Entwicklercamp 2012

Track 4 Session 6

NoSQL-Datenbanken – ein Überblick

Karsten Lehmann | Geschäftsführer | Mindoo GmbH

NOTES & DOMINO
ENTWICKLERCAMP

Über uns

- Mindoo ist IBM Business Partner und Notes/Domino Design Partner
- Wir konzentrieren uns auf die „neuen“ Aspekte der IBM Lotus Notes-Entwicklung
  - Eclipse/Expeditor-Plugins und Rich-Client-Anwendungen
  - XPages-Anwendungen
- Sowie Web-Applikationsentwicklung für IBM Websphere und Oracle Glassfish
- Karsten Lehmann und Tammo Riedinger
  - Gründer der Mindoo GmbH
  - Seit 2004 Entwickler der Applikation MindPlan® für die Haus Weilgut GmbH, Mindmapping und Projekt-Management für Lotus Notes, IBM Award Winner 2008
- Weitere Informationen:
- http://www.mindoo.de

[mindu:]
Mindoo GmbH

Agenda

- Einführung
- Was ist NoSQL?
- Klassen von NoSQL-Datenbanken
- Kundera-Bibliothek
- Bestandsaufnahme: Wo steht Lotus Notes?
- Zusammenfassung
- Q&A

Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel.

Motivation

- Nicht für jedes Problem ist Lotus Notes die beste Lösung
- Lotus Notes eine der ältesten Vertretern von NoSQL
- Seit Jahren mehr und mehr Dynamik im NoSQL-Datenbankbereich, leider wenig Fortentwicklung von NSF
- durch Kenntnis, was "die anderen" können, lernt man die Vorteile von Lotus Notes zu schätzen, stellt berechtigte Forderungen bei IBM und hat mehr Werkzeuge im Repertoire
- Vortrag gibt einen groben NoSQL-Marktüberblick aus Entwicklersicht:
  - was können die anderen?
  - welche Probleme löse ich mit welcher Datenbank?
  - Code-Snippets, um Hemmungen abzubauen – es ist alles ganz leicht :-)

Agenda

- Einführung
- Was ist NoSQL?
- Klassen von NoSQL-Datenbanken
- Kundera-Bibliothek
- Bestandsaufnahme: Wo steht Lotus Notes?
- Zusammenfassung
- Q&A

Was ist NoSQL?

- "**Not only SQL**", nicht "No SQL"!
- nimmt Abstand vom relationalem Schema zugunsten der Skalierbarkeit
  → Vermeidung von JOINs
- ermöglicht / vereinfacht Skalierbarkeit und Sharding: Verteilung der Daten auf mehrere vernetzte Maschinen (Shards)
- flexibles (natürliches) Speichermodell
- große Anzahl mehr oder weniger aktiver Anbieter:
  Über 130 NoSQL-Datenbanken auf http://nosql-database.org !

CAP-Theorem

- nach Eric Brewer, 2000; seit 2002 bewiesen und Theorem
- geeignet zur Bewertung der Datenbankzuverlässigkeit
- wichtige Eigenschaften bei verteilter Datenhaltung:
  - **Konsistenz (Consistency)**
    Alle Clients haben jederzeit dieselbe Datensicht
  - **Verfügbarkeit (Availability)**
    Clients können jederzeit lesen und schreiben
  - **Partitionstoleranz (Partition tolerance)**
    Das System arbeitet vollständig trotz physikalischer Netzwerkteilung
- Theorem: Bei verteilter Datenhaltung nur zwei der drei Eigenschaften möglich

CAP-Theorem

Datenmodelle:
Relational
Schlüssel-/Wert-orientiert
Spalten-basiert
Dokument-orientiert
Graph-orientiert

Verfügbarkeit:
Clients können jederzeit lesen und schreiben

CA
RDBMS
Neo4j

AP
CouchDB
Cassandra
Lotus Notes
Voldemort

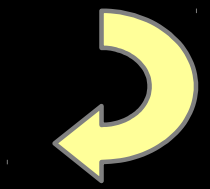
Konsistenz:
Alle Clients haben jederzeit dieselbe Datensicht

CP
CouchBase
HBase
MongoDb
Redis

Partitionstoleranz:
Das System arbeitet vollständig trotz physikalischer Netzwerkteilung

CAP-Theorem

- führt zu Aufgabe der ACID-Anforderungen, hin zu BASE
  - **Atomicity** - Transaktion vollständig oder gar nicht umsetzen
  - **Consistency** - Gewährleistung von Integrität der Datenbank
  - **Isolation** - Transaktionen werden unabhängig von gleichzeitig laufenden Transaktionen ausgeführt
  - **Durability** - Nach Commit der Transaktion sind die Änderungen dauerhaft gespeichert



  - **Basically available** - System ist generell verfügbar, Verfügbarkeit ist aber nicht garantiert
  - **Soft state** - Systemzustand kann sich im Laufe der Zeit selbst ohne Dateneingabe ändern
  - **Eventual Consistency** - letztendliche Konsistenz: temporäre Inkonsistenz der Datenbank wird akzeptiert, Konsistenz wird erst verzögert wiederhergestellt

Agenda

- Einführung
- Was ist NoSQL?
- Klassen von NoSQL-Datenbanken
  - Schlüssel/Wert-orientiert
  - Spalten-orientiert
  - Dokument-orientiert
  - Graph-orientiert
- Kundera-Bibliothek
- Bestandsaufnahme: Wo steht Lotus Notes?
- Zusammenfassung
- Q&A

Schlüssel/Wert-orientiert

Schlüssel/Wert-orientiert (Key-Value)

- Vertreter
  - Memcached (u.a. YouTube, Wikipedia, Twitter, Flickr)
  - Redis (VMWare beschäftigt Entwickler für Arbeit an Redis)
  - Voldemort (Code von LinkedIn unter Open Source gestellt, u.a. Nokia Committer)
- Datenmodell memcached
  - unstrukturierter Map von Key/Value-Paaren
  - Analog einer Java-HashMap

```
function get_foo(foo_id)
    foo = memcached_get("foo:" + foo_id)
    if (foo!=null) return foo

    foo = fetch_foo_from_database(foo_id)
    memcached_set("foo:" + foo_id, foo)
    return foo
end
```


Schlüssel/Wert-orientiert (Key-Value)

- Redis erweitert reinen Key/Value-Store um Listen, Sets und sortierte Sets
- Setzen/Lesen von Werten mit Ablaufzeitraum (Key="resource:lock")

```
SET resource:lock "lockowner1"
//automatic expiration of data
EXPIRE resource:lock 120
GET resource:lock => "lockowner1"
//check time-to-live
TTL resource:lock => 113
```

- Arbeiten mit Listen

```
RPUSH friends "Tom" => ["Tom"]
RPUSH friends "Bob" => ["Tom","Bob"]
LPUSH friends "Sam" => ["Sam", "Tom", "Bob"]
LRANGE friends 0 1 => ["Sam","Tom"]
```


Schlüssel/Wert-orientiert (Key-Value)

- Arbeiten mit Sets (ungeordnete Wertsammlungen)

```
SADD key1 "a"
SADD key1 "b"
SADD key1 "c"
SADD key2 "c"
SADD key2 "d"
SADD key2 "e"
SINTER key1 key2 => "c"
```

- Arbeiten mit Sorted Sets (geordneten Wertsammlungen):

```
ZADD hackers 1940 "Alan Kay"
ZADD hackers 1953 "Richard Stallman"
ZADD hackers 1965 "Yukihiro Matsumoto"
ZADD hackers 1916 "Claude Shannon"
ZADD hackers 1969 "Linus Torvalds"
ZADD hackers 1912 "Alan Turing"
ZRANGE hackers 2 4 => ["Alan Kay","Richard Stallman","Yukihiro
Matsumoto"]
```


Anwendungsfälle

- RAM-Caches
- schnelle Speicherung von Logs (z.B. Fehlerlogs von Load Balancer)
- In-Memory MessageQueue zur Zwischenpufferung von Nachrichten
- mit etwas Kreativität komplexere Anwendungen, z.B. ein Twitter-Clone:
  → http://redis.io/topics/twitter-clone

Vor- und Nachteile

- Vorteile
  - Performance hoch und vorhersagbar
  - Einfaches Datenmodell
  - klare Trennung der Speicherung von der Anwendungslogik (wg. mangelnder Abfragesprache)
- Nachteile
  - Eingeschränkter Funktionsumfang
  - hoher Entwicklungsaufwand für komplexere Anwendungen

Spalten-orientierte Datenbanken

Spalten-orientierte Datenbanken

- Vertreter
  - Apache Cassandra (ehem. Facebook, Twitter für Tweeds, Digg, Netfix)
  - Apache HBase (u.a. Facebook, Twitter)
- Datenspeicherung als Spaltenwerte zu einem Zeilenschlüssel (Row-ID)
- Beliebige Spaltenwerte pro Zeile speicherbar, Anzahl nur begrenzt durch Speicherplatz
- Sharding erfolgt über Row-ID; Daten einer Zeile müssen auf einen Server passen
- Server sortiert Spalten automatisch; ermöglicht Abbildung von Datenlisten

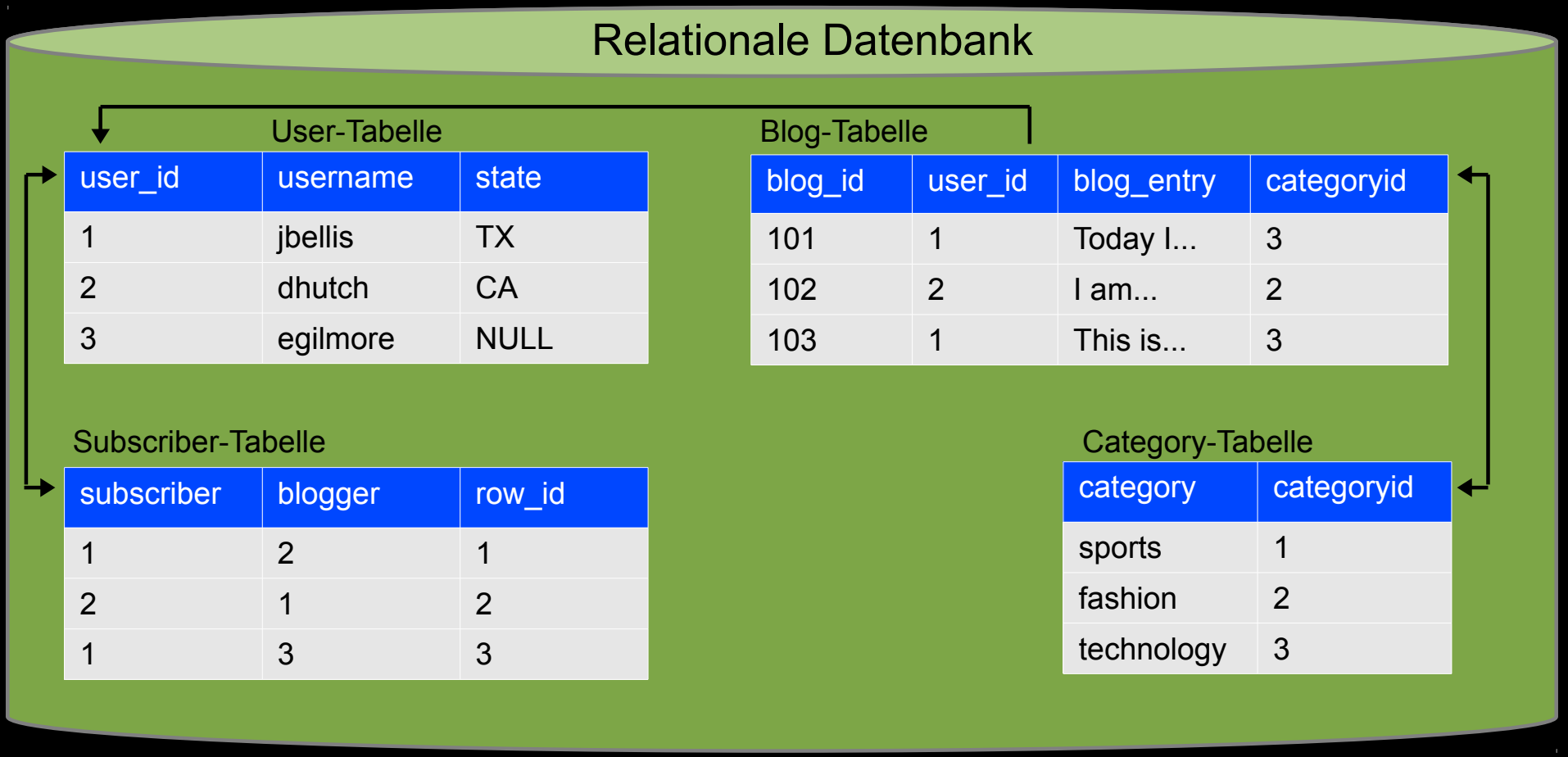
- "If your data model has no rows with over a hundred columns, you're either doing something wrong or you shouldn't be using Cassandra"

Apache Cassandra im Detail

- implementiert in Java
- Sprachenunterstützung: u.a. Java, JavaScript, Python, PHP, Ruby
- ursprünglich bei Facebook entwickelt
- seit 2008 freigegeben, seit 2010 Top Level Projekt bei Apache
- große Community von Entwicklern und Nutzern

Apache Cassandra im Detail

- Relationale Modellierung eines Blogs



User-Tabelle

user_id	username	state
1	jbellis	TX
2	dhutch	CA
3	egilmore	NULL

Blog-Tabelle

blog_id	user_id	blog_entry	categoryid
101	1	Today I...	3
102	2	I am...	2
103	1	This is...	3

Subscriber-Tabelle

subscriber	blogger	row_id
1	2	1
2	1	2
1	3	3

Category-Tabelle

category	categoryid
sports	1
fashion	2
technology	3

Apache Cassandra im Detail

- Umsetzung des Blogs mit Cassandra

Keyspace „Blog“

Statische Column Family (CF)„Users“

jbellis	name	state
	Jonathan	TX
dhutch	name	state
	daria	CA
egilmore	name	state
	eric	

Statische Column Family „Blog entries“

92dbeb5	body	user	category
	Today I...	jbellis	tech
d418a66	body	user	category
	I am...	dhutch	fashion
6a0b483	body	user	category
	This is...	egilmore	sports

Dynamische CF „time_ordered_blogs_by_user“

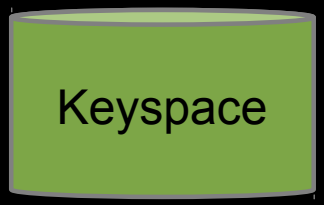
jbellis	1289847840615	
	92dbeb5	
dhutch	1289847840615	
	d418a66	
egilmore		1289847844275
		6a0b483

Dynamische CF „subscribers_of“

jbellis	dhutch	egilmore
dhutch	egilmore	dhutch
egilmore	jbellis	

Apache Cassandra im Detail

- Keyspace
  - Container für mehrere Column Families
  - z.B. zur Kapselung sämtlicher Daten einer Anwendung
- Column Family
  - Vergleichbar mit Datenbanktabelle in RDBMS
  - Gruppierung mehrerer Columns
  - optionale Festlegung der Column-Datentypen
  - jede Zeile kann im Gegensatz zu RDBMS andere Spaltenwerte haben
- Static Column Family
  - Feste Spaltenwerte, z.B. für Eigenschaften eines Objekts
- Dynamic Column Family
  - Beliebige Spalten, sortiert nach Spaltenname (Name z.B. auch Timestamp oder UUID)
  - Abfrage von Spaltenbereichen möglich (Jahr 2011-2012)
  - Je nach Fall genügt Spaltenname als Information aus, Wert bleibt leer

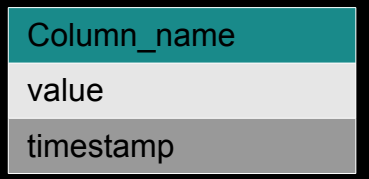


jbellis	name	state
	Jonathan	TX
dhutch	name	state
	Daria	CA
egilmore	name	state
	Eric	

jbellis	dhutch	egilmore
dhutch	egilmore	dhutch
egilmore	jbellis	

Apache Cassandra im Detail

- Column
  - kleinste Dateneinheit
  - Tupel mit (Name, Wert, Zeitstempel)
  - Wert mit höchstem Zeitstempel gewinnt
- Special Columns
  - Expiring columns: Spalten mit automatischem Verfall
  - Counter columns: Spalte ohne Timestamp-Vergleich beim Schreiben; DB garantiert, dass Wert erhöht wird
- Super Columns
  - Gruppierung mehrere Spalten zu einem gemeinsamen Lookup-Wert (z.B. „Address“ mit Spalten „Street“, „ZIP“, „City“)
  - Sortierung von Super Column und enthaltenen Sub Columns definierbar
  - eingeschränkt nutzbar, da das Lesen einer Sub Column das Laden aller Sub-Column der Super-Column erfordert



Anwendungsfälle

- HBase
  - Data-Warehousing, Analyse großer Datenbestände mit Map/Reduce (Hadoop-Projekt)
  - Vorteile gegenüber Cassandra beim Lesen
- Cassandra
  - Protokollierung großer Datenbestände (z.B. Kursdaten, Sensordaten oder Logdateien)
  - Vorteile gegenüber HBase beim Schreiben
  - Daten werden ohne vorherigen Lesevorgang geschrieben (append only)
  - Daten auf Disk nachträglich nicht geändert, Compaction bereinigt veralteten Datenbestand
- generell Anwendungen mit wenig Abfragedynamik und sehr großem Datenumfang

Vor- und Nachteile

- Vorteile
  - Getrimmt auf Performance
  - Gegenüber Key/Value-Store native Unterstützung für persistente Views
  - Sharding: Verteilung der Daten auf mehrere Server über Hashing der Row-ID
  - Spaltenorientierte Systeme effizienter als zeilenorientierte beim Aggregieren von wenigen Spalten von vielen Zeilen
- Nachteile
  - Eingeschränkte Abfrage-/Filtermöglichkeiten für Daten
    → Ausgleich möglich durch Kombination mit Apache Lucene/Solr: Solandra-Projekt
  - Hoher Pflegeaufwand beim Ändern/Löschen bestehender Daten wg. Aktualisierung sämtlicher Listen
  - Weniger effizient als zeilenorientierte Systeme beim Zugriff auf viele Spalten einer Zeile

Dokument-orientierte Datenbanken

Dokument-orientierte Datenbanken

- Vertreter
  - MongoDB (u.a. SAP, MTV, Sourceforge, Foursquare)
  - Apache CouchDb (u.a. BBC)
  - Couchbase (u.a. Adobe, BMW, Cisco, Zynga)
  - Lotus Notes
- Speicherung von Daten in Form von Dokumenten, häufig abgebildet im JSON-Format:

```
{
 title : 'This is a blog post',
 author : 'Peter',
 content : 'It's working!'
}
```

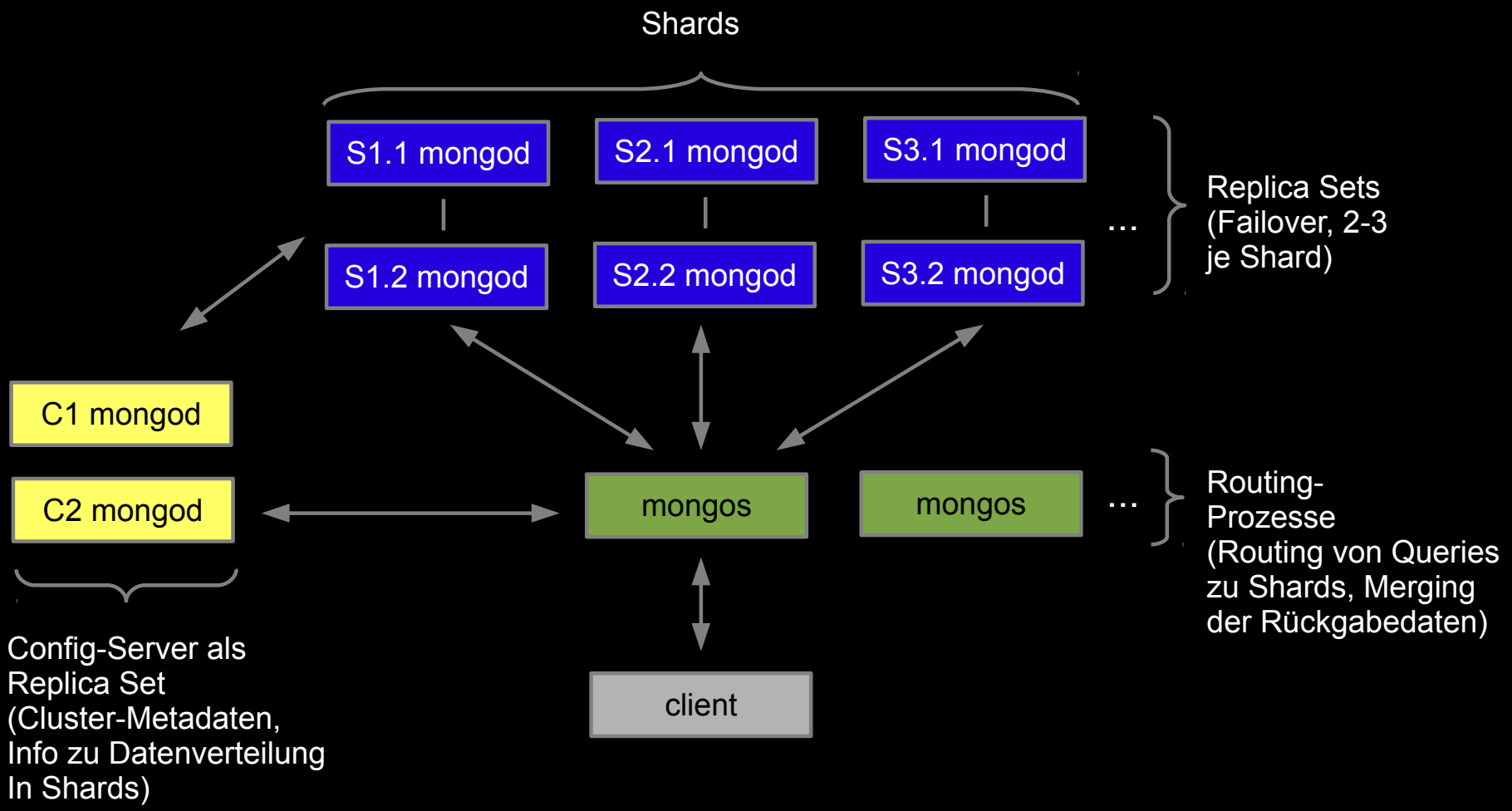
- Zusammenfassung der Dokumente in Collections / Views
- Abfragen adhoc oder über persistenten Ansichtsindex

MongoDB im Detail

- Implementiert in C++
- Sprachenunterstützung: u.a. Java, JavaScript, Python, PHP, Ruby
- gespeichert als binäres JSON: BSON
- Unterstützt Sharding
  - erfolgt über konfigurierbare Dokumentfelder
  - System balanciert Daten automatisch zwischen Shards aus
  - Shards können zur Laufzeit ergänzt werden
  - Wahl des Sharding-Keys essentiell für Performance!
  - Vermeidung von Hotspots, d.h. Servern mit hoher I/O-Last
  - gleichzeitig Indexaktualisierung nach Änderungen auf möglichst wenig Maschinen → Kompromiss finden

MongoDB im Detail

- Sharding-Architektur:



MongoDB im Detail

- Verbindung zu DB herstellen

```
Mongo connection = new Mongo(server,port);
//collection gets created automatically
DBCollection collection =
connection.getCollection(“posts”);
```

- Daten erfassen (JavaScript-Syntax)

```
collection.insert({
 title : 'This is a blog post',
 author : 'Karsten',
 ts: 1331915835960,
 content : 'It's working!'});
```


MongoDB im Detail

- Daten erfassen (Java-Syntax)

```
DBObject post = BasicDBObjectBuilder.start()
 .add("title", "This is a blog post")
 .add("author", "Karsten")
 .add("ts", 1331915835960)
 .add("content", "It's working!"))
 .get();

collection.insert(post);

System.out.println(post.get("_id"));
=> 4a9700dba5f9107c5cbc9a9c
```

- Oder verschiedene Konstruktoren von BasicDBObject

MongoDB im Detail

Daten aktualisieren

```
//$ commands for advanced query/update features
collection.update({_id : post._id},
{
 mod_ts : 1331915836050
 $push : {comments :
  {
      author : 'Tammo',
      comment : 'Cool!'
  }
}});
```

- Erstes Argument: Suchanfrage für Dokument
  zweites Argument: neue Feldwerte
- $push hängt Arraywerte an bzw. erstellt Array

MongoDB im Detail

- Daten auslesen
- Umfangreiche Query Language zur Datenselektion
  - $gt, $gte, $lt, $lte, $eq, $neq, $exists, $set, $mod, $where, $in, $nin, $inc, $push, $pull, $pop, $pushAll, $popAll

```
collection.find({ x : {$gt : 4}}) //x greather than 4
```

- Index-Erstellung für Dokumentwerte beschleunigt Zugriff auf / Suche in Collections

```
//create index for more performance:
collection.ensureIndex({"comments.author": 1})

commentsByPeter = collection.find({
        "comments.author" : "Peter"
});
```


MongoDB im Detail

- Daten lesen mit Paging

```
collection.ensureIndex({ "author" : 1,
 "ts" : -1,
 "comments.author" : 1 });

var cursor=collection.find({author:'Peter'})
  .sort({ts:-1})
  .skip(pageNum * resultsPerPage)
  .limit(resultsPerPage);

while (cursor.hasNext() ) {
 printArticle( cursor.next() );
}
```


Demo
MongoDB-Nutzung aus XPages-Anwendung

Vorteile

- Intuitive Datenstruktur, bereits bekannt von Lotus Notes
- einfache "natürliche" Modellierung von Anforderungen mit flexiblen Abfragefunktionen
  - Views per Map Reduce in Couch
  - Mongo Query Language
- MongoDB/CouchBase: Sharding für Big Data

Vorteile

- MongoDB
  - Abfragesprache ähnlich dynamisch wie SQL
  - GridFS als Aufsatz zu MongoDb zur Speicherung großer Binärdateien inkl. Sharding/Replikation im Cluster (max. Dokumentgröße sonst 4 MB)
  - Geo-spatial indexing built-in (Suche der nächsten Tankstelle zu [longitude,latitude])
- CouchDB
  - Offline-Synchronisation ähnlich wie Lotus Notes
- CouchBase Mobile
  - iOS/Android-Port der Datenbank, kommuniziert mit CouchBase Server in der Cloud

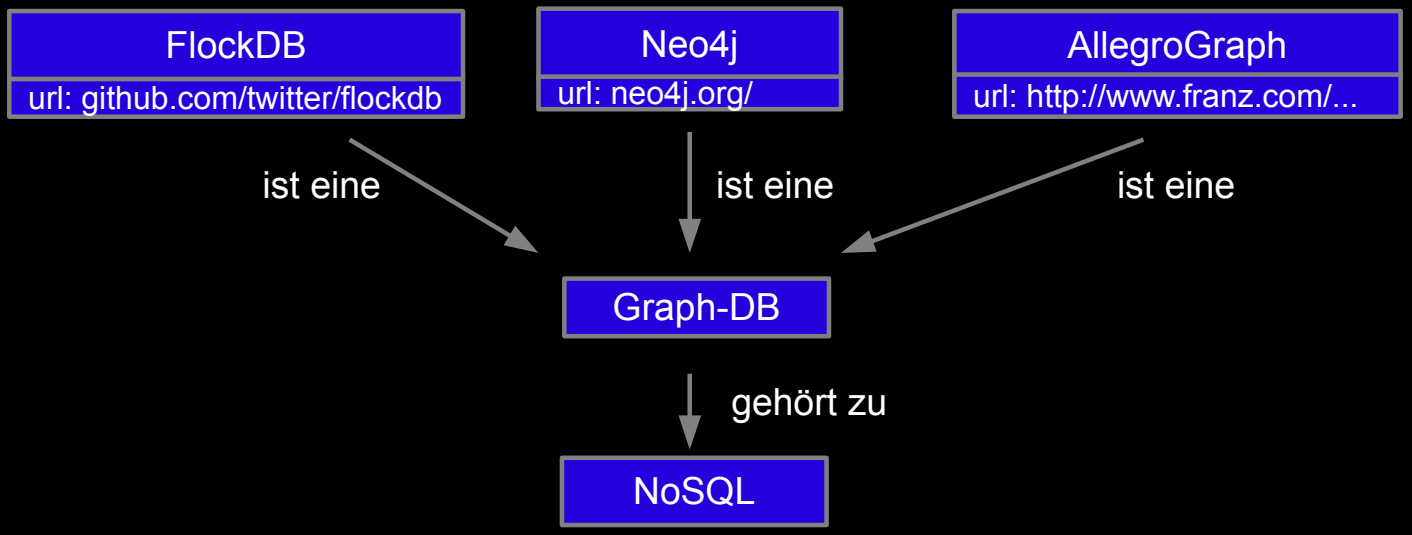
Nachteile

- CouchDB: Zukunft ungewiss, Entwicklerteam konzentriert sich auf neue CouchBase-DB
- MongoDB: keine Lösungen für Offline-Datenhaltung auf mobilen Geräten
- Gegenüber Spalten-basierter Datenbank vermutlich größere Hardware-Anforderungen wg. dynamischerer DB-Abfragen z.T. ohne Vorbereitung der Daten
- Gegenüber RDBMS redundante Speicherung von Daten (Denormalisierung) zugunsten höherer Performance

Graph-orientierte Datenbanken

Graph-orientierte Datenbanken

- Vertreter
  - FlockDB (entwickelt von Twitter für Social Graph)
  - Neo4j (u.a. Deutsche Telekom, Adobe, studiVZ, Cisco)
  - AllegroGraph (u.a. Kodak, Pfizer und in diversen Semantic Web-Projekten)
- Datenbanken optimiert für stark vernetzte Daten
- Graph aus Knoten (Nodes) und Beziehungen (Relationships) zwischen Knoten
- Knoten und Beziehungen können Eigenschaften haben



Neo4j im Detail

- Implementiert in Java
- Im Gegensatz zu bisherigen NoSQL-Beispielen ACID-konform
- Einfach einzubetten in eigene Anwendungen

```
EmbeddedGraphDatabase graphDb =
            new EmbeddedGraphDatabase( DB_STORAGEDIR );
//
//work with graphDb here
//
graphDb.shutdown();
```


Neo4j im Detail

DB-Zugriff gekapselt in Transaktionen

```
Transaction tx = graphDb.beginTx();
try {
    // create nodes and relationship
    firstNode = graphDb.createNode();
    firstNode.setProperty( "message", "Hello, " );
    secondNode = graphDb.createNode();
    secondNode.setProperty( "message", "World!" );

    relationship = firstNode.createRelationshipTo(
                    secondNode, RelTypes.KNOWS );
    relationship.setProperty( "message", "brave Neo4j " );

    // read data
    System.out.print( firstNode.getProperty( "message" ) );
    System.out.print( relationship.getProperty( "message" ) );
    System.out.print( secondNode.getProperty( "message" ) );

    tx.success();
}
finally { tx.finish(); }
```

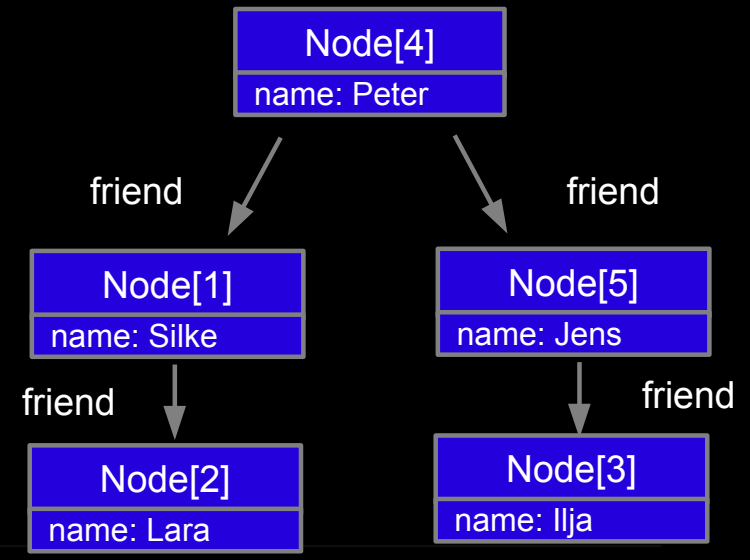

Neo4j im Detail

- API-Funktionen zur Traversierung des Graphs und zur Suche nach allen/den kürzesten Pfaden zwischen zwei Knoten
- Abfragesprachen: Gremlin (Groovy-basierte Traversierungssprache) und Cypher (deklarative Graph-Abfragesprache)
- Cypher: Finde alle Freunde von Peters Freunden, die nicht seine Freunde sind

```
START peter=node:node_auto_index(name = 'Peter')
MATCH peter-[:friend]->()-[:friend]->fof
RETURN peter, fof
```

- Ergebnis:

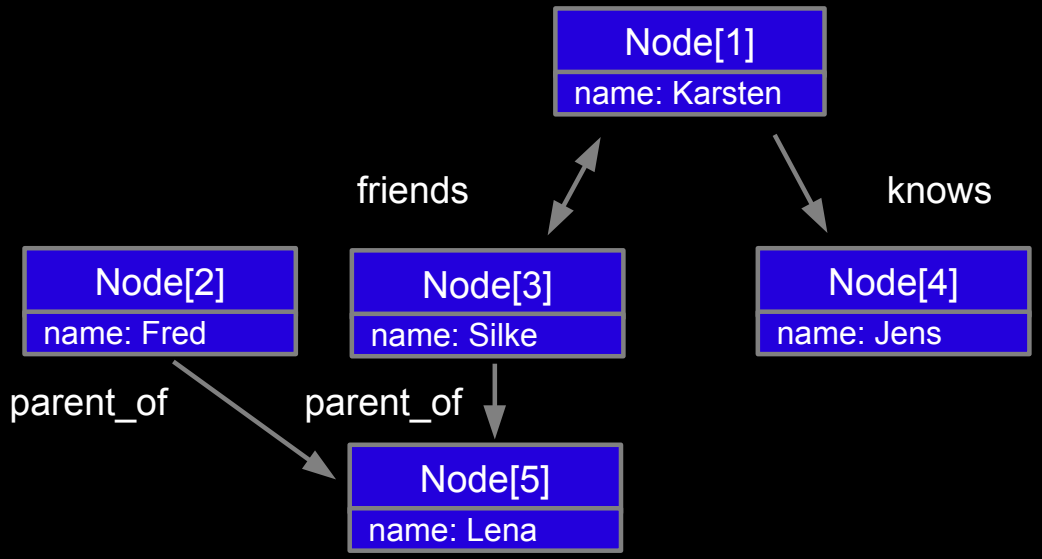
peter	fof
Node[4] {name->"Peter"}	Node[2] {name->"Lara"}
Node[4] {name->"Peter"}	Node[3] {name->"Ilja"}



Neo4j im Detail

- Cypher: Liefere alle Personen zurück, die ich kenne und ihre Kinder, falls sie welche haben:

```
START me=node(1)
MATCH me-->person-[?:parent_of]->children
RETURN person, children
```



Anwendungsfälle

- Modellierung sozialer Netzwerke
  - Freundesempfehlungen
- Prüfung von Zugriffsrechten
  - Person X - ist enthalten in → Gruppe A
  - Gruppe A - ist enthalten in → Gruppe B
  - Gruppe B - ist enthalten als Editor in → ACL von names.nsf
  - Auf welche Datenbanken hat der Nutzer welchen Zugriff?
- Produktempfehlungen
  - Welche Musik, die ich noch nicht gekauft habe, mögen meine Freunde?
- Routenplanung

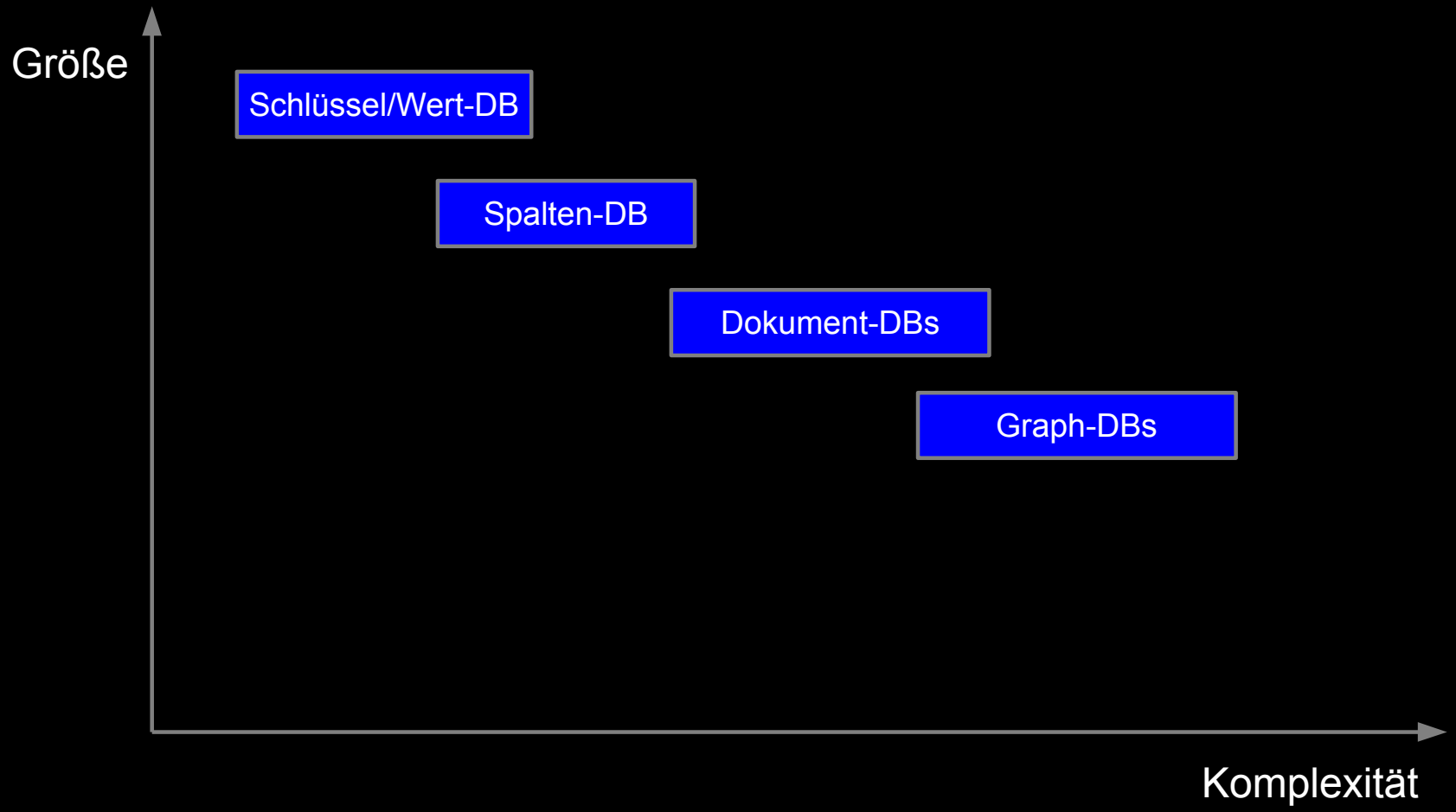
Demo
Neo4j eingebettet in XPages-Anwendung

Vor- und Nachteile

- Vorteile
  - Sehr kompakte Modellierung von vernetzten Daten
  - Große Performance-Vorteile gegenüber RDBMS je nach Anwendungsfall
  - Neo4j
    - ACID-Konformität
    - problemlose Einbettung in andere Anwendungen
- Nachteile
  - Neo4j
    - im Gegensatz zu FlockDB kein Sharding built-in, aber Master-Slave-Replication
    - „This can handle billions of entities…but not 100B“
    - Lizenzgebühren für OEM-Nutzung

Übersicht Größe/Komplexität NoSQL

- Mit zunehmender DB-Komplexität sinkt die Möglichkeit, die Daten effizient zu verteilen



- Einführung
- Was ist NoSQL?
- Klassen von NoSQL-Datenbanken
- Kundera-Bibliothek
- Bestandsaufnahme: Wo steht Lotus Notes?
- Zusammenfassung
- Q&A

Kundera – der gemeinsame Nenner

- JPA*-kompatibles Toolkit zur Speicherung von Daten in RDBMS, HBase, Cassandra und MongoDB
- simple Persistenz von Daten mit Hilfe annotierter POJOs**
- bildet JPA-Query-Language auf DB-Funktionen ab
  - verwendet Apache Lucene als externen Indexer für fehlende Funktionalität
- Wechsel der NoSQL-Datenbank lediglich eine Konfigurationsänderung

*Java Persistence API
**Plain Old Java Objects

Kundera – der gemeinsame Nenner

```
@Entity
@Table(name = "users", schema = "KunderaExamples@cassandra_pu")

public class User {
    @Id
    private String userId;
    @Column(name="fullname")
    private String fullName;

    public User() {

    }
    public String getUserId() {
        return userId;
    }
    public void setUserId(String userId) {
        this.userId = userId;
    }
    public String getFullName() {
        return fullName;
    }
    public void setFullName(String fullName) {
        this.fullName = fullName;
    }
}
```


Kundera – der gemeinsame Nenner

```
User user = new User();
user.setUserId("0001");
user.setFullName("Hans Müller");

EntityManagerFactory emf =
       Persistence.createEntityManagerFactory("cassandra_pu");
EntityManager em = emf.createEntityManager();


//save object
em.persist(user);

//load object
User anotherUser = em.find(User.class, "0002");


em.close();
emf.close();
```


Kundera – Vorteile und Nachteile

- Vorteil
  - investitionssichere Entwicklung wg. Unabhängigkeit von Datenbank
  - umfangreiche Queries auch bei Spalten-orientierter DB, Lucene-Index speicherbar in Cassandra selbst
  - vereinfachter Datentransfer zwischen NoSQL-Datenbanken
- Nachteil
  - weniger Einfluss auf konkrete Datenspeicherung; ggf. kleinster gemeinsamer Nenner der Features
  - Kundera Open Source, aber Community eher klein

- Einführung
- Was ist NoSQL?
- Klassen von NoSQL-Datenbanken
- Kundera-Bibliothek
- Bestandsaufnahme: Wo steht Lotus Notes?
- Zusammenfassung
- Q&A

Was fehlt Lotus Notes, was die anderen können?

- Speicherung von mehr als 16/32/64 KB Daten in Feldern
- Zeitgemäßige Java-API
  - designed für Performance
  - kein Recycle
  - Java Generics
  - Attachment-Streaming
  - Zugriff auf Transaktionen im Code
- Adhoc-Datenbankabfragen
  - UnQL, Mongo Query Language, Map Reduce
- Web 2.0-kompatibles Lizenzmodell für Startups
- Big Data-Support: Beseitigung des 64 GB-Limits, Sharding
- Geo-spatial Indexing

Was kann Lotus Notes, was die anderen nicht können?

- NSF
  - Speichern von Attachments in Dokumenten built-in

- Sonstiges
  - All-in-one-Lösung brauchbarer Einzelkomponenten
  - eigenes Benutzerverzeichnis
  - HTTP-Server
  - Volltext-Engine inkl. Indizierung von Dateiformaten
  - Mail-Server / Calendaring and Scheduling
  - Applikationsserver
  - Lese-/Schreibrechte auf Dokumentebene
  - Verschlüsselung

- Einführung
- Was ist NoSQL?
- Klassen von NoSQL-Datenbanken
- Kundera-Bibliothek
- Bestandsaufnahme: Wo steht Lotus Notes?
- Zusammenfassung
- Q&A

Zusammenfassung

- Viel Dynamik im NoSQL-Markt
  - kein Produkt ist perfekt
- Vergleich von NSF mit anderen Datenbanken ernüchternd
  - aber Lotus Notes ist weit mehr als nur eine Datenbank
- Aufgrund der Vielzahl an Systemkomponenten eher ein schweizer Taschenmesser als ein Hammer
- Auswahl der DB-Plattform kein Entweder-Oder
  - finden des richtigen Werkzeugs für einen Anwendungsfall
  - auch Kombinationen von DBs sind möglich!
- Nicht zu vergessen SQL:
  wer möchte schon Eventual Consistency
  bei Finanztransaktionen? :-)

- Einführung
- Was ist NoSQL?
- Klassen von NoSQL-Datenbanken
- Kundera-Bibliothek
- Bestandsaufnahme: Wo steht Lotus Notes?
- Zusammenfassung
- Q&A

Vielen Dank!

Zeit für Fragen