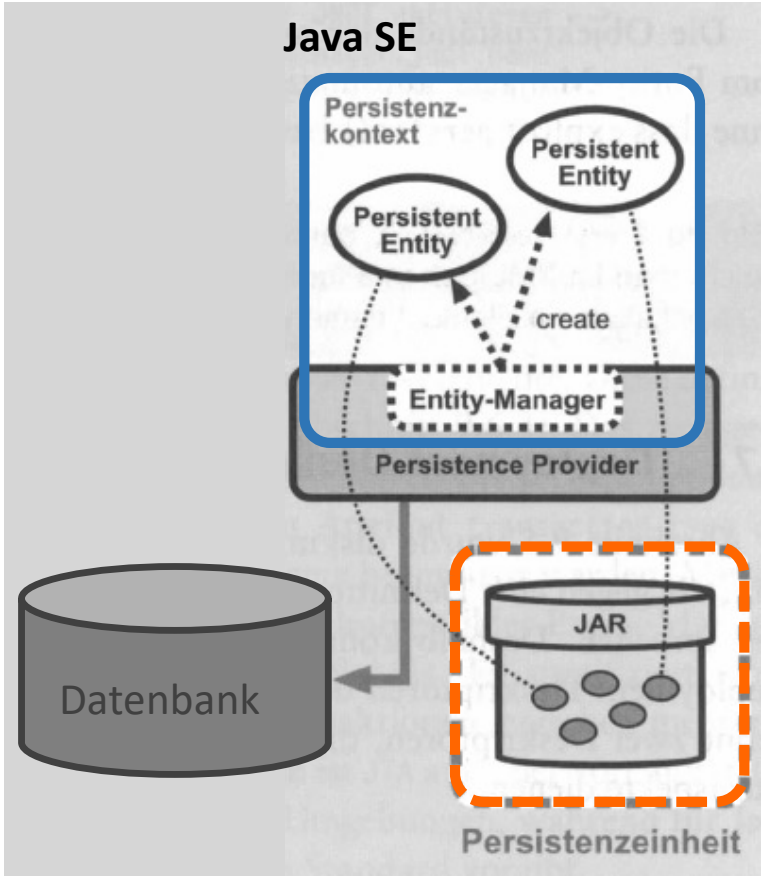
Softwaretechnik 2

Persistente Datenhaltung II

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Struktur



Persistenzkontext

Persistenzeinheit

Java Persistence API (JPA)

- ist eine im JSR 220 standardisierter objektrelationaler Mapper (OR-Mapper) für Java-basierte Anwendungen
- speichert Laufzeit-Objekte einer Java-Anwendung über eine einzelne Sitzung hinaus in einer relationalen Datenbank
- die API wurde im Mai 2006 erstmals veröffentlicht
- innerhalb des *javax.persistence* Pakets definiert
- TopLink Essentials war die Referenzimplementierung für JPA 1.0
- EclipseLink ist die Referenzimplementierung für die JPA 2.0 / 2.1
- weitere JPA-Implementierungen: Hibernate, Apache OpenJPA
- seit 2019: Weiterentwicklung unter dem Namen Jakarta Persistence

JPA

Persistent Entity

- Persistent Entities sind POJOs (Plain Old Java Objects)
  - implementieren kein spezielles Interface → normale Java-Klasse mit Annotationen

```
import javax.persistence.*;

@Entity
public class Kunde{

   @Id
   private int id;
   private String name;

   public int getId() { return id; }
   public void setId(int id) { this.id = id; }

   public String getName() { return name; }
   public void setName(String name) { this.name = name; }
}
```

Zukünftiger Paketname `jakarta.persistence`

Alternativ `@Id` vor getter-Methode angeben!

JPA

Persistent Entity

- Im einfachsten Fall (mit Annotationen) benötigt ein Persistent Entity lediglich
  - die Annotation `@Entity`,
  - ein Feld bzw. Attribut, das durch die Annotation `@Id` als Primärschlüssel ausgewiesen wird und
  - einen Default-Konstruktor, der zwingend erforderlich ist.

⇒ Paradigma: *configuration by exception*

- Persistent Entity Klassen können überall verwendet werden und ihre Instanzen sogar zur Anzeige an Clients verschickt werden, falls sie serialisierbar sind.

- Neue Instanzen werden einfach mit new()-Operator erzeugt:

```
Kunde kunde = new Kunde();
kunde.setName("Mustermann");
```

JPA

Persistent Entity

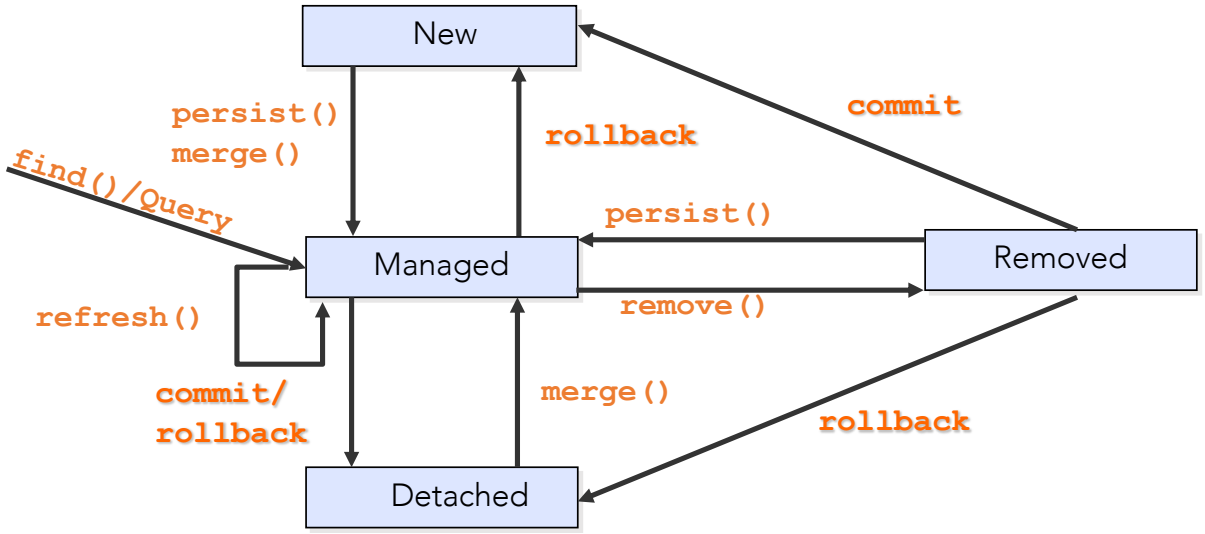
- Neue Instanzen bleiben solange "einfache Objekte" bis der EntityManager das Objekt in der relationalen Datenbank persistiert.
  - pro Entity wird einer Datenbanktabelle angelegt
  - jede Zeile repräsentiert eine konkrete Instanz einer Entity
- Persistent Entities sind entweder managed oder sie sind unmanged.
  - wenn sie managed sind, dann sind sie einem EntityManager zugeordnet (attached) und dieser synchronisiert Zustandsänderungen mit der Datenbank

    oder
  - sie sind unmanaged, dann sind sie NICHT einem EntityManger zugeordnet (detached) und die Änderungen werden NICHT mit der Datenbank synchronisiert.
- Bei fehlenden Angaben trifft der EntityManager Standardannahmen:
  - Tabellenname = Klassenname der Persistent Entities

    (→ kann mit **@Table(name="xyz")** geändert werden)
  - Spaltenname = Attributname (→ kann mit **@Column(name="xyz")** geändert werden)

JPA

Zustandsmodell



Callback-Methoden:

```
@PrePersist
@PostPersist
@PreRemove
@PostRemove
@PreUpdate
@PostUpdate
@PostLoad
```

JPA

Persistent Entity Annotationen

`@Entity(name="JP-QL-name")`

- Deklariert Java-Klasse als Entity Klasse, d.h. Objekte dieser Klasse werden in der Datebank gespeichert
- Optional kann ein Name festgelegt werden, unter dem die Klasse in JP-QL angesprochen werden kann. Defaultmäßig wird der Klassenbezeichner gewählt.

`@Id`

- Identifiziert den eindeutigen Schlüsselwert (Primärschlüssel)
- Entweder an Attribut oder Methode verwenden

`@GeneratedValue(value)`

- Automatische Generierung einer ID
- Muss zusammen mit `@ID` verwendet werden

`@Temporal(value)`

- Mapping von `java.util.Date` oder `java.util.Calendar` auf Datenbanktyp: `Date`, `Time` oder `Timestamp`

JPA

Persistent Entity Annotationen

`@Enumerated(value)`

- Enumerations können persistent sein
  In der Datenbank wird entweder der Ordinalwert (Position beginnend bei 0) oder der Stringwert (Name der Konstante) abgelegt

- Default für die Datenbankablage ist ORDINAL. Der Datentyp in der Datenbank kann varchar oder int sein. varchar ist sowohl für den Stringwert wie für die Zahl geeignet

```
public enum Status { NEU, NORMAL, VIP }

@Entity
public class Kunde{
   @Enumerated(EnumType.ORDINAL)
   public Status getStatus() {return status};
   // oder
   @Enumerated(EnumType.STRING)
  public Status getStatus() {return status};
 }
```

JPA

Entity Manager

- Der Entity Manager realisiert die Schnittstelle zur Datenbank
  - OR-Mapping (Objekte finden, persistieren, entfernen …)
  - Verwaltung der Entities

```
import javax.persistence.*;
import entities.Kunde;

public class Main{
  EntityManagerFactory emf;

        public static void main( String[] args )
        {
          (new Main()).test();
        }

        void test(){…}

  public <T> void createEntity( T entity ) {…}
  public <T> T readEntity( Class<T> clss, Object id ) {…}
}
```

JPA

Entity Manager

- Kunde mit Hilfe des Entity Managers persistieren (1/2):

```
void test(){
   emf = Persistence.createEntityManagerFactory("MeineJpaPU");

   try {

      Kunde kunde = new Kunde();
      kunde.setName( ”Max Mustermann" );

      createEntity( kunde ); // ausgelagerte Methode
      Object id = kunde.getId();

      System.out.println( "\n--- "      + readEntity( Kunde.class, id )
                          + " ---\n" );
   } finally {
      emf.close();
   }
}
```

JPA

Entity Manager

- Kunde mit Hilfe des Entity Managers persistieren (2/2):

```
public <T> void createEntity(T entity){

      EntityManager     em = emf.createEntityManager();
      EntityTransaction tx = em.getTransaction();

      try {
         tx.begin();
         em.persist( entity );
         tx.commit();
      } catch( RuntimeException ex ) {
         if( tx != null && tx.isActive() ) tx.rollback();
         throw ex;
      } finally {
         em.close();
      }
}
```

JPA

Entity Manager

- Entity auf Basis der **id** beim Entity Manager anfragen:

```
public <T> T readEntity(Class<T> clss, Object id){

      EntityManager em = emf.createEntityManager();

      try {
         return em.find(class, id);
      } finally {
         em.close();
      }
}
```

Java Persistence Query Language (JPQL)

- Objektorientierte Abfragesprache als Teil der JPA
- Alternative zum Entity Manager für komplexere Abfragen
- Syntax ist an SQL angelehnt:

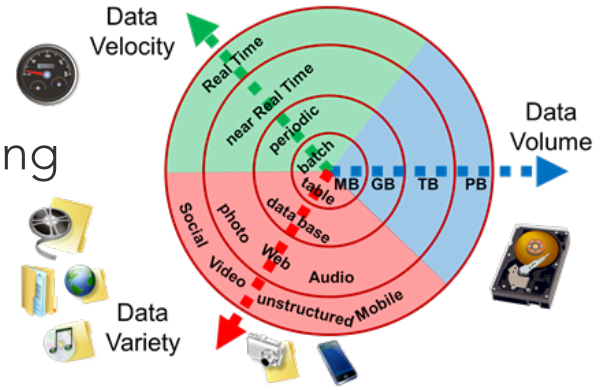
```
public <T> T readEntity(Class<T> clss, Object id){

      EntityManager em = emf.createEntityManager();

      try {
       Query query = em.createQuery("SELECT k.name FROM Kunde k");
       List<String> result = query.getResultList();
      } finally {
         em.close();
      }
}
```

Struktur

NoSQL

Not only SQL (NoSQL)

- Bezeichnung für Datenbanken, die keinen relationalen Ansatz verfolgen und auf festgelegte Tabellenschemata verzichten → Alternative zur SQL-Welt
- NoSQL-Paradigma geht auf ein von Oskarsson und Evans angekündigtes Treffen im Jahr 2009 unter dem Motto „*NoSQL: open source, distributed, and non-relational databases*“ zurück
- Grund: Unzufriedenheit über die Limitierungen von traditionellen relationalen Datenbanken im Hinblick auf die **effiziente** Verarbeitung von **großen** und **unstrukturierten** Datensätzen
- Big Data lassen relationale Datenbanken an ihre Grenzen stoßen
  - Datenvolumen, -vielfalt und –geschwindigkeit nehmen immer mehr zu
  - IoT, Web 2.0, Industrie 4.0, Mobile Computing …



https://gi.de/informatiklexikon/big-data

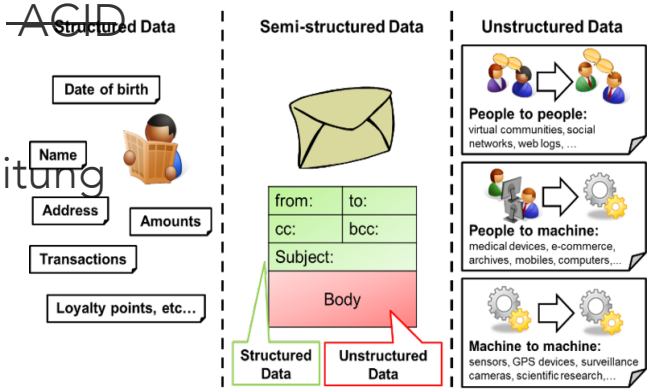
Fachhochschule Dortmund
University of Applied Sciences and Arts

NoSQL

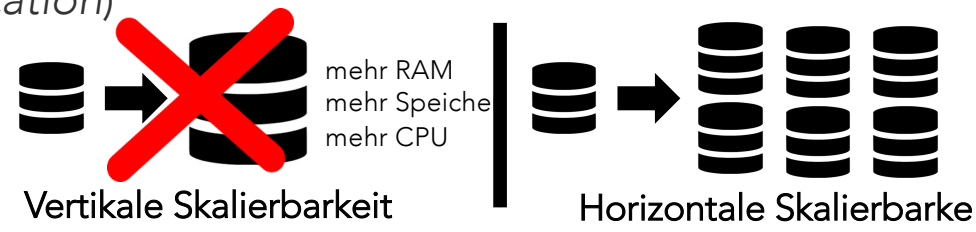
Eigenschaften

we focus on students

- BASE-Modell (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) → ~~ACID~~
  - weitestgehender Verzicht auf Transaktionen
- NoSQL Datenbanken zielen auf die performante und flexible Verarbeitung
  - strukturierter (z.B. Datenbanken, Formular, Kundenstamm),
  - semi-strukturierter (z.B. Email, HTML, JSON, XML) und
  - unstrukturierter (Bilder, Videos, Text) Datensätze ab
- Horizontale Skalierbarkeit und hohe Ausfallsicherheit auf Basis verteilter Datenbanken gegeben (*Sharding and Replication*)
- Zunehmende Nutzung von In-Memory-Datenbanken
- Viele Open Source Varianten verfügbar



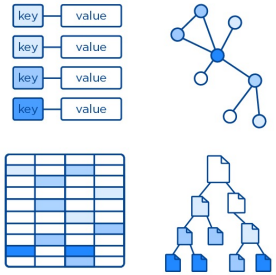
https://gi.de/informatiklexikon/big-data



NoSQL

Datenorganisation

- Verzicht auf Schemarestriktionen
- Stattdessen kommen unterschiedliche Datenmodelle zum Einsatz
  - Wertepaare, Dokumente, Graphen, Zeitreihen, etc.
  - Jede Art ist für einen gewissen Zweck optimiert und hat Vor- und Nachteile
- Objektbasierte APIs für viele Programmiersprachen (z.B. Java, Python, C, etc.)
- Abwägen welche NoSQL-Datenbank für einen Anwendungsfall am besten geeignet ist
  - Keine Datenmodell liefert eine „one fits all“ Lösung
  - Welche Daten liegen vor und wie werden diese in der Anwendung genutzt?
  - Generell eignen sich NoSQL-Datenbanken für verteilte und hochskalierbare Anwendungen mit hohem Durchsatz und geringer Latenz



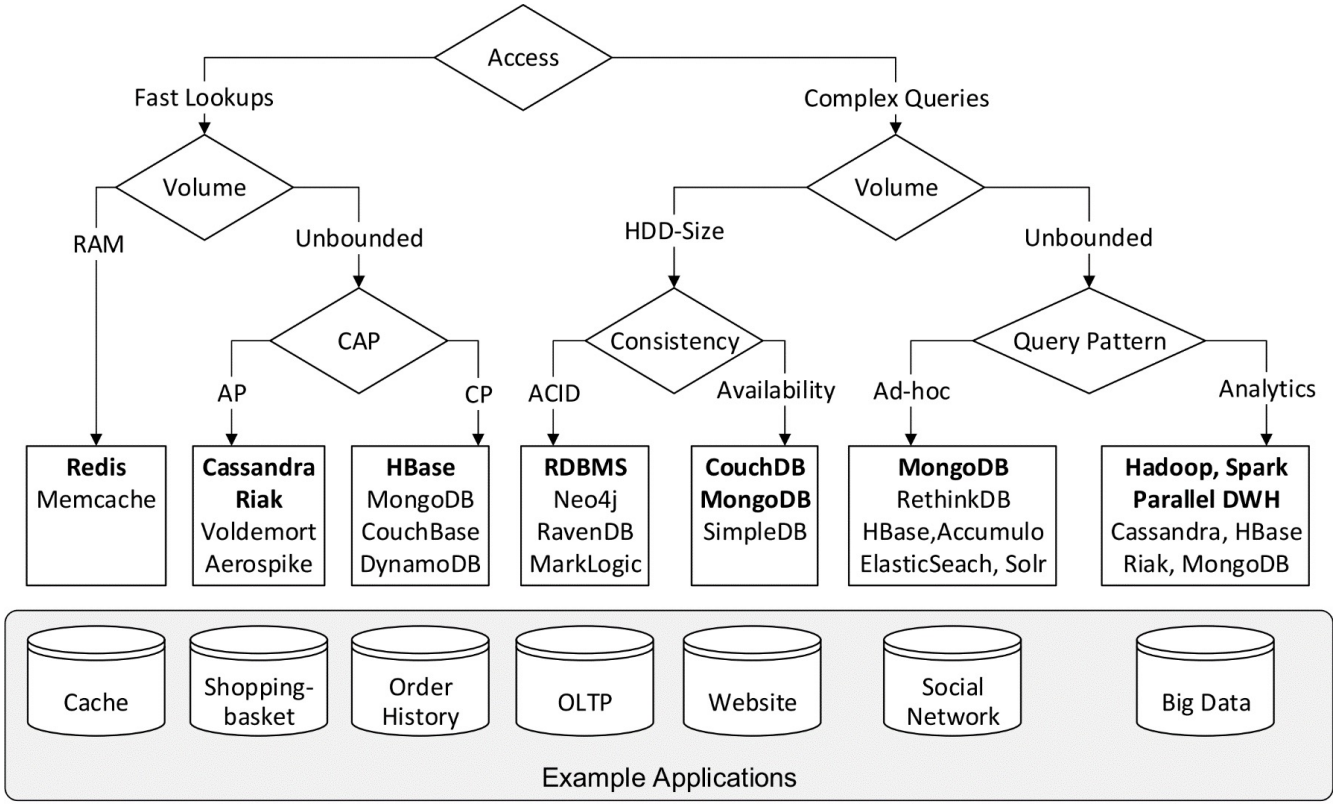
NoSQL

Relationale vs. NoSQL Datenbanken

	Relationale Datenbank	NoSQL Datenbank
Lizenzmodell	Meistens Kommerziell	Meistens Open Source
Datenmodell	Vordefiniertes Tabellenschema mit Zeilen und Spalten	Schemafrei, viele Varianten verfügbar (Key-Value, Dokument, Graph, etc.)
Flexibilität	Klare und strikte Datenstrukturen	Flexibel erweiterbar um neue Datentypen
Skalierbarkeit	Vertikal Skalierbar	Horizontal Skalierbar
Performanz	Abhängig von der Hardware (Single Host), Tabellenstruktur und Effizienz der Algorithmen	Abhängig von der Größe des Hardwareclusters, Netzwerkbandbreiten und dem verwendeten Datenmodell
APIs	SQL	Objektbasierte APIs
ACID	ACID Eigenschaften werden komplett umgesetzt	Nur teilweise umgesetzt (BASE Modell)

NoSQL

Auswahlprozess

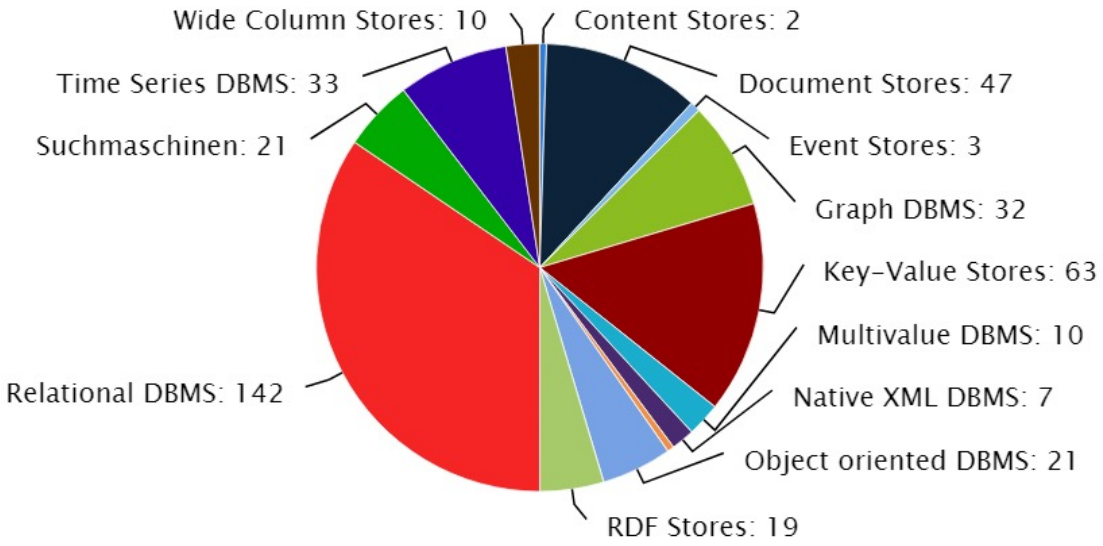


Felix Gessert, Wolfram Wingerath, Steffen Friedrich, Norbert Ritter: NoSQL database systems: a survey and decision guidance. Comput. Sci. Res. Dev. 32(3-4): 353-365 (2017)

NoSQL

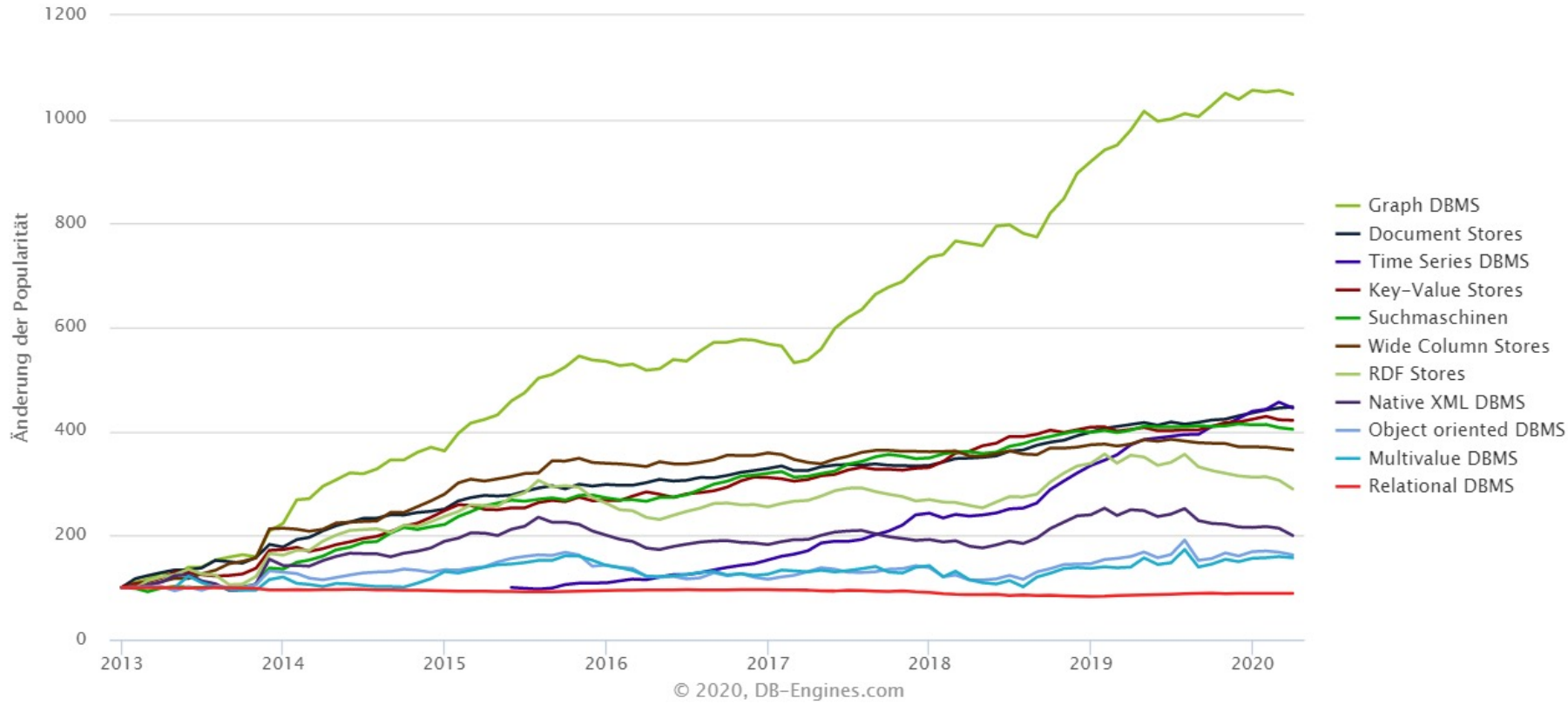
Heterogene Landschaft

- Übersicht über NoSQL Datenbanken: https://hostingdata.co.uk/nosql-database/
- Heterogene Menge mit unterschiedlichen Architekturen und Datenmodellen:



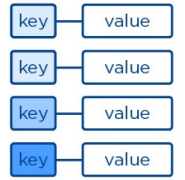
NoSQL

Trends



NoSQL

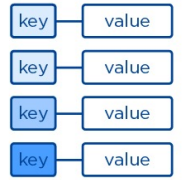
Key-Value Datenbank



- Einfachste Variante einer NoSQL-Datenbank und vergleichbar mit einem Wörterbuch
- Werte (Value) werden über einen eindeutigen Schlüssel (Key) identifiziert
  - Beliebiges Datenformat für einen Schlüssel und Wert möglich, z.B. String, Integer, Listen, Bilder, Code, etc.
- Speicherung via In-Memory-Datenbank oder On-Disk-Lösung
- Vorteile
  - schnelle Schreib- und Lesegeschwindigkeit bei einfachen Datensätzen
  - hohe Skalierbarkeit
  - größtmögliche Freiheit bei der Datenstruktur
- Nachteile
  - sehr eingeschränkte Datenbankoperationen (*GET*, *PUT*, *DELETE*) und Abfragemöglichkeiten
- Anwendungsbeispiele: UNIX, Warenkorb in einem Online-Shop, Userprofile, IP Tabellen, Sessions, Caching

NoSQL

Key-Value Datenbank



- Beispiel Onlineshop:

Schlüssel	Wert
kunde:00001:name	Jens Mustermann
kunde:00001:artikel	[12, 332, 16231]
kunde:00045:artikel	{50, 441, 0222}
artikel:00001:farben	{rot, gelb, grün}

Welche Datentypen für einen Wert unterstützt werden, hängt von der jeweiligen Datenbank ab

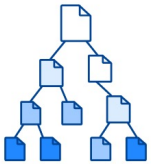
- Beispiel Musikstreaming:

Schlüssel	Wert
künstler:1:name	AC/DC
künstler:1:gerne	Hard Rock
künstler:2:name	Elvis
künstler:2:gerne	Rock 'n' Roll

Die Wahl des Schlüssels sollte gut überlegt sein und für den jeweiligen Anwendungsfall hin definiert sein

NoSQL

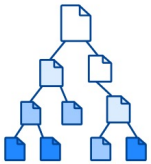
Dokumentenorientierte Datenbank



- Spezielle Variante des Key-Value-Datenmodells
- Speicherung von Daten in dokumentenähnlichen Strukturen
- Dokumente enthalten semi-strukturierte Daten und werden über einen eindeutigen Schlüssel identifiziert
  - Jedes Dokument kann einen eigenen Strukturaufbau verfolgen, z.B. XML-, YAML-, JSON- oder BSON
  - Metadaten werden durch Datenbank-Engine zur Optimierung extrahiert
- Vorteile
  - hohe Flexibilität bei sich häufig ändernden Daten
  - Dokumente repräsentieren Objekte
  - umfangreiche Query-Möglichkeiten
- Nachteile
  - Abfragemöglichkeiten bei komplexen Datenbeziehungen
- Anwendungsbeispiele: CMS, Webapplikationen im Allgemeinen

NoSQL

Dokumentenorientierte Datenbank



- Beispiel Onlineshop:

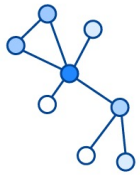
Schlüssel	Dokument
1156sd13fgdaal	{ "_ID": "1", "Name": "Jens Mustermann" }
43ajzddsfdsf34	{ "_ID": "2", "Name": "Klaus Dachpfanne", "Email": "Klaus@fh-dortmund.de" }
5gg990fnsfinas	<kunde> <id>5</> <name>Agathe Bauer</name> </kunde>

Fachhochschule Dortmund
University of Applied Sciences and Arts

NoSQL

Graphdatenbank

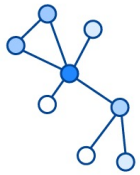
we focus on students



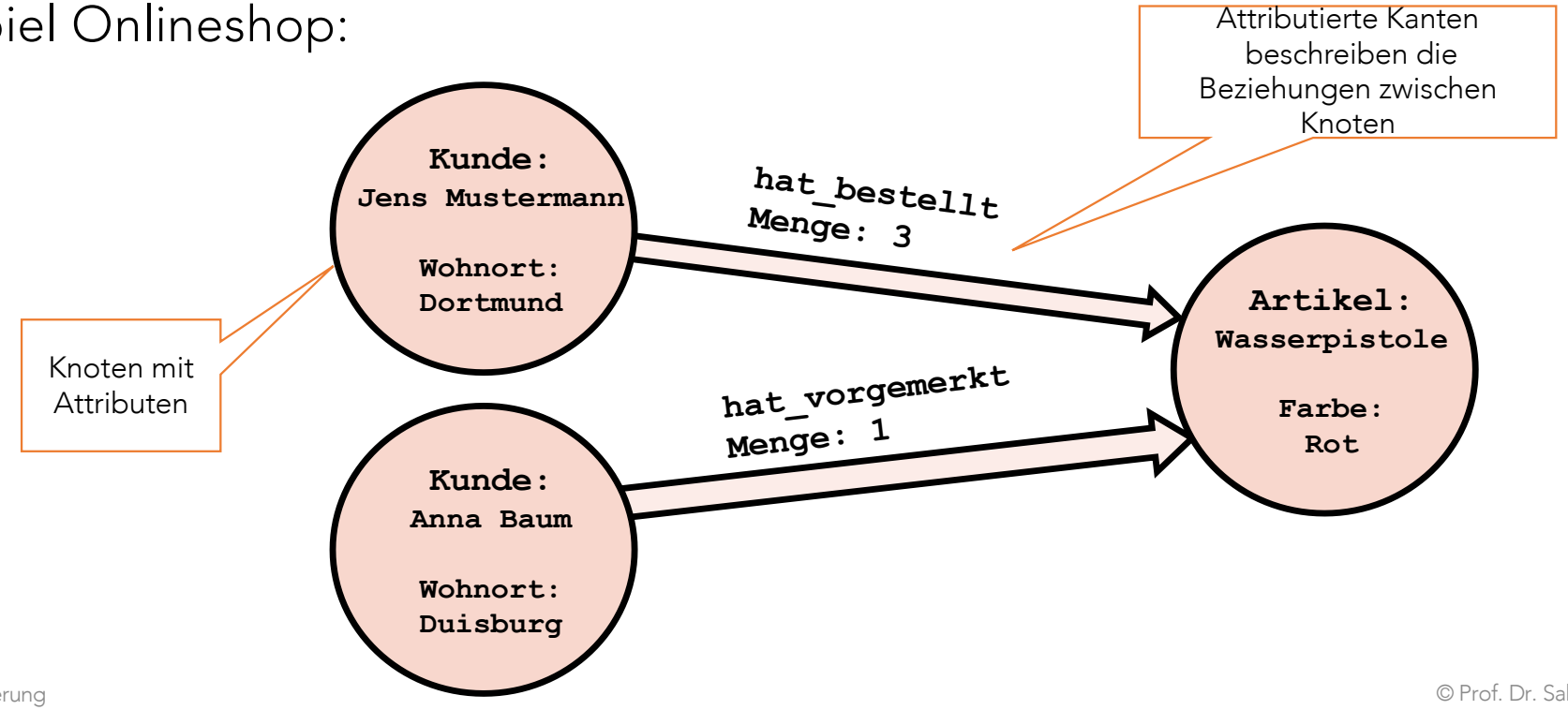
- Darstellung von Beziehungen ist eine große Schwachstelle bei relationalen Datenbanken
- Speicherung von Daten in Graph-Strukturen um stark vernetze Informationen abzubilden
  - Knoten repräsentieren Entitäten und sind über einen eindeutigen Bezeichner identifizierbar
  - Kanten stellen die Relation zwischen Knoten dar und können mittels Attributen gewichtet werden
- Vorteile
  - Graphenmodell ist optimiert für das Speichern und Auffinden von Beziehungen zwischen Daten
  - Nutzung von Graphalgorithmen für komplexe Abfragen
  - Abfragegeschwindigkeit unabhängig von der Gesamtmenge der Daten
- Nachteile
  - schlecht skalierbar
  - keine einheitliche Abfragesprache
- Anwendungsbeispiele: Soziale Netzwerke, Logistik, Empfehlungsanwendungen, Semantic Web

NoSQL

Graphdatenbank



- Beispiel Onlineshop:

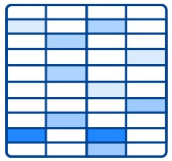


Fachhochschule Dortmund
University of Applied Sciences and Arts

NoSQL

Spaltenorientierte Datenbank (Wide Column Stores)

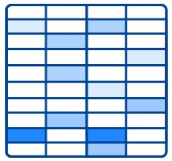
we focus on students



- Konzeptionell die größte Ähnlichkeit zu relationalen Datenbanken
- Zweidimensionale Key-Value Datenbank bei der Daten spaltenweise anstatt zeilenweise gespeichert werden
  - Datenbank ist in unterschiedliche Spaltenfamilien aufgeteilt, die wiederum eine Sammlung von Zeilen aggregieren
  - Jede Zeile verfügt über eine beliebige Anzahl an Spalten und wird über einen eindeutigen Zeilenschlüssel identifiziert
  - Jede Spalte setzt sich aus einen Key-Value Paar und Zeitstempel zusammen, wobei sich der Spaltenaufbau zeilenweise unterscheiden kann
- Vorteile
  - sehr performant bei Aggregationen (z.B. SUM, AVG, COUNT) auf großen Datenbeständen
  - geeignet für Analysen von umfangreichen und strukturierten Daten
  - auf verteilte Systeme ausgelegt durch gute Komprimierungs- und Partitionierungseigenschaften
- Nachteile
  - nicht optimal für transaktionale Anwendungen (INSERT, UPDATE)
- Anwendungsbeispiele: Data Mining, Data Warehouse/OLAP (Online Analytical Processing), Reporting

NoSQL

Spaltenorientierte Datenbank (Wide Column Stores)



- Beispiel Onlineshop:

Fachhochschule Dortmund
University of Applied Sciences and Arts

NoSQL

Zeitreihen Datenbank

we focus on students

- Datenmodell ist auf das Speichern und die Analyse von Zeitreihen ausgelegt
  - Fokus liegt auf dem Hinzufügen (INSERT) von Daten und nicht deren nachträgliche Modifikation (UPDATE)
  - Analysen von Veränderungen in Daten über eine gewisse Zeitspanne
- Jede Zeitreihe enthält einen Zeitstempel und eine Reihe von assoziierten Werten (Messreihendaten)
- Vorteile
  - Umfangreiche Aggregationen über viele Datensätze sind sehr effizient und Anfragen können in Echtzeit bedient werden
  - Komprimierung über Zeitintervalle möglich
  - Auffinden von Trends oder Anomalien in Datensätzen
- Nachteile
  - Nur für Anwendungsfälle mit Bezug zu Zeitreihen ausgelegt
- Anwendungsbeispiele: Industrie 4.0 (Maschinendaten), Connected Vehicles (Telemetriedaten), Smart Home, Wetterdaten, Börsenkurse

NoSQL

Zeitreihen Datenbank

- Beispiel aus dem Forschungsprojekt APPSTACLE (Connected Vehicles): Speicherung von Telemetriedaten in einer InfluxDB

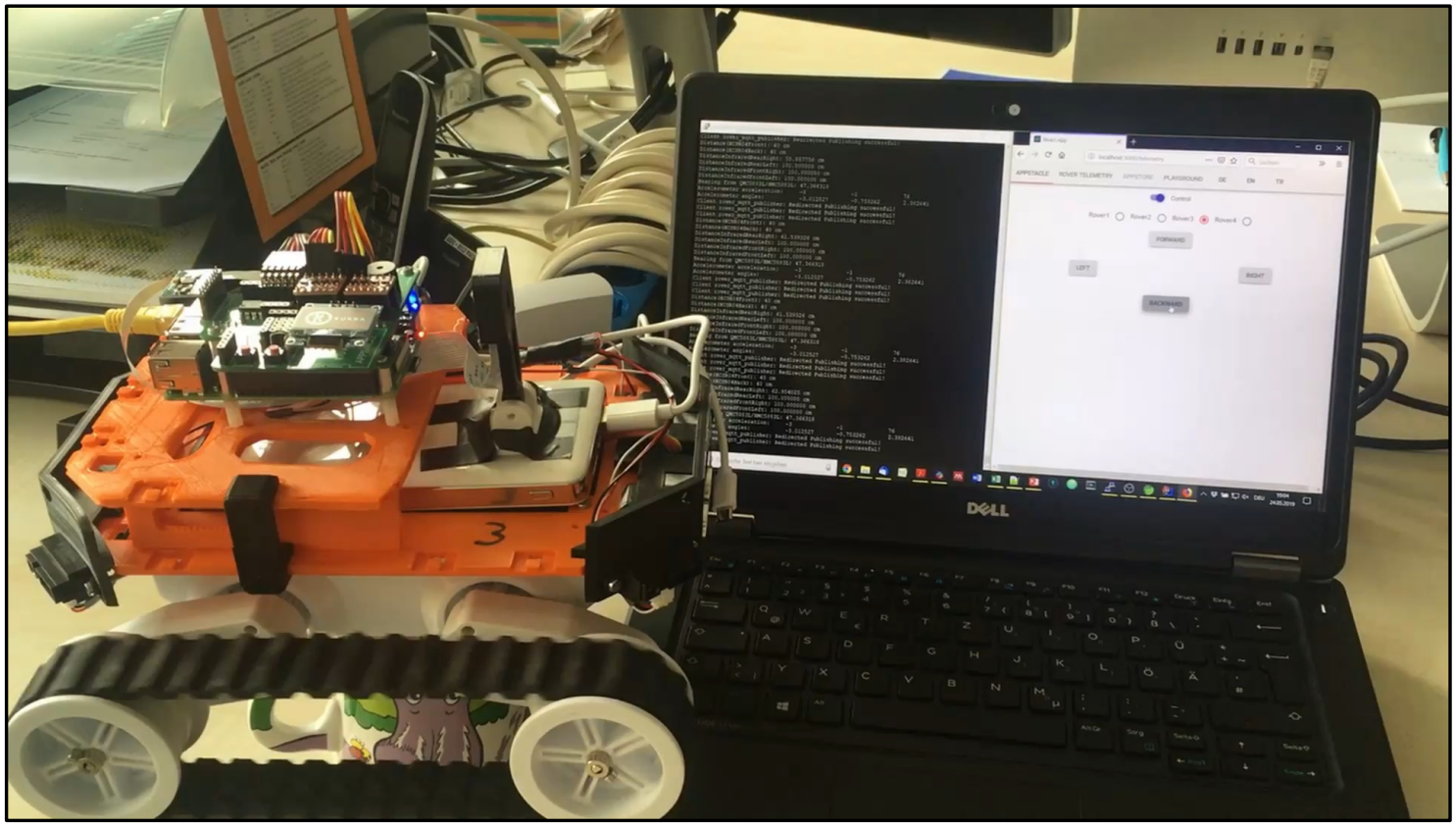
Zeitstempel	Infrarot	Ultraschall	Drehzahl	…
1588691763349	112	23	113	
1588691789971	111	43	113	

Ausschließliche Nutzung von numerischen Werten für eine spätere Visualisierung

Direkte Visualisierung der Werte als Diagramme mittels des Frameworks Grafana möglich

NoSQL

Zeitreihen Datenbank



NoSQL

Polyglotte Datenbankkonzepte

- Polyglotte Persistenz: Einsatz verschiedener Datenbanken für verschiedene Anforderungen an die Datenspeicherung in komplexeren Anwendungen
  - erlaubt unterschiedliche Sichtweisen auf die gespeicherten Daten
- Multi-Modell-Datenbanken vereinen unterschiedliche (NoSQL) Datenmodelle in einer Datenbank
  - Beispiel Mitarbeiterverwaltung:
    - Graph zur Speicherung der Beziehungen zwischen Mitarbeitern und Abteilungen
    - Knoten werden hingegen dokumentenbasiert gespeichert
- NewSQL erweitert relationale Systeme um die Vorzüge von NoSQL (horizontale Skalierbarkeit)
  - ACID Eigenschaften werden in Kombination mit einer Shared-Nothing-Architektur (unabhängige Rechnerknoten, die über ein Netzwerk vernetzt sind) umgesetzt
  - Optimiert für OLTP (Online-Transaction-Processing) Anwendungen

NoSQL

Einheitliche Datenbankschnittstelle

- Bei der Auswahl einer NoSQL-Datenbank besteht die Gefahr einer Herstellerabhängigkeit (Vendor-Lock-in)
  - Produkt wird nicht mehr weiterentwickelt, massive Änderungen an der API, neue Preismodelle, etc.
- Die Jakarta NoSQL Spezifikation [1] hat sich zum Ziel gesetzt, die Integration von Java-Anwendungen und NoSQL-Datenbanken zu vereinfachen
- Schaffung einer einheitlichen Schnittstelle zu NoSQL-Datenbanken
  - Hersteller- und Versionsunabhängige API zur Vermeidung von Vendor-Lock-ins
  - Äquivalent zu JDBC und JPA für SQL
  - Einfacher Wechsel zwischen verschiedenen Datenbanken mittels entsprechender Treiber
- APIs für vier verschiedene NoSQL-Datenbanktypen: Key-Value, Dokumentenorientiert, Graph, Spaltenorientiert

[1] https://projects.eclipse.org/proposals/jakarta-nosql

NoSQL

Event Sourcing

- Event Sourcing [1] ist ein Persistenzmechanismus bei dem alle Änderungen in einer Anwendung als Sequenz von Events gespeichert werden
- Zeigt nicht nur den aktuellen Zustand einer Entität, sondern auch, wie dieser Zustand entstanden ist
  - Chronologischen Liste aller Änderungen (*Domain Events*)
  - Erlaubt es einen beliebigen Zustand aus der Vergangenheit herzustellen
- Beispiel Kontoführung: Kontostand (Zustand) ergibt sich aus den einzelnen Änderungen (Buchungen)
- Vermehrter Einsatz im Zuge von Domain-Driven Design / Microservice-Bewegung

[1] https://microservices.io/patterns/data/event-sourcing.html

Weitere Fragen