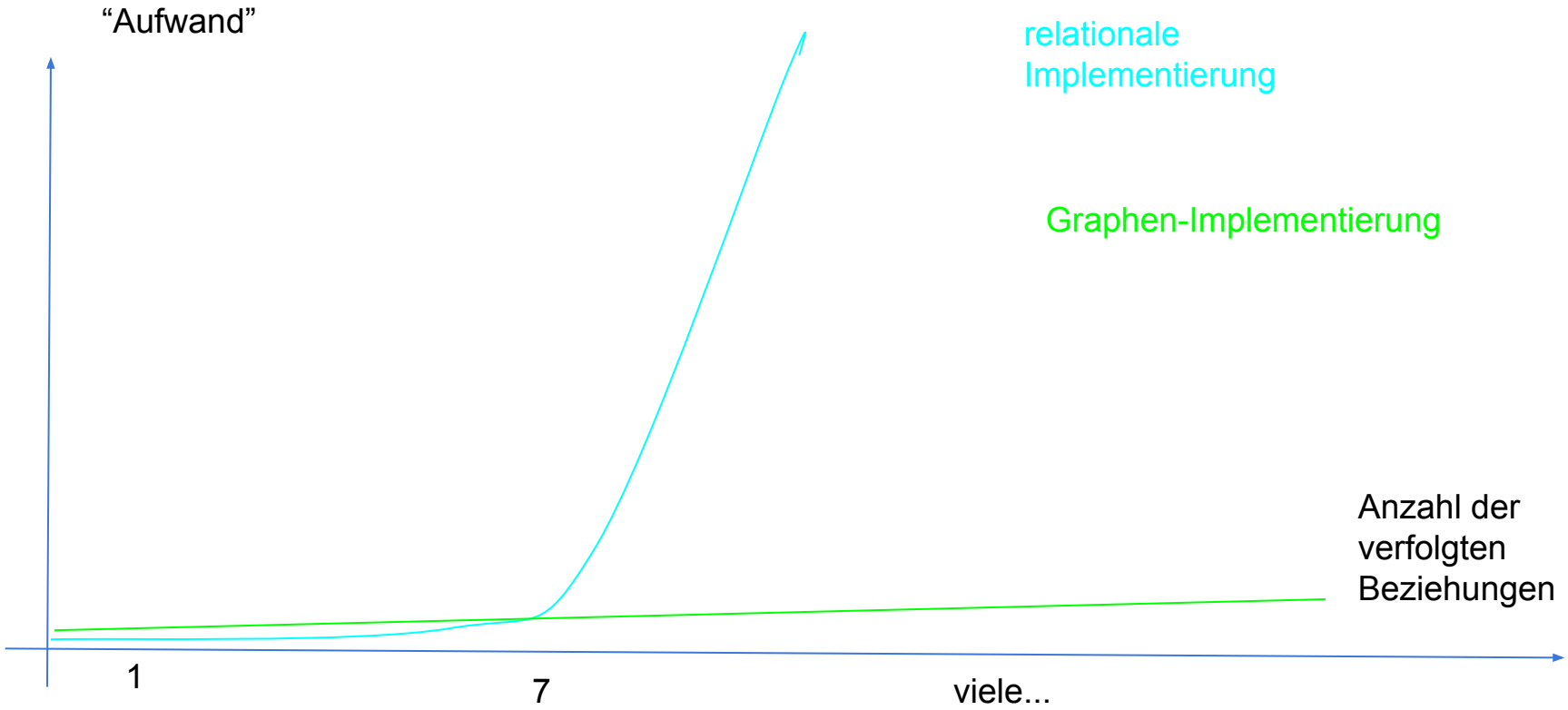
Konkrete Ideen

- “Wie können komplexe Analysen formuliert und effizient durchgeführt werden?”
  - Problematik des “Join”: Relationale Datenbank-Systeme können nur eine beschränkte Anzahl von Joins durchgeführen (etwa 7)
- Lösung
  - **Dokumenten-orientierten Datenbanken**
    - Dokumente enthalten alle zugehörigen Daten “en block” und Beziehungen zwischen Dokumenten sind Links
      - Die Rolle der Client-Anwendung ist hier deutlich höher als bei einem relationalen Client, der seine Daten komplett aufbereitet von der Datenbank bekommt
    - Neue Beziehungen können problemlos eingeführt werden
      -

Konkrete Idee

- Beziehungen und deren Analyse sollen von einem Datenbank-System durchgeführt werden
  - Vorsicht: Very Big Data ist hier definitiv nicht der Fokus
- Ein Knoten, “eine Node” enthält alle Daten, die für ihn relevant sind
  - Damit entspricht ein Knoten einem einzelnen Dokument
- Beziehungen, “eine Relation” sind vollständige Dokumente, die jedoch eine Quelle und ein Target besitzen
  - Eine Relation verbindet zwei Knoten miteinander
    - Relationen können nicht auf andere Relationen verbinden
- Mit Nodes und Relations kann die interne Datenhaltung vollkommen anders implementiert und optimiert werden
  - Graphen-orientierte Datenbank

Vergleich: Auflösen von Beziehungen



Warum sind die relationalen “so schlecht”

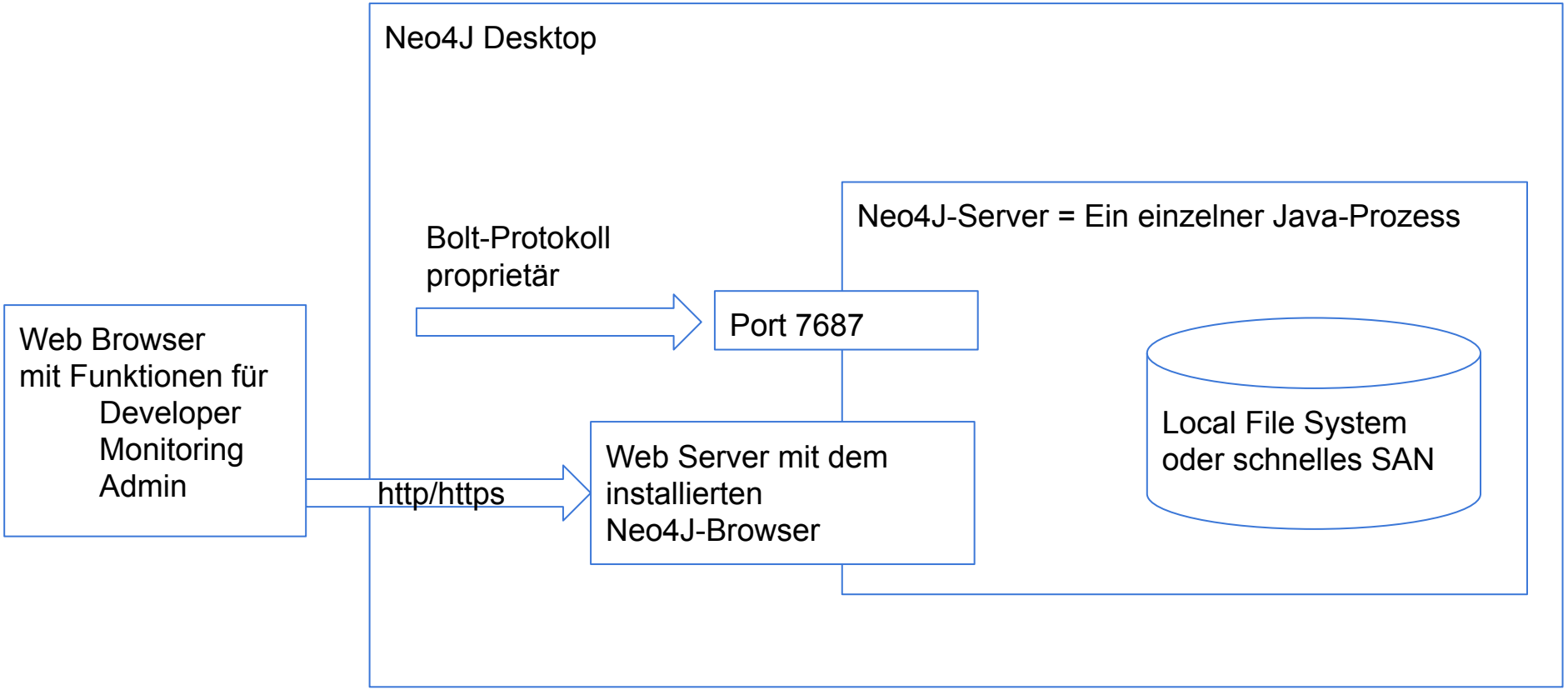
- Sie sind nicht schlecht, sondern bieten ganz andere Features
  - Konsistenz der Datenhaltung
    - Daten-Normalisierung
  - Statisches Schema inklusive Constraints
    - Inklusive Beziehungen zwischen Datensätzen
- Aktuelle Entwicklungen und Trends führen aktuell zu sehr interessanten Umsetzungen von Graphen-orientierten Ideen in relationalen Datenbank-Systemen

Unsere Trainings-Umgebung

- Benutzung der Integrata-Cegos-Rechner
  - Zugriff via RdWeb auf einen fertig eingerichteten Rechner
    - Installiert und eingerichtet ist eine Neo4J Desktop
- Eine lokale Installation des Neo4J Desktops
- Installtion eines Neo4J-Servers auf http://h2908727.stratoserver.net:7474/browser/
  - Die Präsentation erfolgt auf dieser Umgebung
    - neo4j
    - javacream

Neo4J: First Contact

Systemlandschaft



Exkurs: Neo4J-Skalierung

- Laut Katalog
  - “Ein horizontal skalierendes System”
  - VORSICHT: Eine Datenhaltung in mehreren Neo4Js paralle hat deutlichen Einfluss auf die Performance der Abfragen
- Effizientes Neo4J ist ein Server-Prozess
  - Ausfallsicherheit eher klassisch mit Active/Passive
  - Daten-Skalierung eher durch eine Partitionierung der Daten

Neo4J-“Feeling” ist sehr an relationale Datenbanken angelehnt

- Organisation der Datenbanken in einer Datenbank-Server-Instanz
  - User und Rollen regeln den Zugriff auf die Datenbanken
    - Zu den Rollen
      - PUBLIC mit lesende Zugriff auf die Default-Datenbank
      - reader -> editor -> publisher -> architect -> admin
- Abfragesprache “Cypher” ist SQL-orientiert
  - Absolute Ausnahme im Vergleich zu anderen Graphen-orientierten Systemen
    - Deren Abfragesprache sind Script-Programme, die auf der OOP Collection-Verarbeitung gründen
- Tooling
  - Daten-Export, -Import, describe/explain für die Beurteilung der Abfragen, Indizes, über Constraints Schema-Definitionen

Zur Konsole

- Zentraler Bestandteil des Neo4J-Browsers
  - Syntax-Highlighting
  - Autovervollständigung
- Ausgaben erfolgen im JSON-Format
  - Visualisierung in Neo4J-Browser
    - Tabelle
    - Plain Tabelle
    - “Code” = Raw JSON-Format

Erste Konsolen-Befehle

- Datenbanken
  - SHOW DATABASES
  - SHOW DATABASE <db-name>
  - CREATE DATABASE <db-name>
  - DROP DATABASE <db-name>
- User und Roles
  - create user <my_user> set password <pwd> change not required
  - create role...
  - grant ROLE publishers to <my_user>

Anlegen von Daten in Neo4J

- Primär sind Daten in Neo4J “Dokumente”
  - Ein Dokument besteht aus
    - Attribute-Value-Paaren
    - im JSON-Format
  - Typisierung
    - Value kann sein
      - Zeichenkette, “”, ‘’
      - Numerische Werte: 3, 42, 3.42
      - Zustände: true, false
      - eine Liste [“A”, “B”, “C”]
      - {“key”:value}
- Node-Dokumente haben noch zusätzlich ein “Label”
  - Best Practice: Verwenden Sie Label als eine Typisierung ihrer Nodes
- Relations-Dokumente haben einen verpflichtendes Type
  - Source-Node werden über die Relation mit einem Target-Node gerichtet verbunden

Cypher-Befehl zum Anlegen eines Dokuments

- CREATE
  - Node
    - ()
  - Relation
    - <from_direction> [] <to_direction>

Cypher für eine erste Abfrage

- WICHTIG: Elemente eines Graphen werden nicht selektiert, sondern an Hand eines Muster-Ausdruckes “gematched”
- MATCH
  - Node
    - ()
  - Relation [] + Directions
-

Exkurs: Cypher-Sprache

- Angelehnt an SQL
- Aber eigentlich eine Skript-Sprache mit
  - Variablen
    - Scope, eine Lebensdauer
  - Return-Anweisung
- Als erstes Beispiel ein “matche einen beliebigen Knoten”
  - MATCH (result) //result = MATCH ()
  - return result

Detail: Selektieren eines Knoten

- (x)
  - Selektiere ohne Kriterium, Ergebnis steht im Script unter dem Namen ‘x’ zur Verfügung
- (x :Label)
  - Selektion nach Label
- (x {Candidate})
  - Selektiert nach den Attributen des Candidate-Objekts
- Kombinationen möglich

Details zur Ausgabe in einer Kachel

- Umschalter
  - Graph
  - Table
  - Plain Table
  - Code
- Iteration über die Ergebnismenge erfolgt automatisch
- Genaue Darstellung (was ist die farbe der Knoten? Was wird als Info im Knoten dargestellt) wird von der Kachel automatisch bestimmt

ToDo

- Legen Sie sich eine eigene Test-Datenbank an
  - z.B. training
- In dieser Test-Datenbank anlegen von Knoten
- Labels: Person (lastname, firstname, height), Address (city, postalCode, street)
- Lernziel:
  - Umgang mit der Konsole
  - Anlegen, Listen von Knoten
  - Umgang mit der Ergebnis-Kachel
- Hinweis:
  - Es gibt auch eine where-Klausel -> später
  - Ebenso: Relationen

Erzeugen einer Relation

- create []
  - Geht nicht, TYPE ist verpflichtend
- create[:LIVES_AT]
  - Geht auch nicht: es fehlen source und target
- create match ({lastname:”Sawitzki”}) [:LIVES_AT] match (city:”München”)
  - Geht wieder nicht: es fehlt die Direction
- create match ({lastname:”Sawitzki”}) ->[:LIVES_AT] -> match (city:”München”)
  - Korrekt!
- () - [:TYPE] - ()

Details zur Relation

- Relation zwischen Nodes
  - -
- Richtungsangabe erfolgt durch spitze Klammern
  - ->
  - <-
- Sowohl die Source Nodes als auch die Target Nodes sind potenziell MENGEN (!)
- () - [:TYPE] - ()
  - Intern erfolgt intern eine zweifach verschachtelte Iteration
    - for (sourceNode in sourceNodes){
      - for (targetNode in targetNodes)
        - create sourceNode - relation - targetNode

Arbeiten mit Cypher im Detail

Rekapitulation

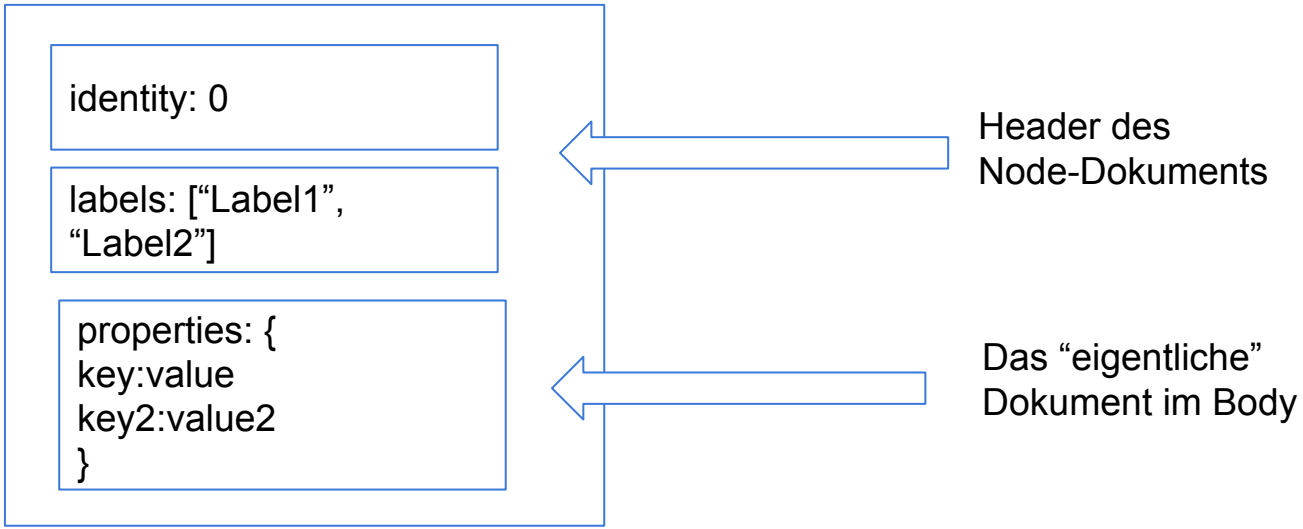
- Dokument als atomarer Datensatz
  - Schema
    - Attributen werden Werte zugeordnet
    - Dokumenten-Format in Neo4J ist JSON
    - “Schema on Read”
      - Im Gegensatz zu relationalen Datenbanken
        - “Schema on write”
  - Typisierung der Attribute erfolgt durch die Zuweisung der Werte
    - JSON unterstützt einfache Datentypen
      - Strings, “”
      - Numbers, 3, 4.2
      - Logische Werte, true|false
      - Datum, formatierter String
      - Liste, [...]
      - Objekte, {key:value}
  - Dokumente haben eine eindeutige ID
  - Können Links auf andere Dokumente enthalten

Rekapitulation

- Dokument-Typen in Neo4J
  - Node-Dokumente
  - Relation-Dokumente
- Einschränkungen im Vergleich zu der allgemeinen Dokumenten-Definition
  - Es können keine Embedded Dokumente definiert werden
  - Relation-Dokumente können nur in Verbindung mit einem Source- und einem Target-Node erzeugt werden
    - Es gibt keine Möglichkeit, “blanke” Relationen zu benutzen

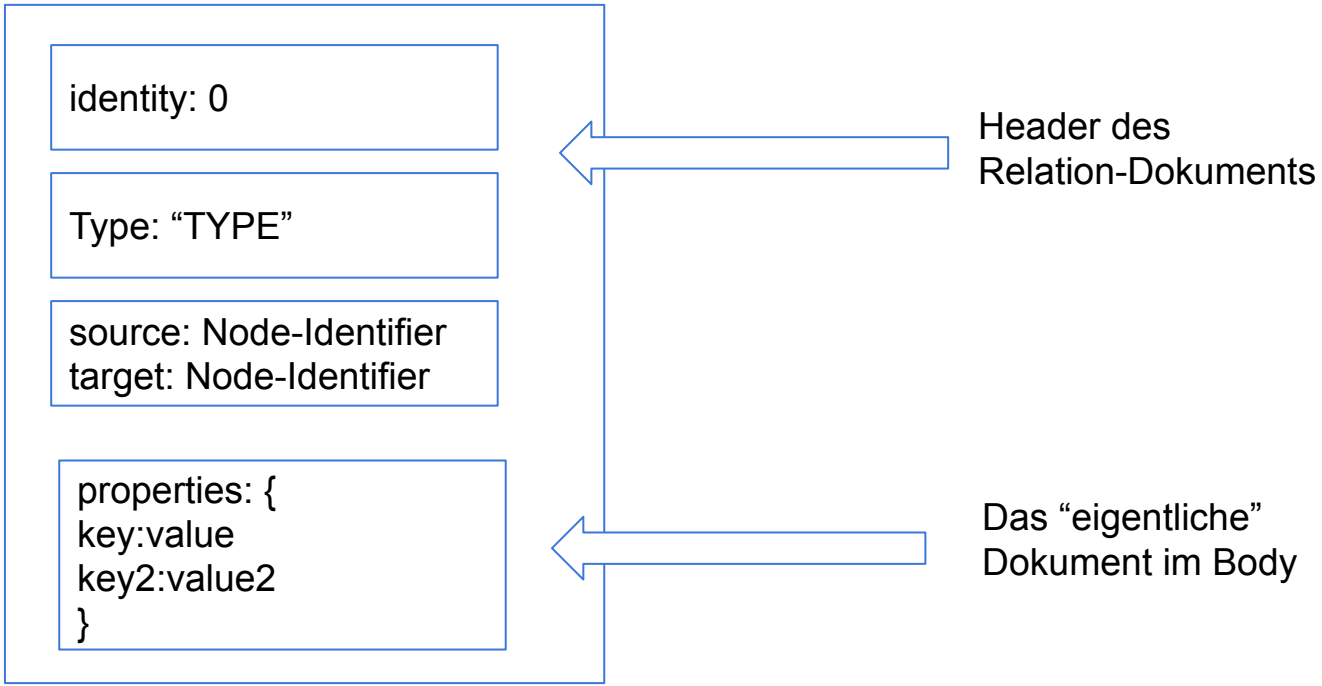
Rekapitulation

- Aufbau eines Node-Dokuments in Neo4J



Rekapitulation

- Aufbau eines Relation-Dokuments in Neo4J



Rekapitulation

- Cypher
  - ()
    - Das ist ein Node
  - () - [] - ()
    - Das ist eine Relation
- Node
  - create ()
    - Auch eine leerer Knoten bekommt selbstverständlich eine identity
  - create (:Label)
  - create ({key:value, key:value2})
  - create (:Label {key:value, key:value2})
  - create (varName :Label {key:value, key:value2})
    - entspricht: varName = create (:Label {key:value, key:value2})

Rekapitulation

- Relation
  - HINWEISE:
    - Bei der Erzeugung einer Relation wird über die Source- und Target-Nodes implizit iteriert
    - Eine Beziehung ist immer gerichtet anzugeben
  - create () - [:RELATION_TYPE] -> ()
    - Diese Anweisung erzeugt eine Relation zwischen allen Nodes

Rekapitulation

- MATCH
  - Definiert ein Muster, ein Pattern, das die Treffer determiniert
- MATCH ()
  - Das ist ein Knoten ohne Eigenschaften
- MATCH (:Label {criteria})
  - criteria: Abfrage, in der die gesetzten Eigenschaften des criteria-Objects mit AND verknüpft
- MATCH (varName :Label {criteria})
  - criteria: Abfrage, in der die gesetzten Eigenschaften des criteria-Objects mit AND verknüpft

Rekapitulation

- MATCH
  - Definiert ein Muster, ein Pattern, das die Treffer determiniert
- MATCH () - [] - ()
  - Treffer “Alle Knoten, die in irgendeiner Beziehung zu einem anderen Knoten stehen”
- MATCH (varSourceNodes) - [varRelation] - (varTargetNodes)
  - Treffer “Alle Knoten, die in irgendeiner Beziehung zu einem anderen Knoten stehen”
- MATCH (varSourceNodes :Label {candidate}) - [varRelations :TYPE {candidate3}] - (varTargetNodes :Label2 {candidate2})
-

Rekapitulation

- Konsole zur Eingabe von Befehlen
- Die Rückgabewerte einer Sequenz von Cypher-Befehler werden in einer Ergebnis-Kachel dargestellt
  - Graph, Tabelle, Plain Tabelle, Code
- RETURN
  - Rückgabe einer **Variablen** oder eines Ausdrucks
    - RETURN n
  - Genauer: Rückgabe einer Liste von Variablen
    - RETURN n, r, m
  - Hinweis: Im Neo4J-Browser gibt es die Einstellung “Automatische Darstellung der Relationen”

Wie geht es weiter?

- Formulierung komplexerer Abfragen?
  - () - [] -> () <- [] - ()
- Projektionsabfragen?
- Aggregate-Funktionen
- Update und Delete

ToDo: Projektionsabfragen, Schema on Read

- Projektionsabfragen
  - RETURN nodes.attribute
    - Die Iteration erfolgt implizit
- Legen Sie hier noch weitere Knoten an
  - Haustier, “Pet”, name, furColor, weight, type (cat, dog, ...)

where-Klausel und String-Funktionen

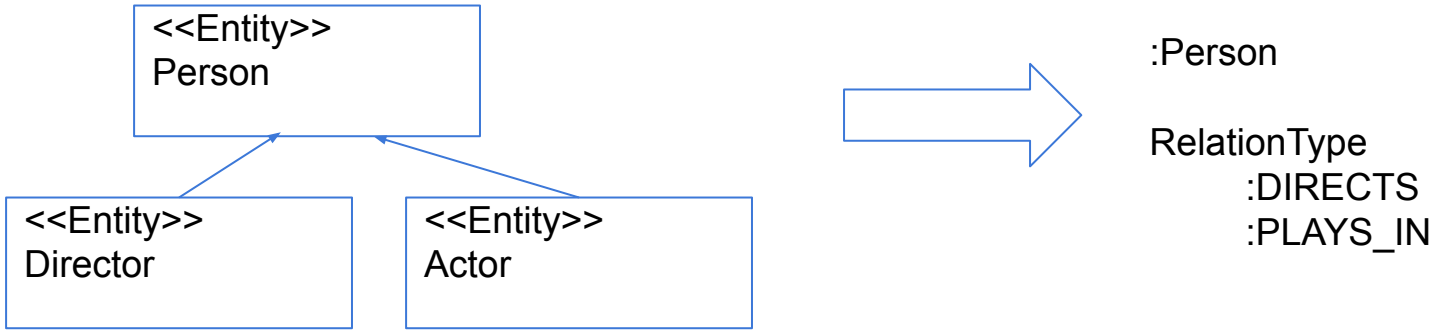
- match (...) **where** condition AND|OR condition2
- condition
  - Vergleichsoperatoren
  - String-Vergleiche
    - z.B. contains substring
    - =~ ‘Regular Expressions’
- String-Funktionen
  - left, trim, toUpperCase(), reverse, …

ToDo

- Suchen Sie Personen mit einer where-Klausel
  - Personen deren Name mit “A” beginnt
  - Geburtsdatum
- Ausgabe der Namen z.B. toUpperCase, …
-

Einbeziehung von Relationen

- “In welchen Filmen hat Tom Hanks mitgespielt?”
  - ToDo: Match unter Einbeziehung der Relation
- Welche Personen sind Movie-Directors?



Exkurs: Richtungen von Relationen

- Beim Anlegen einer Relation muss eine Richtung angegeben werden
  - () - [] - ()
    - Syntax-Fehler: Beim Anlegen muss eine Richtung definiert sein
  - () - [] -> ()
  - () <- [] - ()
- Bei Matchen ist die Richtung optional
  - MATCH () - [] - () //OK

ToDo

- “Welche Rollen spielte ein bestimmter Schauspieler in seinen Filmen?”
  - Hinweis: Diese Information steht in der Relation

Bemerkung zu Ressourcen-Verbrauch und Performance

- Es existieren allgemeine Algorithmen zur effizienten Durchforstung eines Graphen
- Identität der Dokumente wird zur Verfolgung von Relationen benutzt
- Aufbau der Nodes und Relations beinhaltet den Header mit Labels und Types
  - Diese Abfragen sind immer mit einem Index ausgestattet
- Zu beachten Sie where-Klauseln und candidates
  - Hier muss die Neo4J das Dokument öffnen und Parsen
    - {name:“Tom Hanks”}
  - Hier hilft ein selbst angelegter Index
    - create index for (p :Person) on (p.name)
  - Verwendung eines Indexes erfolgt automatisch oder durch Angabe von “using index n:Person(name)

Query Profiling

- explain
  - Der von Neo4J bestimmte Query-Plan wird angezeigt
- profile
  - Metriken der konkreten Ausführung werden mit zurück gegeben

Queries und Listen

- List Expressions
  - size(list)
    - Suche alle Personen, die in einem Film mehr als eine Rolle “gleichzeitig” spielen
  - [element in list where element.attribute …]
    - Suche alle Filme mit einer Rollen die mit “Jim”
  - …
- UNWIND
  - Erzeugt für jedes Array-Element ein eigenes, flaches Ergebnis-Dokument

ToDo

- Suche alle Personen, die in einem Film mehr als eine Rolle “gleichzeitig” spielen
- Suche alle Filme mit einer Rollen die mit “Jim”
- Ergebnis-Liste eines Schauspielers mit jeweils einer einzigen Rolle
- ...

Aktualisierung von Dokumenten

- Was mit dem Header
  - Identität des Dokuments ist absolut unveränderlich
    - Hinweis: identity ist zugreifbar über die Funktion id(n)
  - Labels eines Knoten
    - set n:NewLabel
    - remove n:ToRemoveLabel
  - Type einer Relation ist wiederum unveränderlich
- Was ist mit dem properties-Dokument
  - set n.property = value
    - Ändern oder erzeugen
  - Überschreiben durch set n ={...}
    - Ergänzen: set n += {}
  - remove n.property
  - List-Properties: analog: n.list = [...], [...] + n.list

Ideensammlung

- “Alle Personen mit einer DIRECTED-Relation sollen das Label “Director” bekommen
- “Keanu Reeves spielt in Matrix auch noch die bisher unbekannte Rolle ‘Hugo’”
-

Besonderheiten des Graphen

- Keine Joins, sondern ein Verfolgen von Relationen
- Distanz zwischen Knoten
  - [*]
    - beliebige Distanz
  - [*2]
    - Distanz: 2 Relationen
  - [*1..3]
    - Distanz : 1 bis 3 Relationen entfernt
- Sind Knoten denn überhaupt verbunden?
  - (...) - [*] - (...)
- Utility-Funktionen
  - path-Funktion gibt eine Liste der Knoten und Relationen aus
  - size-Funktion: Wie viele Pfade existieren
  - shortesPath
  - allShortestPaths

ToDo

- “Haben zwei Schauspieler im selben Film mitgespielt”
  - Irgendeiner, wenn ja: Welcher
  - Haben die in einem bestimmten Film zusammen gespielt?
- Was ist die kürzeste Verbindung zwischen zwei Schauspielern
  - Ergebnis: Wie viele Hops?
  - Welche Zwischenknoten?
- Welche Schauspieler haben noch nie miteinander einen gemeinsamen Film gedreht?
- ...

ToDo: Besonderheiten des Graphen 2

- In jeden beliebigen Daten können neue Knoten und neue Relationen eingeführt werden (!)
- z.B. LIKES
  - Personen untereinander, Personen zu Filmen
- Neues Label: Books
  - Personen können Bücher liken
  - Neue Beziehung zu einem Film: Ein Buch ist ein Drehbuch für einen Film

Programm für Mittwoch

- Best Practices für die Modellierung eines Graphen
  - Beispiel von Grund auf
- Einsatzbereiche von Neo4J an Hand etablierter Praxisbeispiele
  - Neo4J Community
- Kleine Übersicht in die Neo4J Administration und Neo4J Werkzeuge
- Workshop, Fragen und Antworten
- Seminarbeginn: 8:30, Ende gegen 16:00

Modellierung

Ausgangssituation: Relationales Modell ist vorhanden

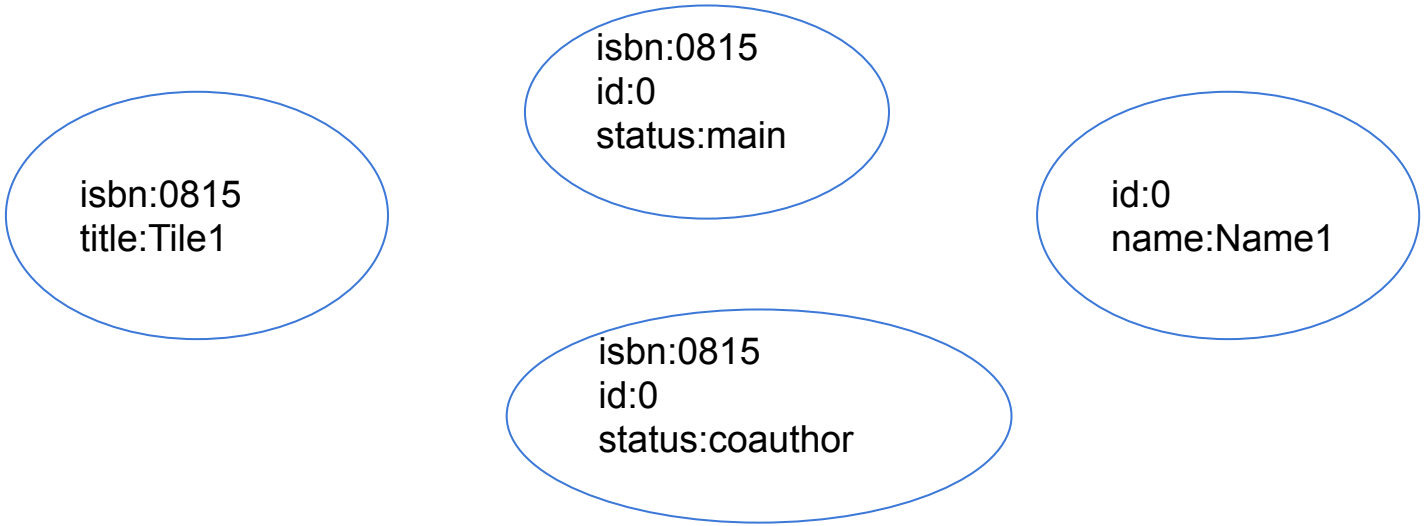
BOOKS
isbn=0815 title=Title1

BOOKS_AUTHORS
isbn=0815 id=0 status:main
isbn=0815 id=0 status:coauthor

AUTHORS
id=0 name=Name1

Umsetzung in den Graphen unter Berücksichtigung des Schemas

Umsetzung in den Graphen unter Berücksichtigung des Schemas und Daten

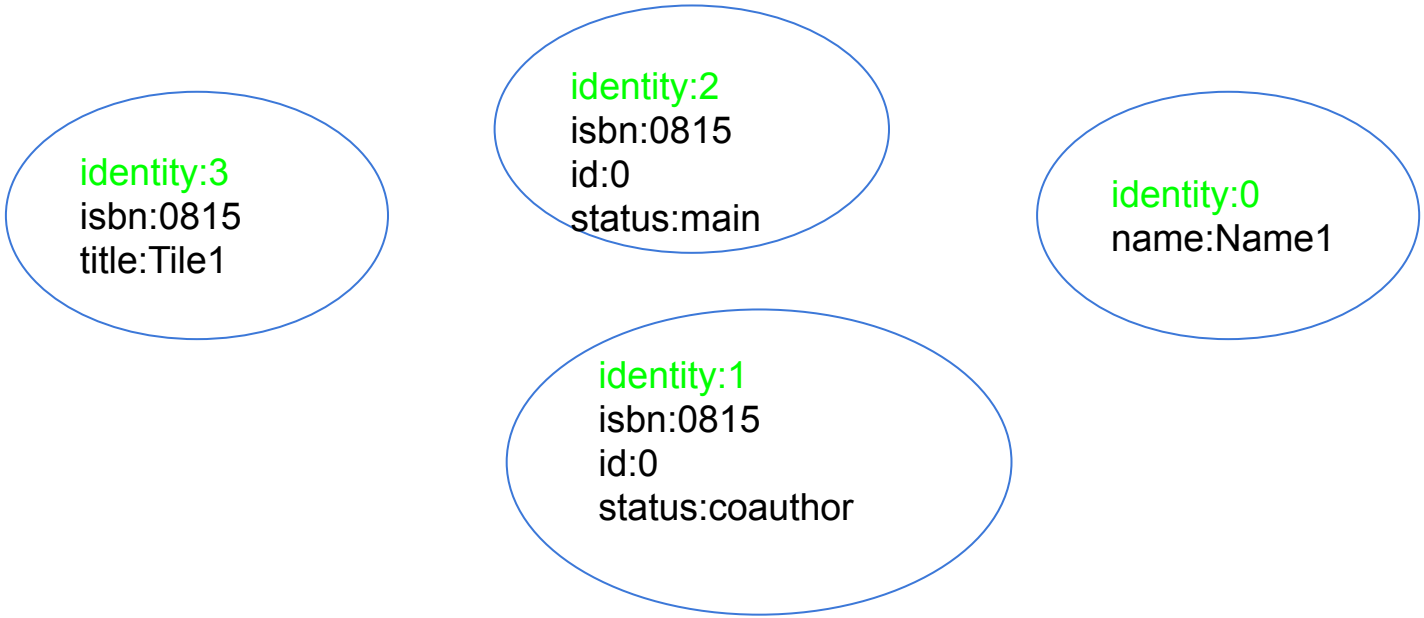


Step1: “Umschubsen der Tabellen”, jeder Datensatz ist konzeptuell ein Dokument

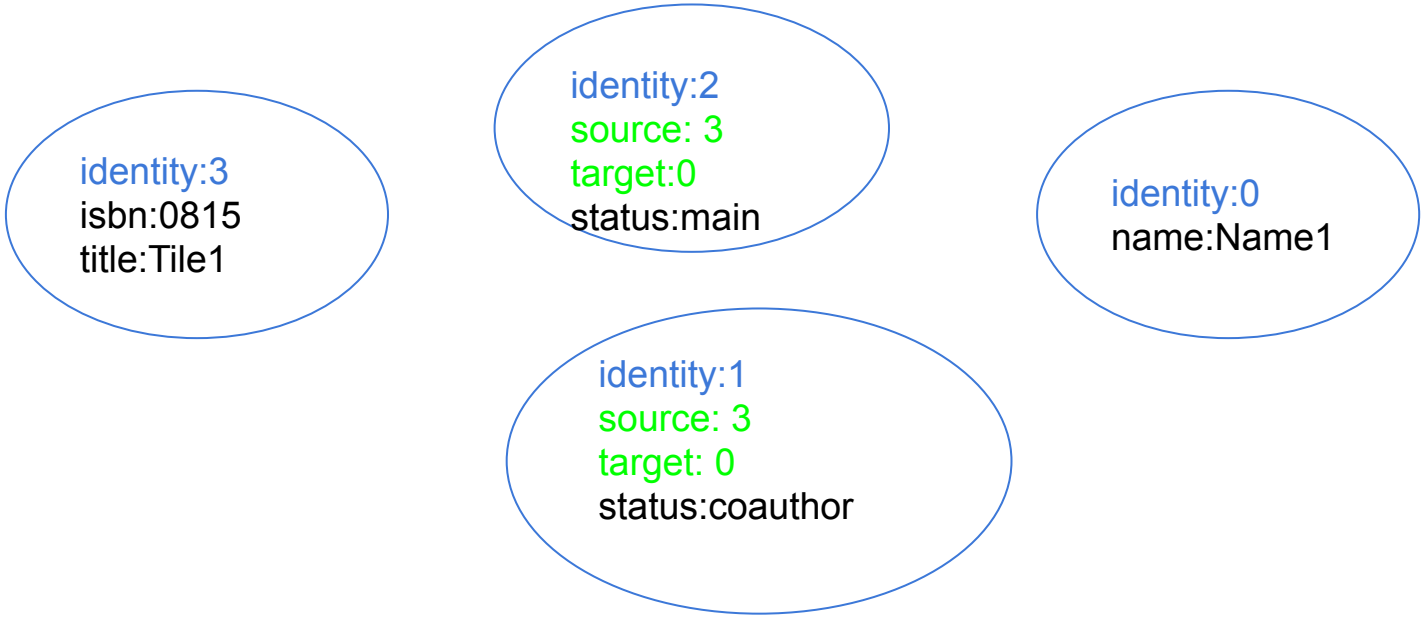
Tabellen-Schema zu Dokumenten-Schema

- Eine Spalte wird zu einem Attribut
- Genauere Analyse kann zu Listen-Attributen führen
  - z.B. BOOKS enthält eine Spalte TAGS mit einer Komma-separierten Liste von Schlüsselwörtern
  - z.B. AUTHORS mit EMAIL1, EMAIL2, EMAIL3

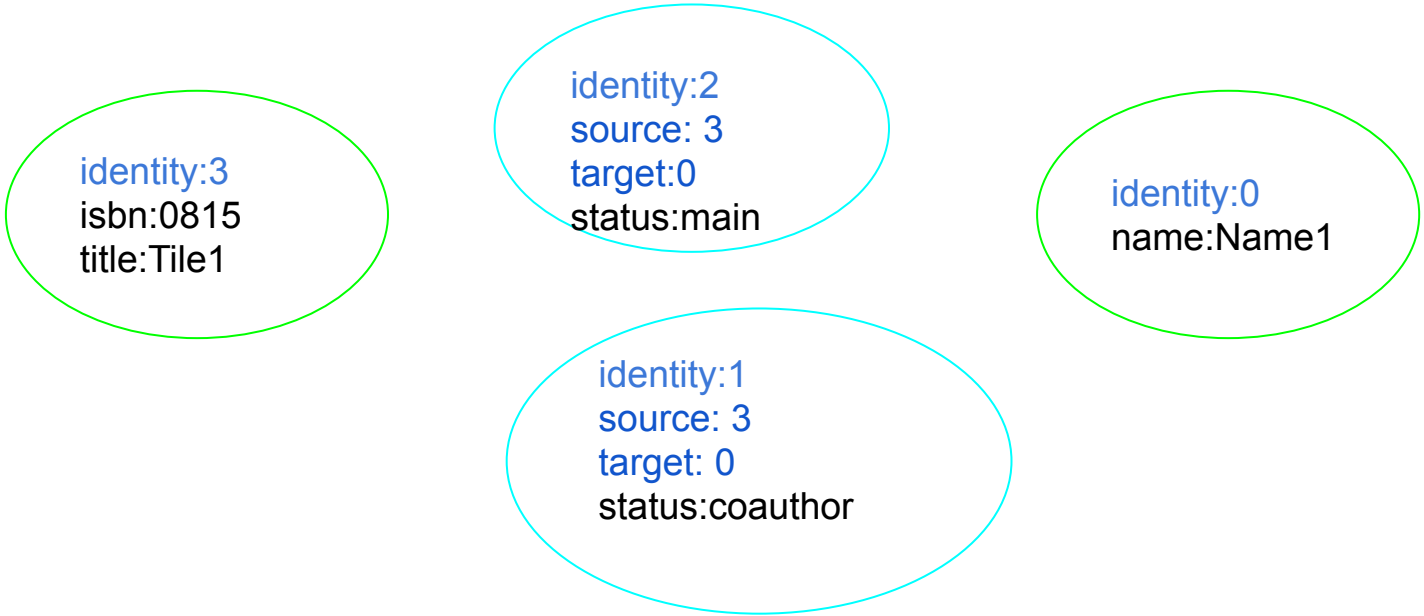
Step 2: Einführung einer Dokumenten-ID



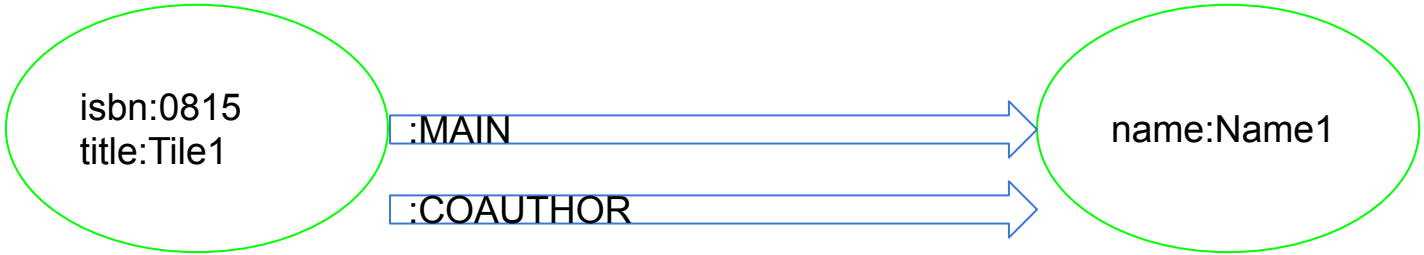
Step 3: Benutzen der Dokumenten-ID



Step 4: Zuordnung Node oder Relation



Step 5: Visualisierung als Graph

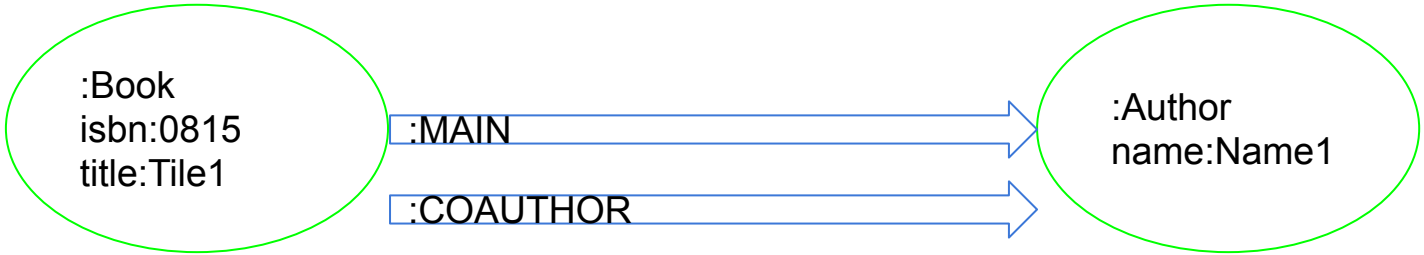


Tabellen-Schema zu Node

JAVACREAM
Training
Consulting
Projectmanagement

- Tabellen-Name wird zu einem Label

Step 5: Visualisierung als Graph

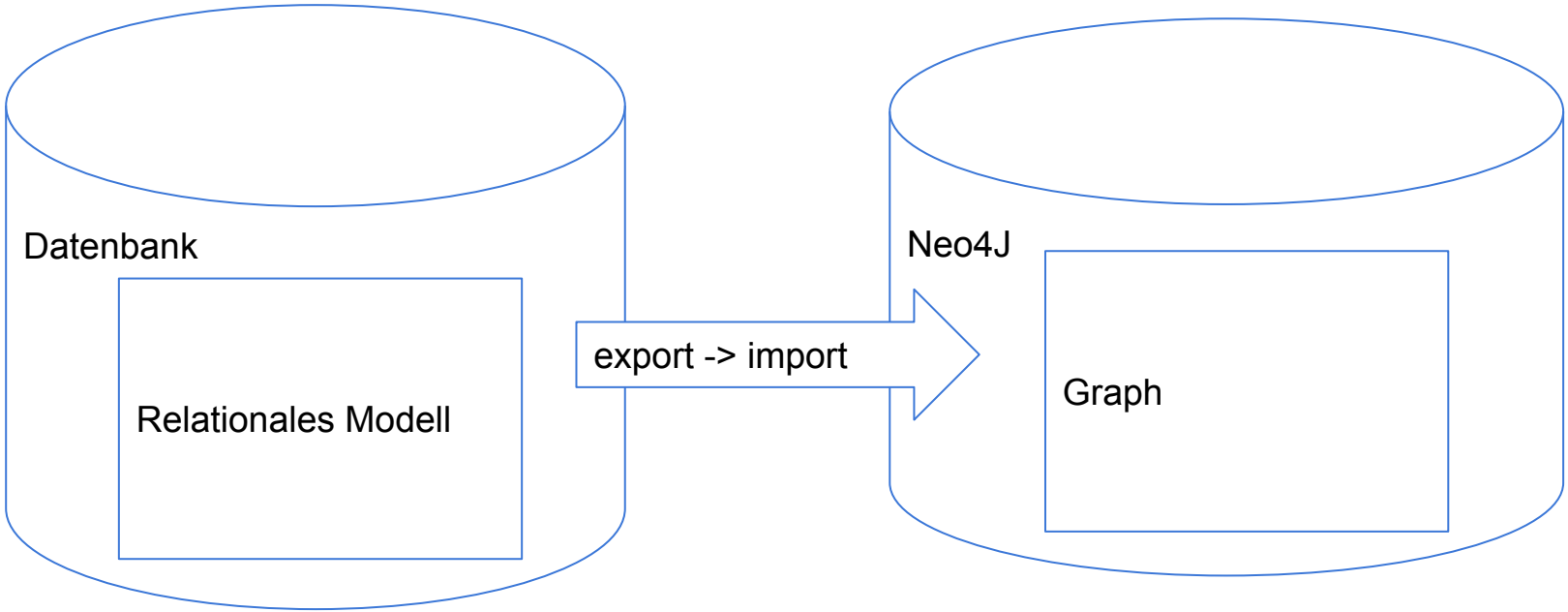


Der gefundene Graph

- In der Realität werden sehr schnell diese Graphen weiterentwickelt
  - Label Author -> Label Person
  - Andere Personen, die nicht Autoren sind können sofort hinzugefügt werden
  - Bestimmte Personen LIKES Book, Person, …
- Das Modell eines Graphen IST der Graph,
  - Visualisierung: “White Board Friendly”
- Visualisierung der “statischen” Logik
  - match(:Person) - [r*1..2] - (:Book) distinct return r

Frage

- Ist es immer notwendig, das relationale Modell in einen Graphen “umzumodellieren?” -> Nein
-



Graphen-Modell aus der Fach-Spezifikation

- “Ein Verleger verlegt Bücher, die von Autoren geschrieben werden”
  - ~~Knoten~~ Labels sind Substantive
  - ~~Relationen~~ Relation-Types sind Aktionen, Verben
- Jedes Buch hat eine Verkaufsstatistik”
  - Mehrdeutig
    - Attribut eines Buch-Node
    - Relation zu einem Verkaufs-Node
  - Entscheidung
    - Falls die Verkaufsstatistik ein zusammengesetztes, eigenes Dokument ist, dann muss bei Neo4J ein eigener Knoten benutzt werden
      - Workaroung, Hack: Embedded Dokumente sind JSON-Strings
    - “Query First”: Eine Abfrage sollte seine Kriterien möglichst im Dokument direkt finden

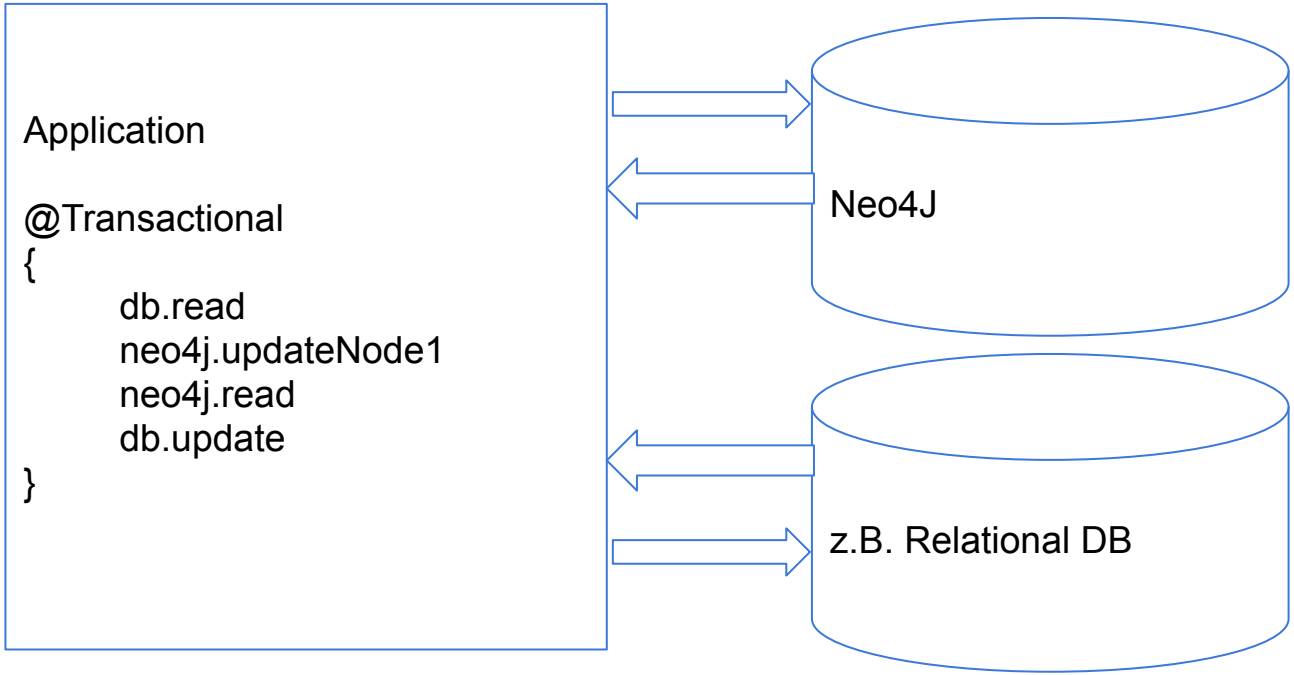
Grundprinzip der Modellierung

- Behalten Sie Ihre Abfragen im Auge!
- Refactoring
  - Konstrukt der “Hilfs-Knoten”
    - Eine Relation mit wiederverwendbaren und damit potenziell redundanten Informationen werden in ein Node umgewandelt
    - Normalisierte Datenhaltung
  - Komplexe Abfragen können vereinfacht werden
    - Relationen haben ebenfalls Dokumente
  -

Features der Neo4J

- Ausführlicher Daten-Export und -Import
  - load csv…
    - URL
      - Datei
      - Streaming
- Transaktions-Support
  - ACID (Atomar, **Consistent**, Isolated, Durable)
    - begin, commit|rollback
  - XA-fähig mit Unterstützung des Two-Phase-Commit-Protokolls
- Neo4J besitzt für alle relevanten Programmiersprache Treiber-Bibliotheken

Exkurs: XA-Transaktionalität



Betriebliche Aspekte

Installation

- Neo4J ist ein Java-basiertes Produkt
  - Notwendig ist eine installierte Java Runtime
  - Hinweise
    - Damit ist Neo4J Plattform-unabhängig
    - Die Java-Runtime verwaltet ihren Speicher selber
      - “Garbage Collection”: Aufräumen und Defragmentieren des Speichers (Heap)
      - Das führt notwendigerweise zu “Totzeiten” = “Stop the world-Effekt”
        - Normalerweise im unteren zweistelligen Millisekundenbereich
        - Schlecht ausbalancierte Systeme (Wenig CPU, viel Hauptspeicher) haben einen ausgeprägten “Stop-the-World-Effekt”, mehrere Sekunden
- Native Distribution
  - Entpacken eines Archivs
- Container-fähig, z.B. offizielle Docker-Images sind vorhanden
- Cloud-Lösungen

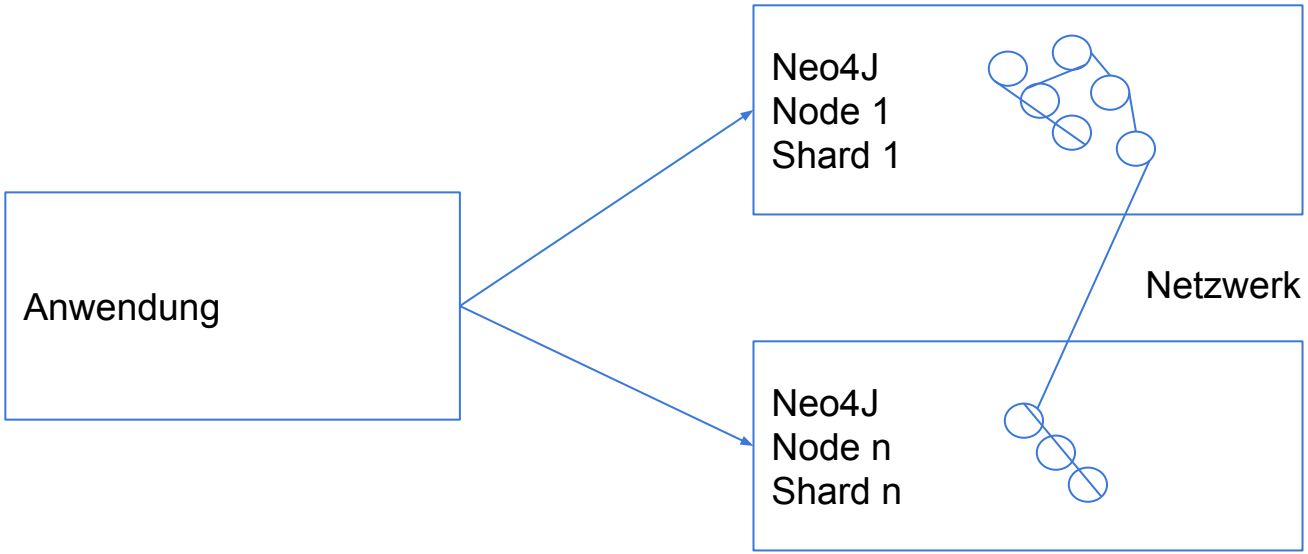
Exkurs: Heap-Größe des Java-Prozesses

- -Xms8G initiale Größe
- -Xmx8G Maximalgröße

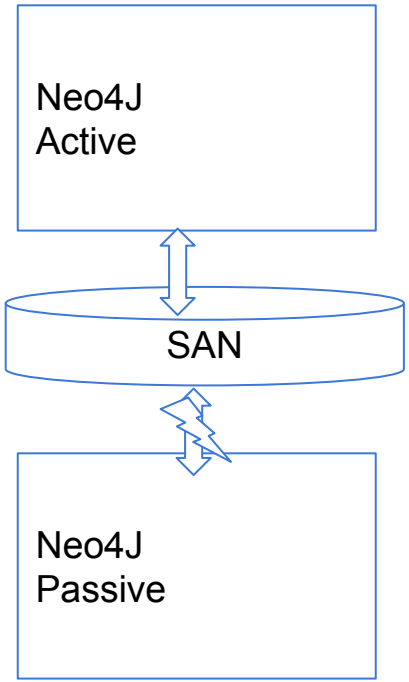
Überwachung von Neo4J

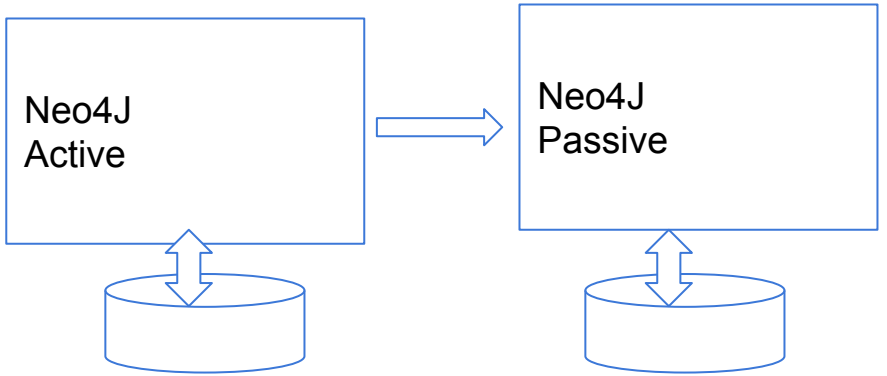
- Java Management Extension, “JMX” liefert sehr detaillierte und ausführlich Metrik-Informationen
  - CPU, I/O liefert die Betriebssystem-Überwachung
  - Speicher etc: java.lang:type=Memory
  - neo4j:ThreadPool
- Sinnvolle Erweiterung
  - “jolokia”
    - https://java.integrata-cegos.de/jolokia-simples-management-von-java-anwendungen/

Cluster-Betrieb von Neo4J



Active/Passive





Endpoints von Neo4J