TECHNISCHE UNIVERSITÄT WIEN
VIENNA UNIVERSITY OF TECHNOLOGY

Persistenz

Software Engineering & Projektmanagement VO
(188.410)

Richard Mordinyi
Richard.mordinyi@tuwien.ac.at

Agenda

- Datenmanagement
- Relationale Datenbanken
- Datenbankzugriff
- Objektorientierte Datenbanken
- XML Datenbanken
- NoSQL Datenbanken

Persistenz

- Dauerhaftes Speichern von Daten
  - Benutzerdaten, Kundendaten
  - Zustand der Software
  - ...

Persistenz - Anforderungen

- Datenstruktur
- Kenngrößen
- Zuverlässigkeit
- Legacy
- Aufwand

Anforderung - Datenstruktur

- Binär

```
00000000   AC ED 00 05  73 72 00 06  50 65 72 73  6F 6E D9 E4  ....sr..Person..
00000010   08 EE 44 CD  CB 60 02 00  02 4C 00 09  66 69 72 73  ..D..`...L..firs
00000020   74 6E 61 6D  65 74 00 12  4C 6A 61 76  61 2F 6C 61  tnamet..Ljava/la
00000030   6E 67 2F 53  74 72 69 6E  67 3B 4C 00  08 6C 61 73  ng/String;L..las
00000040   74 6E 61 6D  65 71 00 7E  00 01 78 70  74 00 03 4D  tnameq.~..xpt..M
00000050   61 78 74 00  0A 4D 75 73  74 65 72 6D  61 6E 6E     axt..Mustermann
```

- Semi-strukturiert

```
<person>
    <firstname>Max</firstname>
    <lastname>Mustermann</lastname>
</person>
```

XML

```
{
   "firstname":"Max",
   "lastname":"Mustermann"
}
```

JSON

- Strukturiert (Datenbank)

Binär - Serialisieren

- Von der Sprache bereitgestellt

```
1.  public static void main(String[] args) {
2.      Person person = new Person("Max", "Mustermann");
3.      File file = new File("max.mustermann.bin");
4.      ObjectOutputStream stream = new ObjectOutputStream(new
    FileOutputStream(file));
5.      stream.writeObject(person);
6.      stream.close();
7.  }
```

```
00000000   AC ED 00 05  73 72 00 06  50 65 72 73  6F 6E D9 E4  ....sr..Person..
00000010   08 EE 44 CD  CB 60 02 00  02 4C 00 09  66 69 72 73  ..D..`...L..firs
00000020   74 6E 61 6D  65 74 00 12  4C 6A 61 76  61 2F 6C 61  tnamet..Ljava/la
00000030   6E 67 2F 53  74 72 69 6E  67 3B 4C 00  08 6C 61 73  ng/String;L..las
00000040   74 6E 61 6D  65 71 00 7E  00 01 78 70  74 00 03 4D  tnameq.~..xpt..M
00000050   61 78 74 00  0A 4D 75 73  74 65 72 6D  61 6E 6E     axt..Mustermann
```

Binär – spezifische Serialisierung

- Selbst definiertes Dateiformat

```
// ...
public NetworkStatsHistory(DataInputStream in) throws
IOException {
    final int version = in.readInt();
    switch (version) {
        case VERSION_INIT: {
            bucketDuration = in.readLong();
            bucktStart = readFullLongArray(in);
            rxBytes = readFullLongArray(in);
            rxPackets = new long[bucketStart.length];
            txBytes = readFullLongArray(in);
            txPackets = new long[bucketStart.length];
            operations = new long[bucketStart.length];
            bucketCount = bucketStart.length;
            break;
        }
// ...
```

(https://android.googlesource.com/platform/frameworks/base.git)

Anforderung - Kenngrößen

- (erwartete) Größe der Datenbank
- Anzahl der Anwender
- Leistungserwartung
- Transaktionen / Sek
- ….

Anforderung - Zuverlässigkeit

- Integrität
- Verfügbarkeit
- Möglichkeiten im Fehlerfall
- Backup Szenarien

Anforderung - Legacy

- Vorhandene Datenbanksysteme / Infrastrukturen
  - Aufwand für Einführung neuer Technologien
  - Aufwand für Administration
  - Akzeptanz
- Vorhandene Datenbestände
- Interaktionsmöglichkeiten mit bestehenden Legacy Systemen

Anforderung - Aufwand

- Entwicklungszeit
- Administration
- Wartung

Welche Datenbank löst nun mein Problem?

- Welche Datenbanktype
  - relational Database (zB.: Postgres)
  - key-value store (zB.: Riak, Redis)
  - column-oriented database (zB.: HBase)
  - document-oriented databases (zB.: MongoDB, CouchDB)
  - graph database (zB.: Neo4J)

- Was sind die treibenden Kräfte?
  - Wurden für spezielle Probleme in reallen Andwendungsfällen entwickelt

Welche Datenbank löst nun mein Problem?

- Welche Datenbanktype?
  - relational Database (zB.: Postgres)
  - key-value store (zB.: Riak, Redis)
  - column-oriented database (zB.: HBase)
  - document-oriented databases (zB.: MongoDB, CouchDB)
  - graph database (zB.: Neo4J)
- Was sind die treibenden Kräfte?
  - Wurden für spezielle Probleme in reallen Andwendungsfällen entwickelt
- Leistung, Skalierbarkeit?

File Persistence

Vorteile:

- einfach zu implementieren
- weniger externe Abhängigkeiten
- erlaubt spezifische Optimierungen
- Einfache Wartung

Nachteile:

- Schwierig zu durchsuchen
- geringe Interoperabilität
- geringe Portabilität
- geringe Skalierbarkeit

Relationale Datenbanken (RDBM)

- meistverbreitete Art von Datenbanksystemen
  - Tabellen, Zeilen, Spalten
  - Werte sind typisiert
    - Numeric, strings, dates, uninterpreted blobs...
- zahlreiche Implementierungen für verschiedene Szenarien
- Unterstützung für viele Plattformen und Sprachen
- Tool support
  - Clients (DBVisualizer, SquirrelSQL)
  - Reporting (Crystal, Jasper)
- SQL als weitgehend standardisierte Schnittstelle

Freie Implementierungen

- MySQL (GPLv2)
- PostgreSQL (BSD License)
- Firebird (Mozilla License)
- H2
  - MPL 1.1 (Mozilla Public *License*)
  - (unmodified) EPL 1.0 (Eclipse Public *License*)
- HSQLDB (BSD License)
- SQLite (public domain)

http://dilbert.com/strips/comic/1995-11-17/

Design-first Datastore

- Tabelle
  - Name
  - Liste an Spalten und deren Typen

```
CREATE TABLE countries (
  country_code char(2) PRIMARY KEY,
  country_name text UNIQUE
);
```

- Einschränkungen
  - PRIMARY KEY
  - UNIQUE

```
CREATE TABLE cities (
  name text NOT NULL,
  postal_code varchar(9) CHECK (postal_code <> ''),
  country_code char(2) REFERENCES countries,
  PRIMARY KEY (country_code, postal_code)
);
```

CRUD-Operationen

- Daten einfügen

```
INSERT INTO cities
VALUES ('Toronto','M4C1B5','ca');
```

- Daten aktualisieren

```
UPDATE cities
SET postal_code = '97205'
WHERE name = 'Portland';
```

- Daten abfragen

```
SELECT cities.*, country_name
FROM cities INNER JOIN countries
  ON cities.country_code = countries.country_code;
```

Indexing

- Geschwindigkeit einer RDBMS hängt von
  - Verwaltung der Daten
  - Minimierung von Disk-Leseoperationen
  - Optimierung von Abfragen und deren Ausführung

- Indizes vermeiden die Überprüfung der gesamten Tabelle
  - Map, B-Tree

```
CREATE TABLE / PRIMARY KEY will create implicit index "events_pkey" \
for table "events"
```

Transactions

- Atomic
- Consistent
- Isolated
- durable

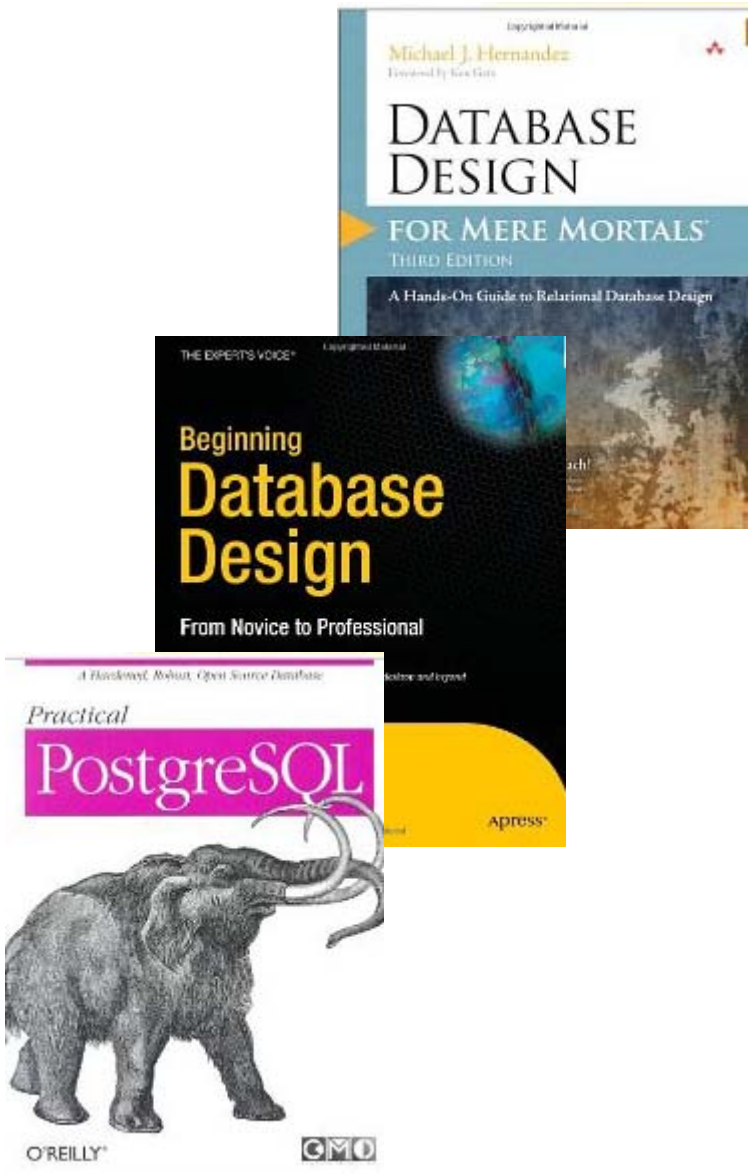
```
BEGIN TRANSACTION;
  UPDATE account SET total=total+5000.0 WHERE account_id=1337;
  UPDATE account SET total=total-5000.0 WHERE account_id=45887;
END;
```

Additional Features

- Stored Procedures
- Triggers
- Views
- Fuzzy Searching
  - LIKE
  - REGEX
  - Levenshtein
- Etc.

Enterprise Features

- Automatisches Optimieren von Queries
- Scalability/Clustering
- Backup im laufenden Betrieb
- Feingranulares Rechtemanagement
- Beispiele:
  - Oracle
  - Microsoft SQL
  - IBM DB2



Embedded Datenbanken

- Serverlos
- Lightweight
  - Tests
  - Development setups
  - Mobile Plattformen
- Daten meist in einzelner Datei
- Single-user
- Beispiele
  - SQLite
  - HSQLDB
  - H2

Datenbankzugriff

- Datenbankspezifische APIs
- Low-level interfaces
  - JDBC, ODBC, ADO.NET
- Utility Libraries
  - Spring JDBC Templates

Impediance mismatch

- In objekt-orientierten Architekturen
- Bruch zwischen Datenmodellen
  - Tabelle ↔ Objekt

Impediance mismatch

- In objekt-orientierten Architekturen
- Bruch zwischen Datenmodellen
  - Tabelle ↔ Objekt
- Keine Vererbung in RDBMs

Impediance mismatch

- In objekt-orientierten Architekturen
- Bruch zwischen Datenmodellen
  - Tabelle ↔ Objekt
- Keine Vererbung in RDBMs
- Identität von RDBM Datensätzen

Impediance mismatch

- In objekt-orientierten Architekturen
- Bruch zwischen Datenmodellen
  - Tabelle ↔ Objekt
- Keine Vererbung in RDBMs
- Identität von RDBM Datensätzen
- Kapselung

Objektrelationales (O/R) Mapping

- Übersetzen der Daten Tabelle ↔ Objekt
- Abbildung von Relationen
- Abbildung von Vererbung
- Abbildung der Navigation
- Schema Migration

O/R Mapping - Varianten

- Manuell
- Data Mapper (MyBatis, ehem. Apache iBatis)
- „Full blown“ O/R Mapper

JDBC Example

```
1. public static void main(String[] args){
2.     Class.forName("com.mysql.jdbc.Driver").newInstance();
3.     String url = "jdbc:mysql://localhost/foodb";
4.     connection = DriverManager.getConnection(url, "username", "password");
5.
6.     String query = "SELECT firstname,lastname FROM Person WHERE id = ?";
7.     PreparedStatement st = connection.prepareStatement(query);
8.     st.setInt(1, 42);
9.     ResultSet rs = st.executeQuery(query);
10.    rs.first();
11.    Person p = new Person(
12.        rs.getString("firstname")
13.        rs.getString("lastname"));
14.
15.    conn.close();
16.}
```

Spring JDBC Templates

Action	Spring	You
Define connection parameters.		X
Open the connection.	X	
Specify the SQL statement.		X
Declare parameters and provide parameter values		X
Prepare and execute the statement.	X	
Set up the loop to iterate through the results (if any).	X	
Do the work for each iteration.		X
Process any exception.	X	
Handle transactions.	X	
Close the connection, statement and resultset.	X	

http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/jdbc.html

Spring JDBC Example (1)

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <beans xmlns="http://www.springframework.org/schema/beans" ...>
3.
4.    <bean id="dataSource"
5.          class="org.apache.commons.dbcp.BasicDataSource"
6.          destroy-method="close">
7.      <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
8.      <property name="url" value="jdbc:mysql://localhost/foodb"/>
9.      <property name="username" value="username"/>
10.     <property name="password" value="password"/>
11.   </bean>
12.
13.   <bean id="app" class="com.example.MyApp">
14.     <property name="dataSource" ref="dataSource"/>
15.   </bean>
16.
17. </beans>
```

Spring JDBC Example (2)

```
1.  public class App{
2.
3.    private JdbcTemplate jdbcTemplate;
4.
5.    public Person getPerson(Long id){
6.      String query = "SELECT firstname,lastname FROM Person WHERE id = ?";
7.      Object[] parameters = new Object[]{ id };
8.
9.      RowMapper mapper = new RowMapper<Person>(){
10.       public Person mapRow(ResultSet rs, int rowNum) throws SQLException {
11.         return new Person(rs.getString("firstname"),
12.                           rs.getString("lastname"));
13.       }
14.     }
15.
16.     List<Person> persons = template.queryForObject(query, parameters,
17.                                                    mapper);
18.     return persons.get(0);
19.   }
20.
21. }
```

Data Mapper

- Verwaltet Mappings zwischen
  - SQL ↔ Objekt
  - Stored Procedure ↔ Objekt
- SQL-Statements extrahiert → Konfiguration
- Implementierung: MyBatis (früher iBatis)

MyBatis - Beispiel

```xml
<mapper namespace="my.app.PersonMapper">

  <select id="getPerson" parameterType="long" resultType="Person">
    SELECT firstname, lastname FROM Person
    WHERE id = #{id}
  </select>

</mapper>
```

PersonMapper.xml

```java
public interface PersonMapper {

  Person getPerson(Long id);

}
```

PersonMapper.java

```xml
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
  <property name="basePackage" value="my.app" />
</bean>
```

spring-context.xml

„Full-blown“ O/R Mapper

- Direktes Mapping zwischen Objekten und Tabellen
- SQL-Statements werden generiert
- Anwendung weiß nichts von der Datenbank
- Beispiele:
  - Hibernate (NHibernate)
  - EclipseLink
  - OpenJPA

Java Persistence API (JPA)

- Vereinheitlichung von O/R Mapper Interfaces
- Ursprung in JBoss Hibernate/Oracle Toplink
- Beschreibung von Entities als POJOs
- Beschreibung der Metadaten mittels Annotationen oder separatem XML-File
- Implementierungen:
  - Hibernate
  - EclipseLink (JPA 2.0 Referenz-Implementierung)
  - Apache OpenJPA

JPA Example (1)

```
1.  <persistence xmlns="http://java.sun.com/xml/ns/persistence" version="2.0"
    ...>
2.     <persistence-unit name="myapp">
3.        <class>my.app.Person</class>
4.
5.        <provider>
6.          org.apache.openjpa.persistence.PersistenceProviderImpl
7.        </provider>
8.        <properties>
9.          <property name="openjpa.ConnectionDriverName"
10.                   value="com.mysql.jdbc.Driver" />
11.         <property name="openjpa.ConnectionURL"
12.                   value="jdbc:mysql://localhost/foodb" />
13.         <property name="openjpa.ConnectionUserName" value="username" />
14.         <property name="openjpa.ConnectionPassword" value="password" />
15.       </properties>
16.    </persistence-unit>
17. </persistence>
```

META-INF/persistence.xml

JPA Example (2)

```java
@Entity
public class Person {

    @Id
    private Long id;
    private String firstname;
    private String lastname;

    // getters and setters
}
```

Person.java

```java
public static void main(String[] args){
  EntityManagerFactory emf;
  emf = Persistence.createEntityManagerFactory("myapp");
  entityManager = emf.createEntityManager();
  Person p = entityManager.find(Person.class, 42);
}
```

App.java

Query in JPA

- Query by Example (Criteria-API)
- Query Language (JPQL)
- Wrapper Libraries (Querydsl)
- Native Queries

Example - Criteria API

```
1. public static void main(String[] args){
2.   entityManager = ...
3.   CriteriaBuilder cb = entityManager.getCriteriaBuilder();
4.   CriteriaQuery<Person> query = cb.createQuery(Person.class);
5.   Root<UserData> root = query.from(Person.class);
6.   query.where(cb.equal(root.get("id"), 42));
7.   query.select(root);
8.   TypedQuery q = entityManager.createTypedQuery(query, Person.class)
9.   Person p = q.getSingleResult();
10.}
```

Example - JPQL

```
1.public static void main(String[] args){
2.   entityManager = ...
3.   String jpqlQuery = "SELECT p FROM Person p WHERE p.id=42";
4.   Query q = entityManager.createQuery(jpqlQuery);
5.   Person p = (Person) q.getSingleResult();
6.}
```

O/R Mapper - Kritik

- Erhöht Komplexität (Overkill?)
- „Schwarze Magie“
- Generierte Queries oft langsamer
- Probleme mit Permissions
- Domain != Database model
- Schema Ownership Problem

Objektorientierte Datenbanken

Objektorientierte Datenbanken

- Keine Definition von Datenmodellen notwendig
  - Verwendung des Domänenmodells
- Finden Verwendung in Marktnischen
  - Embedded Systems
  - z.B. db4o, Cache
- Kein Mapping notwendig
  - Nah an Objekt-orientierter Sprache
- „Enterprise“-fähig
  - Transaktionen
  - Performance
  - Clustering

Objektorientierte Datenbanken

- Beispiel – db4o
- Kommerzielle Lizenz
  - Db4o GPL Lizenz
    - kaum für einen produktiven Einsatz geeignet

- Sehr einfach zu lernen und einzusetzen
- Operationen
  - Beispiele von der Website
  - Objekt anlegen
  - Objekt suchen
  - Objekt aktualisieren
  - Objekt löschen

OODB - Klassen

```
package com.db4odoc.f1.chapter1;
public class Pilot {
    private String name;
    private int points;

    public Pilot(String name,int points) {
        this.name=name;
        this.points=points;
    }

    public int getPoints() {
        return points;
    }

    public void addPoints(int points) {
        this.points+=points;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return name+"/"+points;
    }
}
```

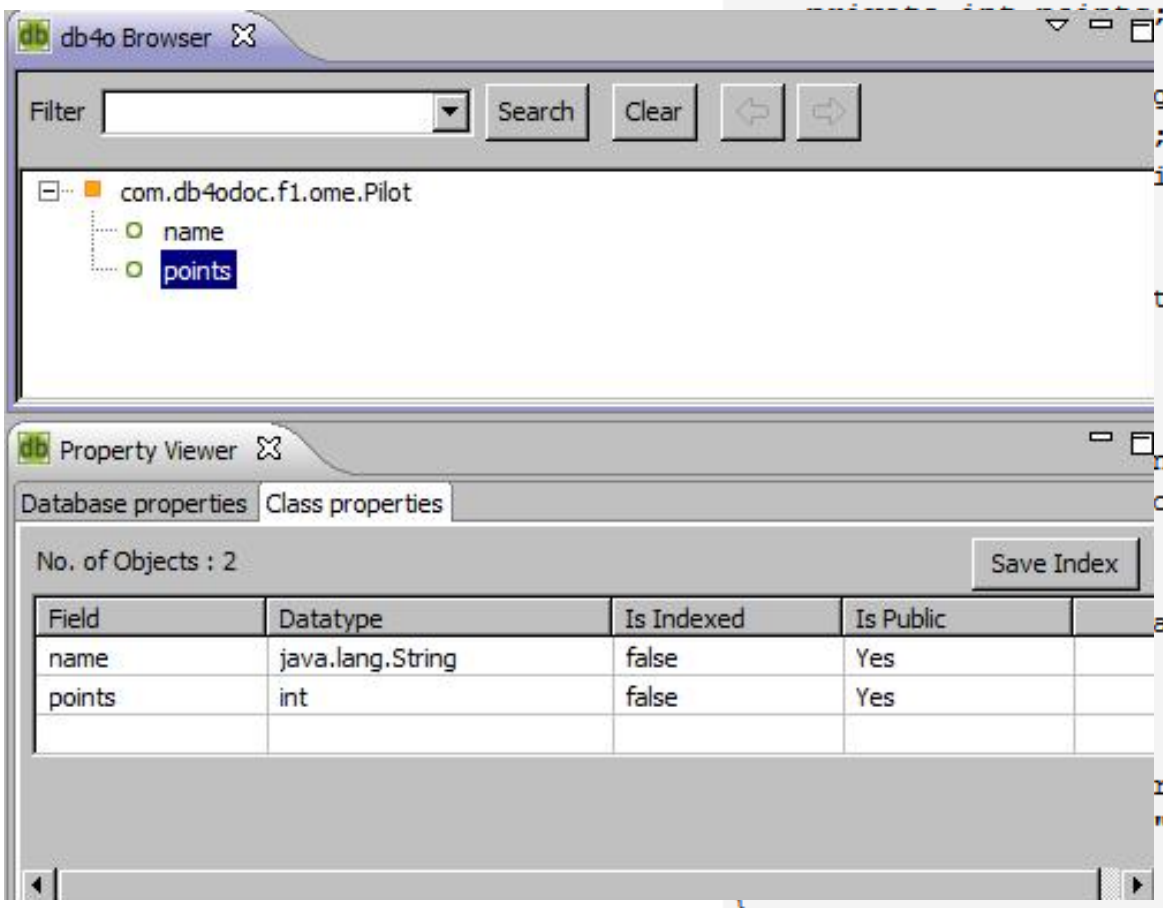
OODB – Objekt anlegen

```
// accessDb4o
ObjectContainer db = Db4oEmbedded.openFile(Db4oEmbedded
        .newConfiguration(), DB4OFILENAME);
try {
    // do something with db4o
} finally {
    db.close();
}
```

```
// storeFirstPilot
Pilot pilot1 = new Pilot("Michael Schumacher", 100);
db.store(pilot1);
System.out.println("Stored " + pilot1);
```

```
// storeSecondPilot
Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
db.store(pilot2);
System.out.println("Stored " + pilot2);
```

OODB – Objekt anlegen



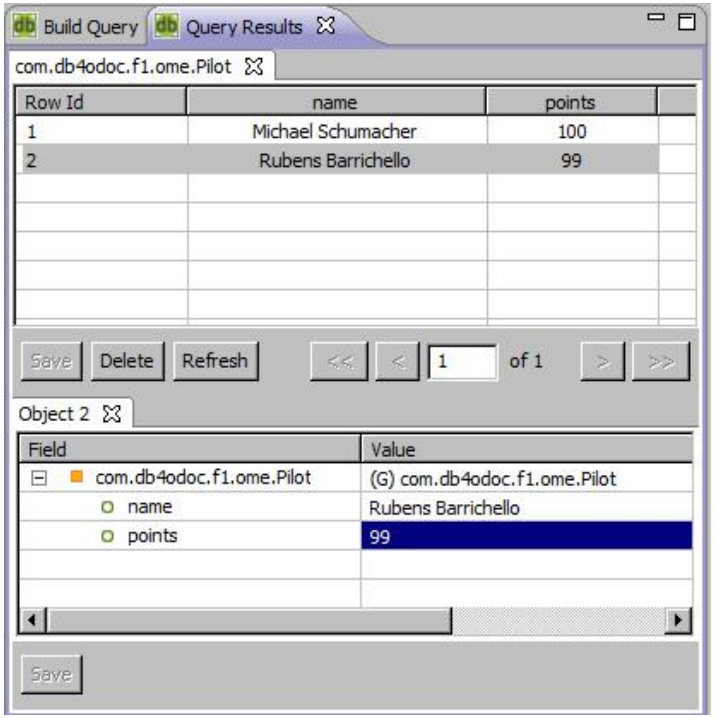

```
package com.db4odoc.f1.chapter1;
public class Pilot {
    private String name;
                                    g name,int points) {
                                    ;
                                    ints;


                                    ts() {


                                    nts(int points) {
                                    oints;

                                    ame() {


                                    ring() {
                                    "+points;
}
```

OODB – Objekte suchen

- Query by Example (QBE)
  - Prototype

```
// retrieveAllPilotQBE
Pilot proto = new Pilot(null, 0);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

```
// retrieveAllPilots
ObjectSet result = db.queryByExample(Pilot.class);
listResult(result);
```




```
// retrievePilotByName
Pilot proto = new Pilot("Michael Schumacher", 0);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

```
// retrievePilotByExactPoints
Pilot proto = new Pilot(null, 100);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

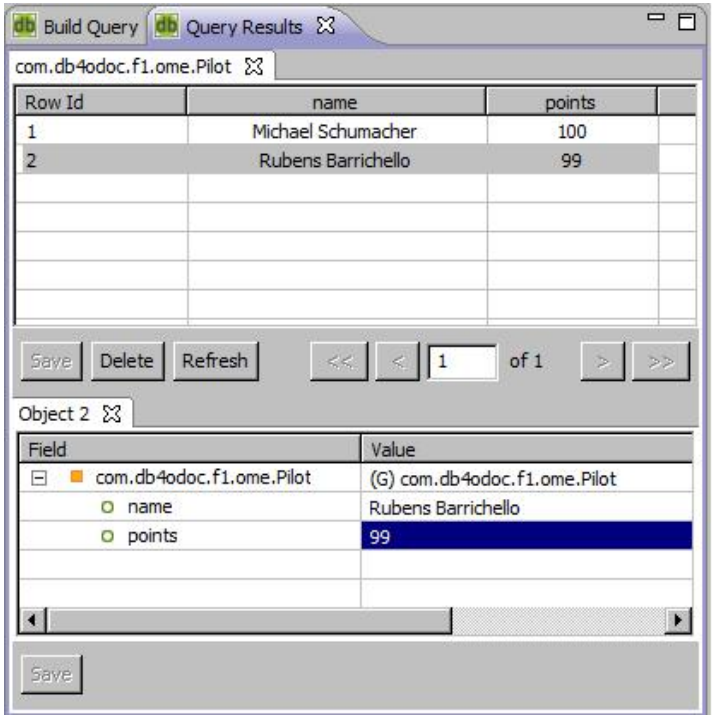
- Suche nach Prototype mit identen Werten der Variablen
  - Default-Werte werde nicht berücksichtigt

OODB – Objekte suchen

- Query by Example (QBE)
  - Prototype

```
// retrieveAllPilotQBE
Pilot proto = new Pilot(null, 0);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

```
// retrieveAllPilots
ObjectSet result = db.queryByExample(Pilot.class);
listResult(result);
```




```
// retrievePilotByName
Pilot proto = new Pilot("Michael Schumacher", 0);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

```
// retrievePilotByExactPoints
Pilot proto = new Pilot(null, 100);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

- Prototype muss alle Variablen beinhalten
- keine komplexen Abfragen (AND, OR, NOT, etc.)
- Keine Abfrage mit default-Werten

OODB – Objekte suchen

- Native Queries (NQ)
  - db4o query interface

```
List <Pilot> pilots = db.query(new Predicate<Pilot>() {
    public boolean match(Pilot pilot) {
        return pilot.getPoints() == 100;
    }
});
```

  - Methode „match“ returniert „true“, falls Instanz zur Ergebnismenge gehört

```
// storePilots
db.store(new Pilot("Michael Schumacher",100));
db.store(new Pilot("Rubens Barrichello",99));
```

```
List <Pilot> result = db.query(new Predicate<Pilot>() {
    public boolean match(Pilot pilot) {
        return pilot.getPoints() > 99
            && pilot.getPoints() < 199
            || pilot.getName().equals("Rubens Barrichello");
   }
});
```

OODB – Objekt aktualisieren

- Objekt muss bekannt sein

```
// updatePilot
ObjectSet result = db
        .queryByExample(new Pilot("Michael Schumacher", 0));
Pilot found = (Pilot) result.next();
found.addPoints(11);
db.store(found);
System.out.println("Added 11 points for " + found);
retrieveAllPilots(db);
```

OODB – Objekt löschen

- Objekt muss bekannt sein

```
// deleteFirstPilotByName
ObjectSet result = db
        .queryByExample(new Pilot("Michael Schumacher", 0));
Pilot found = (Pilot) result.next();
db.delete(found);
System.out.println("Deleted " + found);
retrieveAllPilots(db);
```

OODB - Klassen

```
package com.db4odoc.f1.chapter2;
public class Car {
    private String model;
    private Pilot pilot;

    public Car(String model) {
        this.model=model;
        this.pilot=null;
    }

    public Pilot getPilot() {
        return pilot;
    }

    public void setPilot(Pilot pilot) {
        this.pilot = pilot;
    }

    public String getModel() {
        return model;
    }

    public String toString() {
        return model+"["+pilot+"]";
    }
}
```

OODB – Objekt anlegen

```
// storeFirstCar
Car car1 = new Car("Ferrari");
Pilot pilot1 = new Pilot("Michael Schumacher", 100);
car1.setPilot(pilot1);
db.store(car1);
```

```
// storeSecondCar
Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
db.store(pilot2);
Car car2 = new Car("BMW");
car2.setPilot(pilot2);
db.store(car2);
```

OODB – Objekte suchen (QBE)

```
// retrieveAllCarsQBE
Car proto = new Car(null);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

```
// retrieveAllPilotsQBE
Pilot proto = new Pilot(null, 0);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

```
// retrieveCarByPilotQBE
Pilot pilotproto = new Pilot("Rubens Barrichello", 0);
Car carproto = new Car(null);
carproto.setPilot(pilotproto);
ObjectSet result = db.queryByExample(carproto);
listResult(result);
```

OODB – Objekte suchen (NQ)

```
// retrieveCarsByPilotNameNative
final String pilotName = "Rubens Barrichello";
List<Car> results = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return car.getPilot().getName().equals(pilotName);
    }
});
listResult(results);
```

OODB – Objekt aktualisieren

```
// updateCar
List<Car> result = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return car.getModel().equals("Ferrari");
    }
});
Car found = (Car) result.get(0);
found.setPilot(new Pilot("Somebody else", 0));
db.store(found);
result = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return car.getModel().equals("Ferrari");
    }
});
listResult(result);
```

```
// updatePilotSingleSession
List<Car> result = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return car.getModel().equals("Ferrari");
    }
});
Car found = result.get(0);
found.getPilot().addPoints(1);
db.store(found);
result = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return car.getModel().equals("Ferrari");
    }
});
listResult(result);
```

```
// updatePilotSeparateSessionsPart1
List<Car> result = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return car.getModel().equals("Ferrari");
    }
});
Car found = result.get(0);
found.getPilot().addPoints(1);
db.store(found);
```

```
// updatePilotSeparateSessionsPart2
List<Car> result = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return car.getModel().equals("Ferrari");
    }
});
listResult(result);
```

- default update depth für alle Instanzen ist 1
  - Primitive und String werden aktualisiert

OODB – Objekt löschen

```
// deleteFlat
List<Car> result = db.query(new Predicate<Car>()
    public boolean match(Car car) {
        return car.getModel().equals("Ferrari");
    }
});
Car found = result.get(0);
db.delete(found);
result = db.queryByExample(new Car(null));
listResult(result);
```

```
// retrieveAllPilotsQBE
Pilot proto = new Pilot(null, 0);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

```
// deleteDeep
EmbeddedConfiguration config = Db4oEmbedded.newConfiguration();
config.common().objectClass(Car.class).cascadeOnDelete(true);
ObjectContainer db = Db4oEmbedded.openFile(config, DB4OFILENAME);
List<Car> result = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return car.getModel().equals("BMW");
    }
});
if (result.size() > 0) {
    Car found = result.get(0);
    db.delete(found);
}
result = db.query(new Predicate<Car>() {
    public boolean match(Car car) {
        return true;
    }
});
listResult(result);
db.close();
```

```
// retrieveAllPilots
Pilot proto = new Pilot(null, 0);
ObjectSet result = db.queryByExample(proto);
listResult(result);
```

- Rekrusives Löschen

XML Datenbanken

XML Datenbanken

- Definiert ein Model für ein XML Dokument
- Definiert nicht das physische Speichermodel
- Verwaltung von Informationen in XML Format
  - Relationale Datenbanksysteme
  - Native XML Datenbanksysteme
- Implementierungen
  - Xindice
  - eXist
  - BaseX
  - Berkeley DB XML, …

XML Datenbanken

- Native XML Datenbanksysteme
  - Bewahren physische Struktur
  - Speichern XML Dateien ohne das Schema kennen zu müssen
  - Zugriff mittels XML-basierter Technologien und XML-spezifische APIs
    - Xpath, Xquery, XSLT
    - XQJ or XML:DB API
  - Geringe Leistung

NoSQL Datenbanken

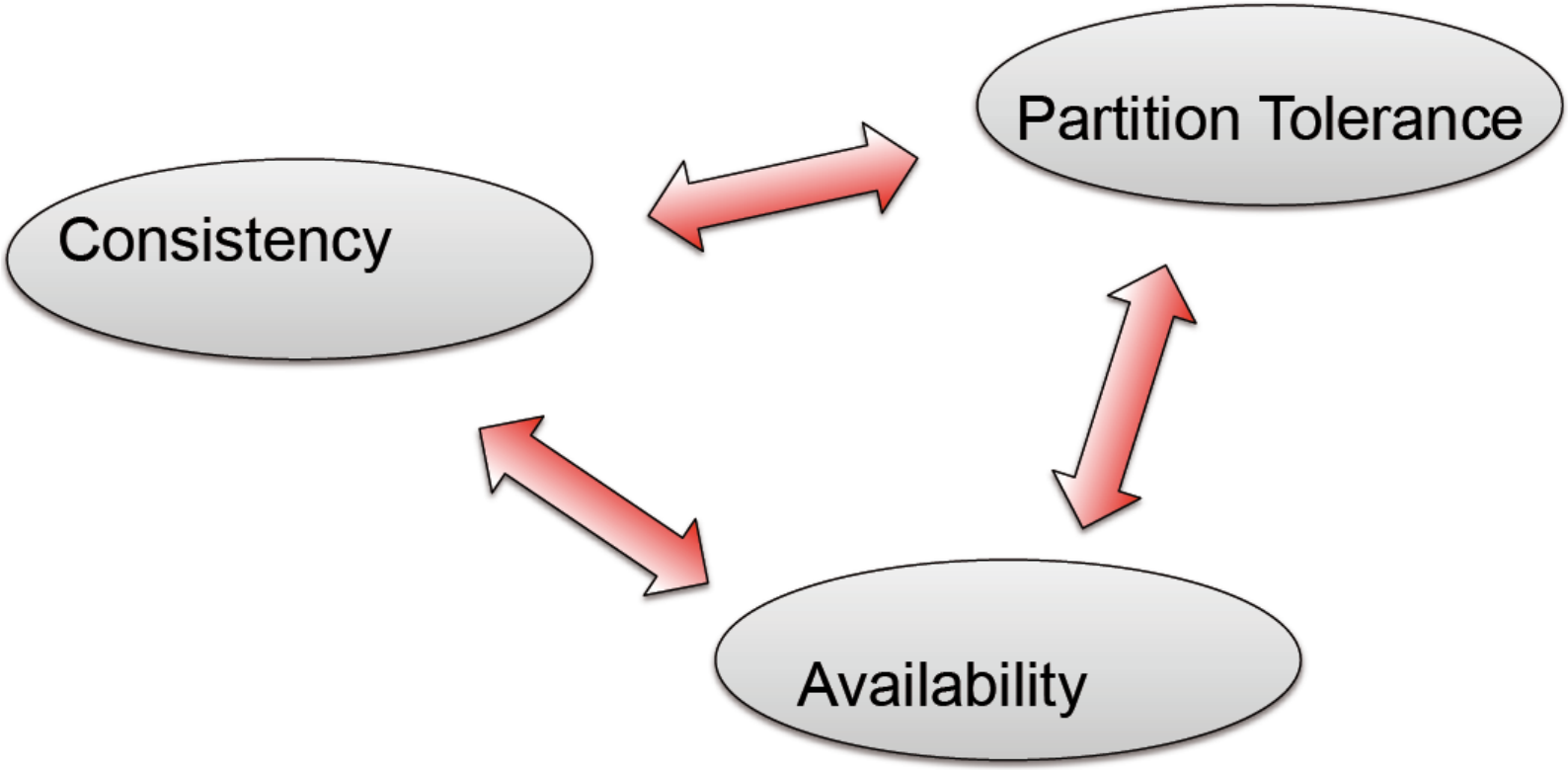
NoSQL Datenbanken

- Nicht-relationaler Ansatz
- Kein vordefiniertes Datenmodell/Tabellenschema
- Horizontale Skalierung
- Umgang mit großen Datenmengen

- ACID Eigenschaften nicht im Vordergrund
  - Wichtig ist Skalierbarkeit
  - „Eventually Consistent“

- Produkt-spezifisches API mit Fokus auf konkrete Anwendungsfälle
- Anwendungsfall beeinflusst Datenstruktur

CAP Theorem



- Eric Brewer
- 2 Aspekte können gleichzeitig garantiert werden

NoSQL Datenbanken

- Key-value Datenbank
- Spaltenorientierte Datenbank
- Dokumentenorientierte Datenbanken
- Graphdatenbanken

Key-Value Datenbanken

- Einfaches Datenmodell
  - Key referenziert Value
- Speichert Daten wie in einer Map / Hashtable

- Leistungsfähig in unterschiedlichen Szenarien
- Limitierungen bei Aggregationen und komplexen Abfragen

- Implementierungen
  - Memcached, Memcachedb, Membase
  - Voldemort
  - Redis
  - Riak

Spaltenorientierte Datenbank

- Speichert Inhalte spaltenweise ab
  - Statt zeilenweise
- Spalten können einfach hinzugefügt werden
  - Unabhängig von der Zeile
  - Minimiuerung von Null-Zellen

- Implementierungen
  - Hbase
  - Cassandra
  - Hypertable

Dokumentenorientierte Datenbank

- Speichert Dokumente
  - Map / Hashtable ähnlich
  - unique ID field
- Dokumente
  - Strukturierte Dateien in unterschiedlichen Datenformaten
  - Key-value Paare
  - JSON
  - XML

- Verschachtelte Strukturen
  - Hohe Flexibilität

- Implementierungen
  - MongoDB
  - CouchDB

Graphdatenbanken

- Datenbank, die Graphen benutzt, um Informationen darzustellen und Daten abzuspeichern
- Knoten, Kanten / Beziehungen
  - Key-value Paare speichern Informationen
- Navigation durch den Graphen
  - Auf Grund der Kanten / Beziehungen
- Implementierungen
  - Neo4J
  - Polyglot

Taxonomie

- Key-value
  - Riak
  - Membase
  - Cassandra
  - Berkeley DB
  - Redis
- Column-oriented
  - Hbase
  - Hypertable
- Document Store
  - MongoDB
  - CouchDB
  - Amazon SimpleDB
- Graph Databases
  - Neo4J

REST

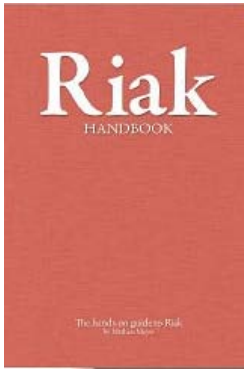
- Representational State Transfer
  - Ressourcen, keine Services

- Verwendet http Operationen (get, post, put, delete)
- Alle Datentypen können übertragen werden (html, jpg, gif, XML…)

- Ressourcen werden über eine URI referenziert
  - http GET mydomain.com/user/32213
  - http POST mydomain.com/article/a6557448x
  - http PUT mydomain.com/order
  - http DELETE mydomain.com/article/b443235

- Client/Server
- Stateless
- Cacheable, Scalable

Riak



- Key-Value Datenbank
- Werte
  - text, JSON, XML, Bilder, Videos…
  - HTTP-REST Schnittstelle

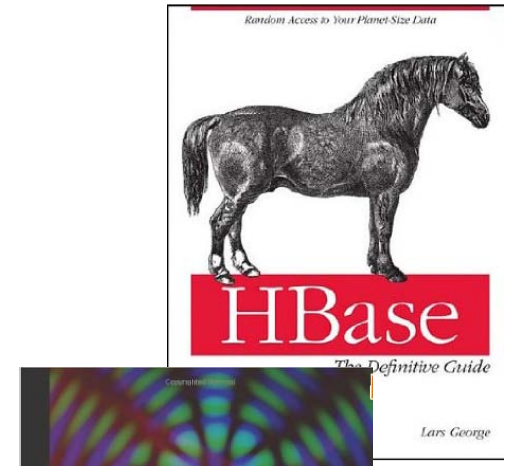
- Daten ablegen

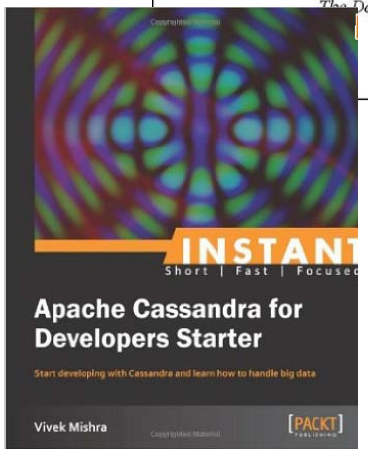
```
$ curl -v -X PUT http://localhost:8091/riak/favs/db \
  -H "Content-Type: text/html" \
  -d "<html><body><h1>My new favorite DB is RIAK</h1></body></html>"
```

- Bucket
  - Erlaubt eine Klassifizierung von Keys
  - http://SERVER:PORT/riak/BUCKET/KEY
- Links
  - Links are metadata that associate one key to other keys
  - Link: </riak/bucket/key>; riaktag=\“linktag\"

Apache Cassandra

- Spaltenorientierte Datenbank
- Auf große, verteilte Systeme ausgelegt
  - hohe Skalierbarkeit
  - Hohe Ausfallsicherheit
- Daten werden in Schlüssel-Wert-Relationen gespeichert
  - Eventually-consistent
- Datenmodell
  - Columns
  - Column Family
  - Super Columns
- Strukturierung erlaubt Leistungsverbesserung





Apache Cassandra

- Column (Triplet)
  - Name
    - raw byte array
  - Value
    - raw byte array
  - Timestamp

{
name:“email“
Value: „xy@z.com“
Timestamp:123456
}

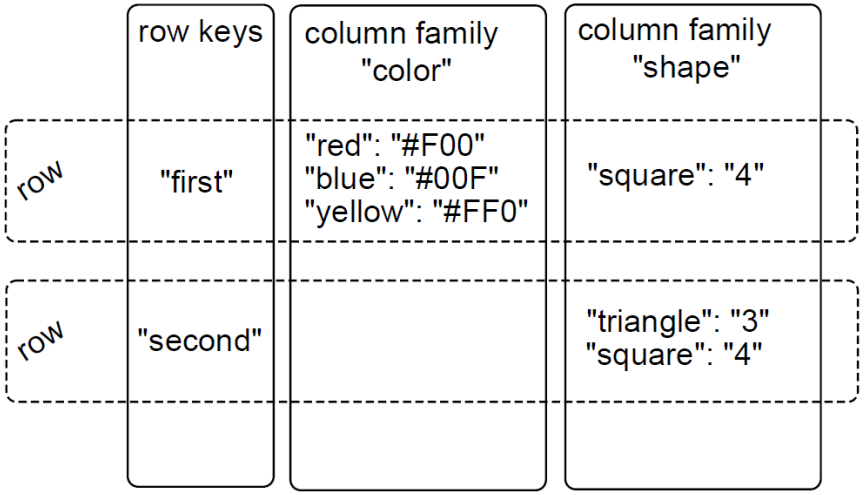
Apache Cassandra

- ColumnFamily
  - Name
    - raw byte array
  - Value
    - Menge an Columns

```
{
name: “Kontaktdaten”,
value:
{
handy: {name: “handy”, value: “0661123456789 ”, timestamp: 123456789},
festnetz: {name: “festnetz”, value: “01123456789”, timestamp: 123456789},
}
}
```

Apache Cassandra

- ColumnFamily
  - Name
    - raw byte array
  - Value
    - Menge an Columns




```
{
name: “Kontaktdaten”,
value:
{
handy: {name: “handy”, value: “0661123456789 ”, timestamp: 123456789},
festnetz: {name: “festnetz”, value: “01123456789”, timestamp: 123456789},
}
}
```

Apache Cassandra

- Column Family
  - Container für Columns
  - Statisch
  - Dynamisch
- Sortierung
  - BytesType
  - UTF8Type
  - AsciiType
  - LongType
  - TimeUUIDType
  - …
- Keyspaces
  - Definiert Replikationsstrategie
  - Gruppiert Column Families

```
<Keyspace Name=„IFS">
  <ColumnFamily Name=„CDL" CompareWith="UTF8Type" />
  <ReplicaPlacementStrategy>org.apache.cassandra.locator.RackUnawareStrategy</ReplicaPlacementStrategy>
  <ReplicationFactor>1</ReplicationFactor
</Keyspace>
```

set <ksp>.<cf>['<key>']['<col>'] = '<value>'

set IFS.CDL[‘LVA'][‘NAME'] = ‘SEPM‘
get IFS.CDL[‘LVA']

Apache Cassandra

- Tabelle erstellen
  - *Create „wiki“, „text“*
- Daten hinzufügen
  - *Put „wiki“, „Home“, „text:“ , „Welcome“*
- Abfragen
  - Tabellenbezeichnung, Zeilen-Schlüssel
  - *Get “wiki”, “Home”, “text:”*
  - ➔ *timestamp=12348263474, value=“Welcome”*
- Tabellenstruktur ändern ist aufwendig und erfordert
  - Neue ColumnFamily erstellen
  - Daten kopieren

Apache CouchDB

- dokumentenorientierte Datenbank
- Daten werden in Dokumenten gespeichert
  - JSON Objekte
- Eventual Consistency
- Multi-Version Concurrency Control (MVCC)
  - Kein Lock der Daten
- RESTful HTTP API
  - GET
  - PUT/POST
  - DELETE

Apache CouchDB

- Dokumenten
  - JSON
  - Zwei weitere Felder
    - id - Dokument
    - Rev - revision information

- Datenbank anlegen

  curl –X PUT http://serverip:port/database

  curl –X DELETE http://serverip:port/database

- Dokument suchen

  curl –X GET http://serverip:port/database/doc_id

  curl –X GET http://serverip:port/database/doc_id?rev=946B7D1C

HTTP/1.1 200 OK
Etag: "946B7D1C“
Date: Tue, 29 May 2012 05:19:42 +0000GMT Content-Type: application/json Content-Length: 256 Connection: close
{ "_id":“doc_id", "_rev":"946B7D1C", "Subject":“lecture", „name":“SEPM“}

Apache CouchDB

- Neues Dokument erstellen

Curl –X PUT „Content-Type: application/json“ – d `{ "Subject":“lecture", „name":“SEPM“}` http://serverip:port/database/doc_id

HTTP/1.1 200 OK
Etag: "946B7D1C“
Date: Tue, 29 May 2012 05:19:42 +0000GMT Content-Type: application/json Content-Length: 256 Connection: close
{"ok": true, „id":“doc_id", "rev":"946B7D1C“}

- Bestehendes Dokument aktualisieren

Curl –X PUT „Content-Type: application/json“ – d `{"_id":“doc_id", "_rev":"946B7D1C", "Subject":“lecture", „name":“SEPM2“}`
http://serverip:port/database/doc_id

HTTP/1.1 200 OK
Etag: "946B7D1C“
Date: Tue, 29 May 2012 05:19:42 +0000GMT Content-Type: application/json Content-Length: 256 Connection: close
{"ok": true, „id":“doc_id", "rev":„ 366846F01C“}

Apache CouchDB

- Dokument löschen

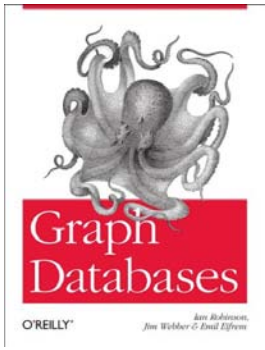
Curl –X DELETE http://serverip:port/database/doc_id?rev=366846F01C

HTTP/1.1 200 OK
Etag: "946B7D1C“
Date: Tue, 29 May 2012 05:19:42 +0000GMT Content-Type: application/json Content-Length: 256 Connection: close
{"ok": true, "rev":“366846F01C“}

Neo4J

- Graphdatenbanken benutzen Graphen um Daten zu speichern
  - Knoten
  - Kanten / Beziehungen
  - Properties

- Fokus ist auf den Beziehungen zwischen den Daten
  - Einfache Schemaerweiterung
  - Keine Join-Operationen



Neo4J

- Graph erstellen

```
firstNode = graphDb.createNode();
firstNode.setProperty( "message", "Hello, " );
secondNode = graphDb.createNode();
secondNode.setProperty( "message", "World!" );

relationship = firstNode.createRelationshipTo( secondNode, RelTypes.KNOWS );
relationship.setProperty( "message", „SEPM" );
```

Neo4J

- Knoten löschen

  firstNode.getSingleRelationship(RelTypes.KNOWS, Direction.OUTGOING).delete();
  firstNode.delete();
  secondNode.delete();

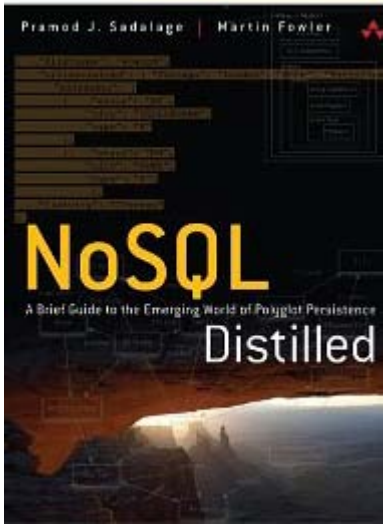
- Knoten, die noch in Beziehung stehen, können nicht gelöscht werden
  - Kanten haben immer einen Start- und Endknoten

- Abfragen

  ExecutionEngine engine = new ExecutionEngine(db);
  ExecutionResult result = engine.execute("start n=node(*) where n.name! = 'my node' return n, n.name");

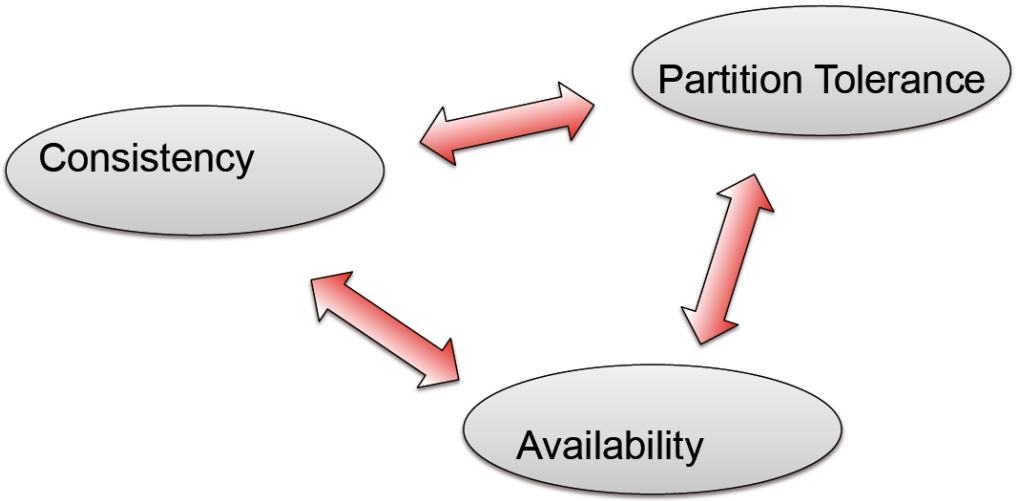
Polyglot Persistence

“Polyglot Persistence using multiple data storage technologies, chosen based on the way data is being used by individual applications. Why store binary images in relational database, when there are better storage systems.” (Martin Fowler)



Fokus der Technologien

- Availability - Consistency
  - RDBMS
- Availability – Partiton Tolerance
  - Apache Cassandra
  - CouchDB
  - Riak
- Consistency – Partition Tolerance
  - Hbase
  - MongoDB
  - Redis



Vorteile

- Große Anzahl an verfügbaren Systemen mit unterschiedlichen Eigenschaften
- Lösung eines speziellen Problems der Persistierung
  - Optimal
- Flexibler Umgang mit Schemata
- Skalierbarkeit und Umgang mit Partitionierung ist wichtiger als ACID Eigenschaft

Nachteile

- Große Anzahl an verfügbaren Systemen mit unterschiedlichen Eigenschaften
- Lösung eines speziellen Problems der Persistierung
  - Optimal
- Flexibler Umgang mit Schemata
- Skalierbarkeit und Umgang mit Partitionierung ist wichtiger als ACID Eigenschaft

Zusammenfassung

- Anforderungen sammeln
- Anwendungsfall verstehen
- Datenbanktechnologie(n) ableiten
  - RDBMS: Erfahrung, langjähriger Einsatz, ACID
  - NoSQL: Skalierbarkeit

- CDL Themen
  - http://qse.ifs.tuwien.ac.at/topics.htm
  - http:// cdl.ifs.tuwien.ac.at
  - http:// cdl.ifs.tuwien.ac.at/jobs
  - richard.mordinyi@tuwien.ac.at

References

- Eric Redmond and Jim R. Wilson, Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement, Pragmatic Bookshelf, May 18, 2012
- Mathias Meyer, Riak Handbook
- Pramod J. Sadalage and Martin Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley Professional; 1 edition, Aug 18, 2012
- Joshua Drake D. and John C. Worsley, Practical PostgreSQL, O'Reilly Media, Jan 14, 2002
- Clare Churcher, Beginning Database Design: From Novice to Professional, Apress; 1 edition, January 24, 2007
- Michael J. Hernandez, Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design, Addison-Wesley Professional; 3 edition, February 24, 2013
- Lars George, HBase: The Definitive Guide, O'Reilly Media; 1 edition, September 20, 2011
- Ian Robinson, Jim Webber, Emil Eifrem, Graph Databases, O'Reilly Media; Auflage 1, (12. Juni, 2013

References

- JPA - http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
- NoSQL Datenbanken
  - Dynamo: Amazon (http://dx.doi.org/10.1145/1323293.1294281)
  - Cassandra - http://www.datastax.com/docs/1.1/index
  - CAP Theorem http://www.julianbrowne.com/article/viewer/brewers-cap-theorem
  - ChouchDB http://guide.couchdb.org/
- Db4o - http://www.db4o.com/
  - http://community.versant.com/Documentation/Reference/db4o-8.0/java/tutorial/
  - http://community.versant.com/Documentation/Reference/db4o-8.0/java/tutorial/docs/ObjectManagerOverview.html#ObjectManagerOverview
- Cache - http://www.intersystems.com/
- XML Datenbanken
  - http://www.w3.org/TR/xquery-30/
  - http://www.rpbourret.com/xml/XMLDatabaseProds.htm
- NoSQL Datenbanken
  - http://nosql-database.org/
- http://blog.nahurst.com/visual-guide-to-nosql-systems

Stellenausschreibung
Software Entwickler (m/w)

- Wo: TU Wien, Institut für Softwaretechnik, CDL-Flex
- Was: Arbeitsabläufe in komplexen Software-Landschaften unterstützen
  - Mitarbeit an der Weiterentwicklung bestehender Lösungen für Industriepartner
  - Konzeption und Umsetzung projektspezifischer Anforderungen
  - Erarbeitung von Strategien zur Einbindung von On-Site Engineering Aktivitäten in projektspezifische Arbeitsabläufe
- Qualifikation
  - Sehr gute Kenntnisse in objektorientierter Software-Entwicklung, vorzugsweise: Java
  - Erfahrung mit OSGI, maven, Git und agiler Software-Entwicklung und Open-Source Software
  - Idealerweise Erfahrung mit zumindest einer der folgenden Technologien: Apache Felix, Apache Karaf, Apache Aries, Apache Wicket, Pax-Wicket –Projekten
  - Teamgeist, Qualitätsorientierung und hohes Maß an Problemlösungskompetenz
- Kontakt: Dr. Richard Mordinyi (richard.mordinyi@tuwien.ac.at)