1. Graphdatenbanken
    - Konzept entstand bereits in den 80er / 90er Jahren (Objektdatenbanken)
    - wieder aufgegriffen aus der Anforderung heraus komplexe Datenbestände (Semantic Web) effizient zu verwalten

1.1. Anwendung
- Verwaltung [umfangreicher] stark vernetzter [semi-strukturierter] Daten
- Effiziente Traversierung ("Indexfreie Adjazenz" konstante Performance bei Abfragen)
- Einfacher Umgang mit rekursiv vernetzten Informationen

1.2. Datenmodell
- Property [Hyper]Graph
    - gerichteter, multi-relationaler Graph
    - Knoten und Kanten durch Attribute (Key, Value) erweitert
    - Mehrere Knoten- und Kantentypen (Schema) (sones)
    - Darstellung aller Arten von Graphen (ungerichtet, gerichtet, gewichtet)
    - Explizite Darstellung der Miniwelt
    - Möglichkeit zur Darstellung von Mehrfachkanten (verschiedenen Typs / Labels)
    - Relationen höherer Ordnung (Hyperkanten) (sones, HypergraphDB)

1.3. Vertreter
- neo4j (Java, GPLv3)
- DEX (C++ Kern, Java / .NET API)
- OrientDB (Kombination aus Graph- und Dokumentendatenbank, Java, AGPLv3)
- sones (.NET, nicht persistente open source Implementierung des Property-Hypergraph)

1.4. Vorteile
- konstante Performance bei der Abfrage vernetzter Daten bis theoretisch unendlicher Tiefe
- Vermeidung von JOINs
- Support durch Community (insbesondere neo4j)
- Unterstützung vieler Programmiersprachen
- Vermeidung des OR Mapping Problems

1.5. Nachteile
- noch keine Standards (in Arbeit z.B. Blueprints Projekt)
- keine einheitliche Abfragesprache (GraphQL, Gremlin, Algorithmisch über APIs)

1.6. Einsatzszenarien
- Ontologien (Abbildung, Verwaltung)
- Semantic Web, Information Retrieval (Kookkurrenzgraphen)
- Webgraph (Hyperlinkstruktur)
- Ranking Algorithmen (Page Rank)
- "Wer kennt wen" Szenarien in sozialen Netzwerken (kürzeste Pfade)
- Fahr- / Flugplanoptimierung (Maximaler Fluss)
- Empfehlungssysteme (Bipartites Matching)
- GIS (kürzeste Wege, Routing)
- Dokumentenverwaltung (z.B. Patente, Vertäge)

1.7. Relevant für Praktikum
- anderes Anwendungsgebiet, Einsatz jedoch nicht grundsätzlich falsch
- Indizierung von Zeitstempeln und Bereichsabfragen möglich (Lucene bei neo4j)

- Datenmengen sind verwaltbar
- Schemafreie bzw. semi-strukturierte Datenspeicherung möglich

2. Dokumentendatenbanken

2.1. Anwendung

- Verwaltung schemalosen Daten
- Daten werden in Dokumenten verwaltet (Äquivalent zu Tupel)
- Zugriff über einfache Schnittstellen (HTTP REST (CRUD))

2.2. Datenmodell

- Verwaltung der Daten in Dokumenten in spezifischem Formaten
    - JSON, XML, BSON, YAML, binäre Formate
- jedes Dokument besitzt einen eindeutigen Bezeichner (unique key, URI)
- verschiedene APIs zur Abfrage der Daten (z.B. MapReduce bei CouchDB)

2.3. Vertreter

- CouchDB
- MonoDB
- OrientDB

2.4. Vorteile

- Schemafreiheit (flexibel hinsichtlich Änderungen)
- Robust, fehlertolerant
- Konflikterkennung und Konfliktmanagement (CouchDB)
- Skalierbarkeit (CouchDB Lounge, MonoDB)
- Unterstützung vieler Programmiersprachen

2.5. Nachteile

- Umsetzung von Normalformen nicht vorgesehen
- Referentielle Integrität nicht möglich
- keine Standards

2.6. Einsatzszenarien

- Webanwendungen
- Dokumentenablage (Lotus Notes)

3. Graphdatenbank neo4j
    - seit 2003 in Betrieb
    - seit 2007 eigenständiges Projekt
    - ACID-transaktionale Graphdatenbank
    - Indexstrukturen via Apache Lucene

3.1. Datenmodell
    - Implementierung des Property Graph Datenmodells
    - Schemalos im Kern
    - Schemata können durch APIs bzw. die Applikation definiert werden

3.2. Datenabfrage
    - Traverser API
    - Gremlin
    - Tinkerpop
    - Cypher (deklarative Matchingsprache)

3.3. Replikation / Skalierbarkeit
    - aktuell "nur" Master-Slave Replikation
    - Ein Write-Master N Read Slaves (write master bottleneck)
    - aktuell kein Sharding möglich

4. Dokumentendatenbank CouchDB
    - seit 2005 in Entwicklung (davon 2 Jahre bei IBM)
    - seit 2008 Apache Top Level Projekt
    - schemafreie, dokumentenorientierte Datenbank
    - Daten werden im JSON Format verwaltet
    - Zugriff via RESTful JSON API (http)
    - Filtern (Queries) via Javascript Funktionen (MapReduce)
    - unterstützt Replikation auf mehrere Knoten (Parallelisierung von Anfragen)
    - gewährleistet ACID Eigenschaften (optimistisches Locking via MVCC)

4.1. Datenmodell
    - Dokumente werden in einem B-Baum organisiert und enthalten eindeutige Dokument-ID und Revisions-ID (letzteres dient der inkrementellen Änderungsverfolgung)
    - Datenformat JSON
        - JSON = Set of <Eigenschaft>
        - <Eigenschaft> = Key => Value | <Eigenschaft>
        - Darstellung komplexer, verschachtelter Informationen möglich
        - wird auch zur Speicherung verwendet

4.2. Datenabfrage
    - View Modell
    - erzeugen von "Views" = Sichten auf die DB in speziellen "design documents"
    - Nutzung von MapReduce (Anwender schreibt Map Funktion in Javascript)
    - Views werden in dedizierten Indices abgelegt (welche bei Änderung ebenfalls aktualisiert werden um die Anfrage zu beschleunigen)

4.3. Replikation

- unterstützt Replizieren von Daten auf mehrere Knoten
- bidirektionale Konflikterkennung
- *peer based distribution*, *offline by default* (typische für Webanwendungen)
- CouchDB wählt deterministisch gewinnende Version bei einem Merge aus, Anwender kann dies aber widerrufen und manuell auswählen

4.4. Skalierbarkeit
- Skalierung von Daten über CouchDB Lounge (wurde für Meebo entwickelt)

6. MongoDB
- entwickelt für die Speicherung umfangreicher Datenmengen
- entwickelt mit dem Ziel einer möglichst hohen Leistung (kurze Reaktionszeit auch bei umfangreichen Datenmengen)

6.1. Datenmodell
- schemalos (grundlegende Struktur für Indizierung zu empfehlen)
- Möglichkeit, mehrere Datenbanken zu verwalten
- Datenbank beinhaltet "Collections" (äquivalent zu Tabellen)
- Collections enthalten Dokumente (äquivalent zu Tupel)
- Dokumente ähneln assoziativen Arrays (Maps, Dictionary)
- Dokumente im BSON (Erweiterung von JSON Binary Json) , binär-encodierte Serialisierung von JSON
- maximale Dokumentengröße 4MB (sharding jedoch möglich)

6.2. Datenabfrage
- MapReduce via Javascript (ermöglicht Batch Processing, Gruppierung)
- API Java

6.3. Replikation
- Master / Slave (Write Master) eventual consistency
- Slaves beantworten Leseanfragen
- Master kann entweder manuell oder automatisch festgelegt werden
- Replica Set (in Verbindung mit sharding) Erweitert Master Slave um automatische Fehlerbehandlung und automatisches Wiederherstellen von Nodes

6.4. Skalierbarkeit
- automatisches Sharding
- auf Ebene der Collections
- Festlegung von Sharding Keys (einer oder mehrere) (Ziel: lokaler Bezug benachbarter Dokumente)
- Meta Daten über die MongoDB Instanz und den aktuellen Status des Sharding werden von mehreren Config Servern verwaltet
- Client -> Routing Server -> Config Server -> Zuweisung zum Shard

7. Eignung

	Neo4j	CouchDB	MongoDB
Persistenz	Ja	Ja	ja
Datenmodell	Property Graph	JSON Dokumente	BSON Dokumente
Datenvolumen (Single Machine)			Peta-, Terabyte Bereich (via Sharding), Dokumentengröße auf 4MB beschränkt
Entwicklungssprache	Java	Erlang	C++
ACID	ja	Ja (MVCC)	Ja (in place update)
Anfragearten	Traverser-API, Blueprints, Tinkerpop Gremlin, SPARQL	MapReduce (JavaScript)	MapReduce (JavaScript), APIs (Java,...)
Bereichsabfragen	Ja (Lucene)	Ja (Map Funktion)	Ja (Map Funktion)
Replikation	Master Slave Replizierung mit Master-Failover	Fehlertolerant "peer based distributed", komplexe Merge Mechanismen	Master / Slave, Replica Set
Sharding	In Entwicklung	Horizontal via CouchDB Lounge	Automatisches Sharding (sharding keys)
Versionierung	Durch Applikations-schema oder evtl. über Github Projekt "neo4j-versioning"	Ja, Revision-ID	Evtl. über Diff Dokumente (ist ja eher selten)
Lese- vs. Schreibperformance	Leseperformance, Write Master ist bottleneck ohne sharding	Leseperformance MVCC verringert Schreibperformance	Ausgeglichen (Leseperformance überwiegt)
Schemafreiheit	Ja (Schemata über externe APIs)	Ja	ja
Lizenz	GPLv3 (Community Edition)	Apache License 2.0	AGPLv3
Bemerkungen		Capped Collections (feste Größe, FIFO, hohe Insert Rate)	64 Bit OS erforderlich (Memory Mapped Files), Fremdschlüsselbeziehungen über embedded documents oder DBRef Object

Entscheidung: MongoDB