Hochschule Karlsruhe
University of
Applied Sciences

Hochschule Karlsruhe – Data Engineering
WS 2021/2022
DSCB330 – Vorlesung 14 – Zusammenfassung

Foto: Quirin Leppert

Themenübersicht

Data Integration

- Data Formats (csv, XML, json)
- Extract, Transform, Load
- Staging

Data Processing

- Relationale Datenbanken
- nicht-relationale Datenbanken
- Resource Description Framework (RDF)
- Ontologien
- Data Warehouse
- MapReduce
- Large Scale Data Analytics – Apache Spark

Data Modelling

- Serialisierung
- OPC UA
- MQTT
- Pub/Sub
- Data pipelines
  - Apache Airflow
  - gRPC

Web-Service Architektur

- Backend for Frontend (BFF)
- Micro Services
- Docker Container

Security

- Security ist wichtig in allen Phasen der Softwareentwicklung und Datenbereitstellung.

Organisatorisches

Klausur

- Donnerstag, 10.02.2022, 9 Uhr, Dauer 90 min
- Abgabe auf Papier
- Es sind keine Hilfsmittel erlaubt.
- 3G-Nachweis
- FFP2-Maske
- Zugang zur Prüfung nur mit unterschriebener Erklärung zur Teilnahme an einer Präsenzprüfung (https://bwsyncandshare.kit.edu/s/BmB56Jk3YBQq98P) und mit Nachweis eines 3G-Status, zusätzlich mit CampusCard bzw. Lichtbildausweis

- Musterlösung der Übungsklausur siehe ILIAS bzw. github

Aufgaben eines Data Engineers

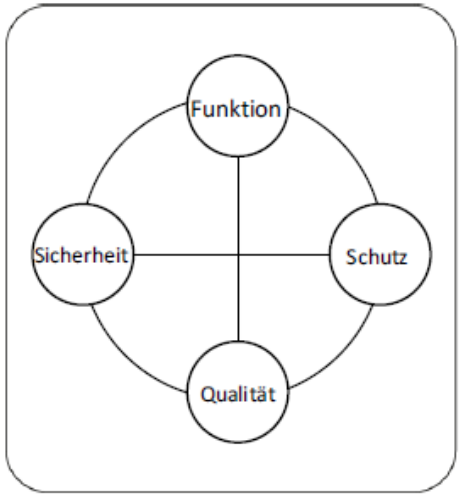
- Instandhaltung der Datenbank-Infrastruktur
- Extraktion, Zusammenführen und Bereitstellung von Daten in einer Pipeline (ETL)

Big Data

- **Volume of data** – Scale of Data
  Die Datenmenge hat in den letzten Jahren rapide zugenommen
- **Variety** – Different Forms of Data
  Daten aus verschiedenen Datenbanken in unterschiedlichen Formaten müssen vereinheitlicht werden
- **Velocity** – Analysis of Streaming Data
  Auswertung von Daten nahezu in Echtzeit
- **Veracity** – Uncertainty of Data
  Unsicherheit der in den Datenbanken abgespeicherten Informationen

Smart Data

- **intelligente Daten**
- haben Informationen über sich selbst
- kennen ihre Bedeutung und Deutung (**Semantik**)



Merkmale von Smart Data

Quellen: Data Engineering with python, Paul Crickard, Packt, Oktober 2020; Data Engineering 4.0, Kompositionale Informationsmodelle für industrielle Anwendungen, Herbert Weber, Springer, 2021

Lesen von csv-Dateien

- Lesen mit Paket csv
- Lesen mit pandas → Sonderzeichen (Umlaute, °) werden nicht eingelesen.

Genauigkeit des Zeitstempels:

- python datetime bis Mikrosekunden
- Pandas bis 1 Nanosekunde
- Numpy bis 1 Attosekunde (10^{-18})
- C# struct DateTime bis 100 Nanosekunden
- MS SQL Server, Datentyp datetime2 bis 100 Nanosekunden

Einlesen von Daten im Binärformat

Abfrage und Umwandeln der Daten im Binärformat mit

```
cursor.execute("SELECT [stream_data] FROM [table]")
row=cursor.fetchone()
valuesnp=numpy.frombuffer(row[0],dtype=np.int16)
```

Resource Description Framework SPARQL

Resource Description Framework (RDF)

+ technische Herangehensweise im Internet zur Formulierung logischer Aussagen über beliebige Dinge (Ressourcen)
+ Syntax für gemeinsamen Datenaustausch
+ W3C-Empfehlung

+ Abfragesprache: SPARQL

+ Beispiel: Recent Events
- instance of (wdt:P31) OR subclass of (wdt:P279*)
- Q1190554: occurrence of a fact or object in space-time

+ Übung 36-SPARQL-Recent-Events

```
#title: Recent events
SELECT ?event ?eventLabel ?date
WHERE
{
  # find events
  ?event wdt:P31/wdt:P279* wd:Q1190554.
  # with a point in time or start date
  OPTIONAL { ?event wdt:P585 ?date. }
  OPTIONAL { ?event wdt:P580 ?date. }
  # but at least one of those
  FILTER(BOUND(?date) && DATATYPE(?date) = xsd:dateTime).
  # not in the future, and not more than 31 days ago
  BIND(NOW() - ?date AS ?distance).
  FILTER(0 <= ?distance && ?distance < 31).
  # and get a label as well
  OPTIONAL {
    ?event rdfs:label ?eventLabel.
    FILTER(LANG(?eventLabel) = "en").
  }
}
# limit to 10 results so we don't timeout
LIMIT 10
```

Quelle: https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples#Recent_events

Machine-to-Machine-Communication (M2M)
Standards für den Datenaustausch

MQTT

+ Pub/sub-Architektur
- Clients veröffentlichen Nachrichten (publish)
- Server verteilt Nachrichten an „subscribed“ clients
+ Keine Semantik

AMQP

+ Senden: Clients produzieren Nachrichten
+ Server verteilt (konfigurierbar) die Nachrichten, z.B. pub/sub, round-robin
+ Clients können den Empfang und die Abarbeitung bestätigen
+ Empfangen: Clients konsumieren Nachrichten

OPC UA

+ Open Platform Communications Unified Architecture
+ plattformunabhängige, service-orientierte Architektur
+ Transport von Maschinendaten (Regelgrößen, Messwerten, Parametern, HMI-Beschreibung, …)
+ inklusive semantischer Beschreibung
+ Integriertes Security-Konzept (UA Security)
+ Standardisierung IEC 62541

Quelle: https://de.wikipedia.org/wiki/MQTT; https://mqtt.org/; https://de.wikipedia.org/wiki/OPC_Unified_Architecture; https://opcfoundation.org/

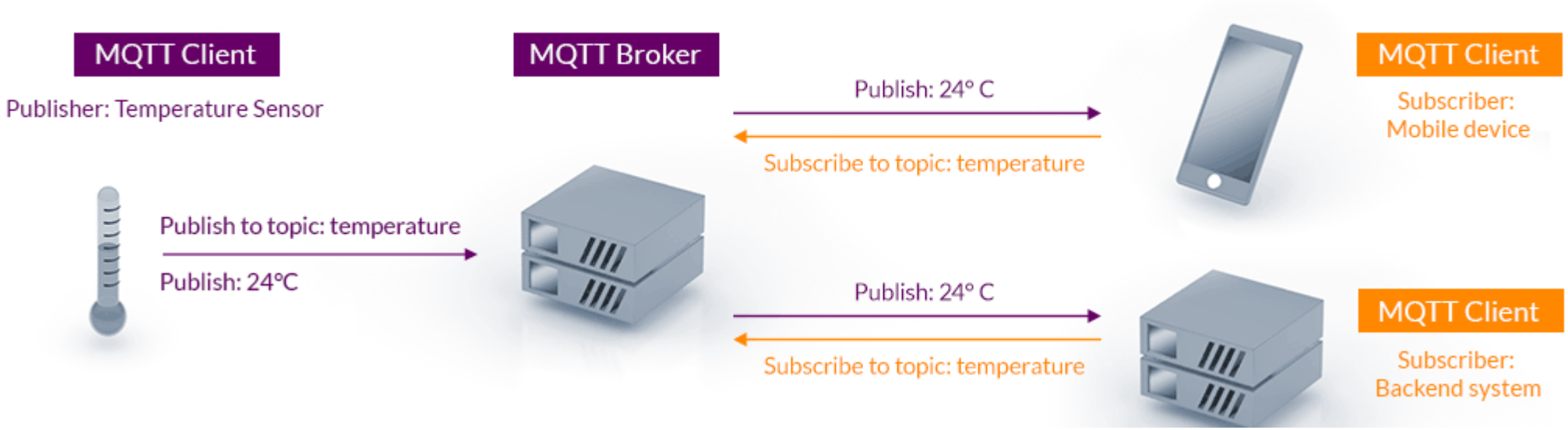
Machine-to-Machine-Communication (M2M)
Message Queuing Telemetry Transport (MQTT)

MQTT Architektur

- Server (Broker) dient als Drehscheibe für die gesamte Kommunikation
- Clients melden sich beim Server an
- Für die Datenübertragung veröffentlicht (publish) der Client eine Nachricht mit einem bestimmten Topic.
- Für den Datenempfang meldet (subscribe) sich der Client für ein oder mehrere Topics an.

Security (optional)

- Unverschlüsselte Kommunikation über Port 1883
- Verschlüsselte Kommunikation (Transport Layer Security (TLS)) über Port 8883
- im Prinzip beliebiger, freier Port nutzbar
- Verifizierung der Echtheit des Servers über ein Zertifikat
- Client-Login über User/Passwort



Quelle: https://mqtt.org/

Machine-to-Machine-Communication (M2M)
Advanced Message Queuing Protocol (AMQP)

Standardisierung

- Standardisierung durch OASIS
- ISO/IEC 19464
- Vision: To become the standard protocol for interoperability between all messaging middleware

Implementierungen

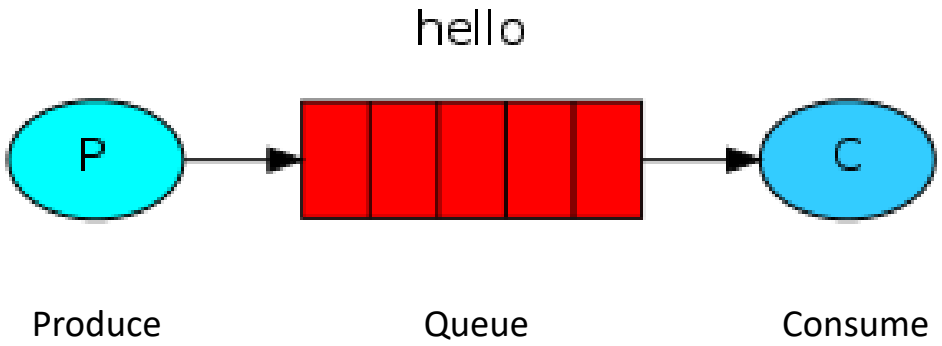
- SwiftMQ
- Microsoft Windows Azure Service Bus
- RabbitMQ …

Einsatz

- Event bus e.g. between Microservices
- Exchange of messages
- Streaming of data

RabbitMQ

- Supports messaging protocols like AMQP 0.9.1/1.0, MQTT…



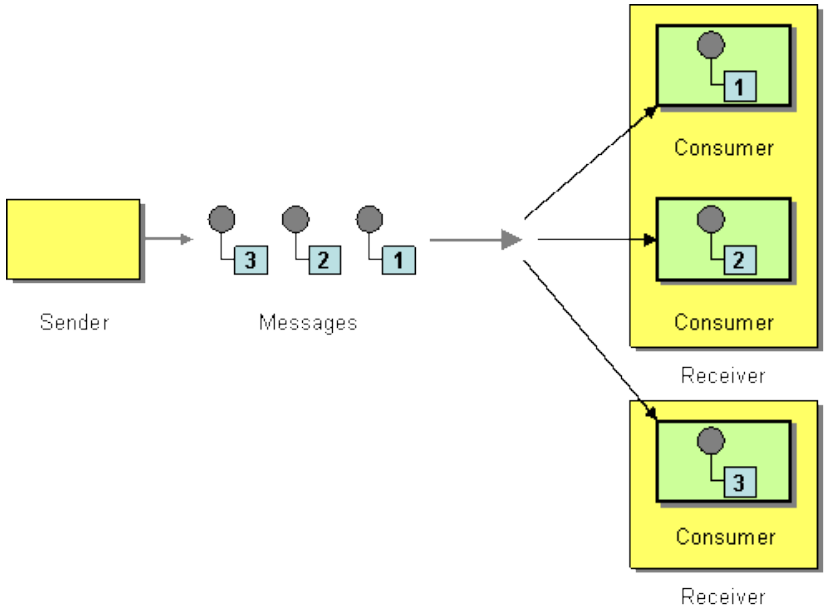
Quelle: https://de.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol; https://www.amqp.org/; https://de.wikipedia.org/wiki/RabbitMQ

Machine-to-Machine-Communication (M2M)
Advanced Message Queuing Protocol (AMQP)

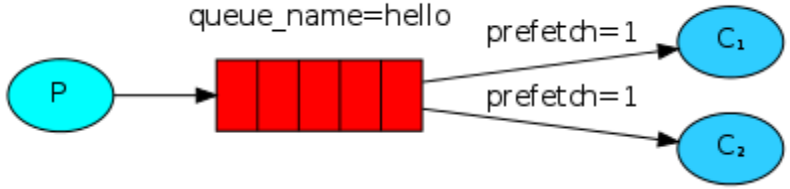
Work Queues

- Verteilung zeitintensiver Tasks auf verschiedene Worker

Competing Consumers Pattern



- Die Queue wird geleert, indem an den nächsten verfügbaren Empfänger eine Nachricht geschickt wird (round-robin). Als Standardverhalten wird eine Nachricht nach dem Verschicken gelöscht.
- Optional: Acknowledgement durch den Empfänger. Sendet der Empfänger kein Acknowledgement (z.B. durch einen Verbindungsabbruch), so sendet der Server die Nachricht an einen anderen Empfänger.

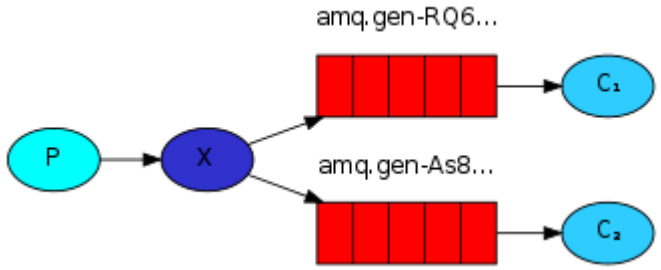


Quelle: https://www.enterpriseintegrationpatterns.com/patterns/messaging/CompetingConsumers.html

Machine-to-Machine-Communication (M2M)
Advanced Message Queuing Protocol (AMQP)

Verteilung einer Nachricht auf mehrere Empfänger

- Publish/subscribe
- Der Sender (Producer) schickt die Nachricht an einen Exchange. Der Exchange leitet die Nachricht weiter. Der Producer muss nicht wissen, was mit der Nachricht passiert.



- Die Nachricht kann über einen Exchange vom Typ “fanout” an beliebig viele Queues weitergeleitet werden (broadcast).

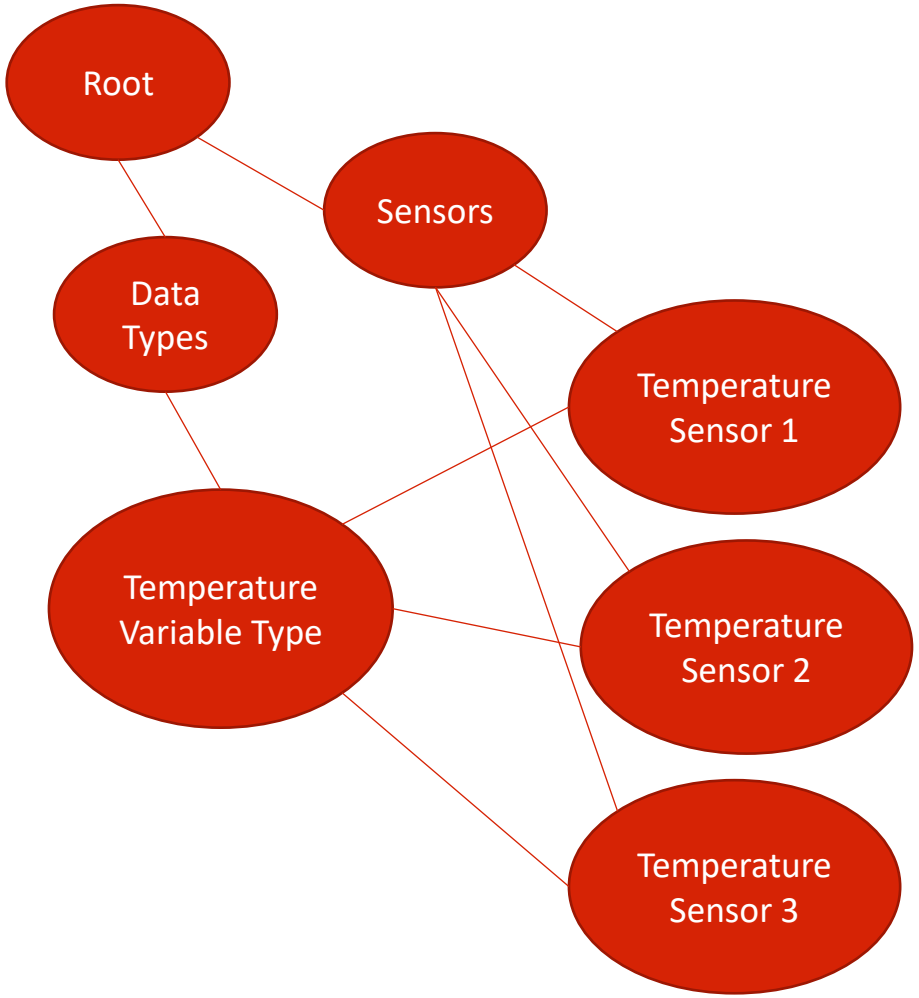
Machine-to-Machine-Communication (M2M)
OPC UA

OPC UA Struktur

- Knotengeflecht (Netzwerk)
- Root (NodeID 84)
- Navigation in mehrere Richtungen
- hasProperty: Blatt (Ende)
- hasComponent: Ast (mit weiteren Verzweigungen)

Nutzen

- Im Dictionary können alle Instanzen „Manufacturer“ abgefragt werden
- Über die Typ-Definition können z.B. alle Temperaturmessgeräte gefunden werden



Quelle: https://opcfoundation.org/about/opc-technologies/opc-ua/; https://reference.opcfoundation.org

ETL – Extract, Transform, Load Transformationsprozess

Alle Aktivitäten zur Umwandlung der operativen Daten in fachlich interpretierbare, entscheidungsorientierte Daten

Teilprozesse

+ Filterung
- Unter der Filterung wird die Extraktion aus den operativen Daten und die Bereinigung syntaktischer oder inhaltlicher Defekte in den zu übernehmenden Daten verstanden.
+ Harmonisierung
- Die Harmonisierung bezeichnet den Prozess der betriebswirtschaftlichen Abstimmung gefilterter Daten.
+ Aggregation
- Die Aggregation ist die Verdichtung gefilterter und harmonisierter Daten.
+ Anreicherung
- Die Bildung und Speicherung betriebswirtschaftlicher Kennzahlen aus gefilterten und harmonisierten Daten wird als Anreicherung bezeichnet.

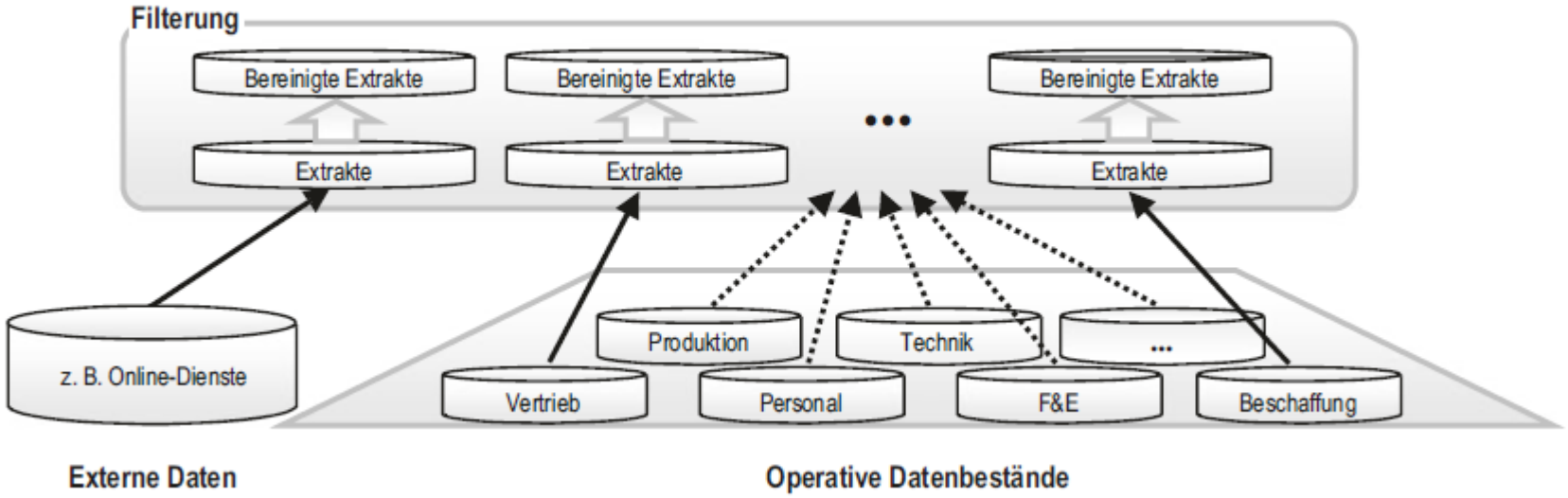
Quelle: Henning Baars, Hans-Georg Kemper; Business Intelligence & Analytics – Grundlagen und praktische Anwendungen; Springer 2021

ETL – Extract, Transform, Load Transformationsprozess

Filterung

- Erste Schicht der Transformation
- Heterogene, interne und externe Datenquellen
- Im Rahmen der Extraktion können die Daten in Extraktionsbereiche (Staging Areas) evtl. in-memory bereitgestellt werden.

- Bei der Bereinigung können syntaktische Mängel (z.B. falsche Formate, Sonderzeichen) automatisiert entfernt.
- Semantische Mängel können z.T. automatisiert entfernt. Manche Fehler sind nur manuell durch Experten identifizierbar.

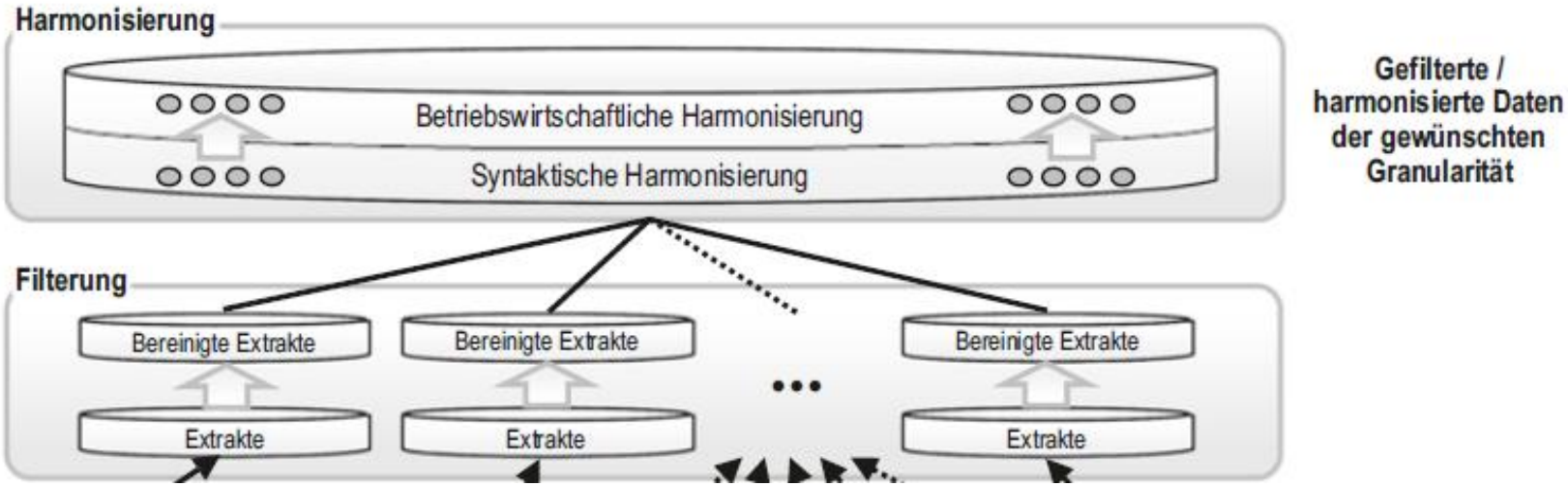


Quelle: Henning Baars, Hans-Georg Kemper; Business Intelligence & Analytics – Grundlagen und praktische Anwendungen; Springer 2021

ETL – Extract, Transform, Load Transformationsprozess

Harmonisierung

- Unterschiedliche Kodierung wird vereinheitlicht.
- Synonyme können z.B. durch Mapping-Tabellen vereinheitlicht werden.
- Homonyme haben unterschiedliche Bedeutungen
- Nach Abschluss der Transformationen der Filterungs- und der Harmonisierungsschicht liegt ein bereinigter und konsistenter Datenbestand vor.



Quelle: Henning Baars, Hans-Georg Kemper; Business Intelligence & Analytics – Grundlagen und praktische Anwendungen; Springer 2021

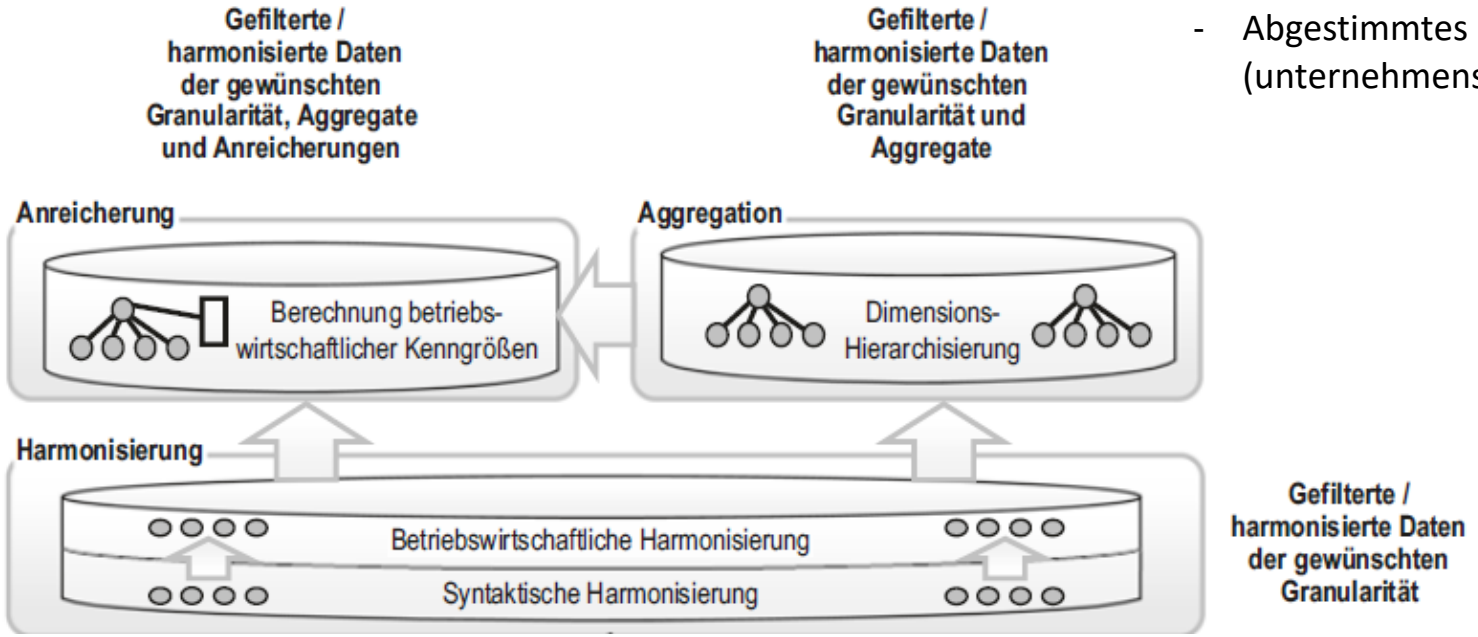
ETL – Extract, Transform, Load Transformationsprozess

Aggregation

- \+ Verdichtung der Daten
- \+ Nutzung von Hierarchien,
  z.B. Kunde → Kundengruppe → Gesamt
  Stadt → Bundesland → Land → Vertriebsregion → Gesamt

Anreicherung

- \+ Berechnung von Kennzahlen
- \- Vorberechnung reduziert Antwortzeitverhalten
- \- Garantierte Konsistenz der kalkulierten Werte (anwendungsübergreifend einmal berechnet, „Single Point of Truth“)
- \- Abgestimmtes betriebswirtschaftliches Instrumentarium (unternehmensweit)



Quelle: Henning Baars, Hans-Georg Kemper; Business Intelligence & Analytics – Grundlagen und praktische Anwendungen; Springer 2021

ETL – Extract, Transform, Load
Eigenschaften einer Production Pipeline

Building idempotent data pipelines

+ **idempotent**
- Data pipelines, die mehrfach ausgeführt werden können, ohne das Ergebnis zu verändern.
- Mathematische Definition mit dem Objekt *a* und einer Verknüpfung •:
  $a \bullet a = a$
+ Vorteile
- Bei Abbruch der Ausführung einer Data pipeline kann die Ausführung einfach erneut angestoßen werden.

Realisierung

+ Guard Clauses
- Abfrage, ob eine Datenbank/Tabelle/Zeile bereits vorhanden ist, bevor diese erzeugt/geschrieben wird.
+ Erzeugung einer neuen Partition/eines neuen Index bei jedem Durchlauf der Pipeline, z.B. durch Nutzung eines Zeitstempels als Suffix

Quelle: Data Engineering with python, Paul Crickard, Packt, Oktober 2020; https://www.red-gate.com/hub/product-learning/flyway/creating-idempotent-ddl-scripts-for-database-migrations

ETL – Extract, Transform, Load
Data pipelines mit Apache Airflow

Apache Airflow

+ entwickelt von Airbnb
+ beinhaltet
  - Web Server
  - Scheduler
  - Metastore
  - Queuing System
  - Executors
+ Zur Konfiguration (Orchestrierung) der Workflows werden Directed Acyclic Graphs (DAGs) verwendet.
+ Verteilung auf viele Nodes möglich
+ Workflows werden mit python scripts erstellt.
+ Apache Airflow wird auf der Google Cloud Platform und Amazon Web Services angeboten.

Motivation

+ Verwaltung und effiziente Ausführung von ETL-Prozessen

Quelle: Data Engineering with python, Paul Crickard, Packt, Oktober 2020

ETL – Extract, Transform, Load
Data pipelines mit Apache Airflow

Parallele Ausführung der Tasks / Remote Execution

+ Bei der einfachsten Installation wird der `SequentialExecutor` genutzt. Tasks werden nacheinander ausgeführt, auch wenn der DAG eine parallele Ausführung zulassen würde.
+ Auf einem Rechner kann eine Parallelisierung mit Hilfe des `LocalExecutor` erreicht werden.

Für die Ausführung in einem Cluster stehen verschiedene Möglichkeiten zur Verfügung, u.a.

+ `Kubernetes Executor`
- Verteilung der Arbeit auf mehrere Docker Container
+ `Celery Executor`
- Verteilung der Tasks mit Hilfe von `Celery`. `Celery` nutzt AMQP (z.B. RabbitMQ) zur internen Kommunikation

Quelle: https://airflow.apache.org/docs/apache-airflow/stable/executor/index.html

Web-Services

Ziel

+ Erstellung eines Web-Services mit folgenden Eigenschaften
+ **Benutzer**
  - Sehr gute User Experience
+ **Entwicklung**
  - Kurze Entwicklungszeiten
  - Gute Wartbarkeit
+ **Production Operations**
  - Einfaches Deployment
  - Hohe Stabilität und Verfügbarkeit
  - Skalierbar
  - Deployment möglich in der Cloud und on-premises
+ **Enterprise**
  - Kostengünstig
+ **Security**

Quelle: https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf

Web-Services
gRPC

+ Entwickelt von Google seit 2015
+ System für Remote Procedure Calls
+ HTTP/2 als Transportmechanismus
+ Protocol Buffers als Interface Description Language
+ Features
- Authentifizierung
- Bi-direktionales Streaming
- Flow control
- Blocking und Non-blocking Bindings
- Cancellation
- Timeouts
- Compiler unterstützt viele Sprachen

Nutzung

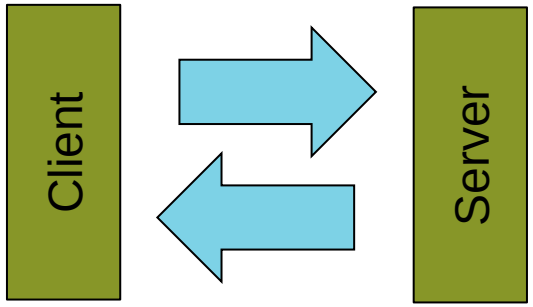
+ Kommunikation zwischen Microservices
+ Verbindung mobiler Geräte zum Backend

Quelle: https://en.wikipedia.org/wiki/GRPC

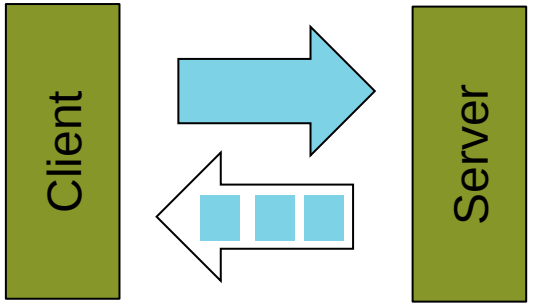
Web-Services
gRPC

- API Varianten in gRPC auf Basis von HTTP/2
- Unterstützung synchroner als auch asynchroner RPC-Aufrufe

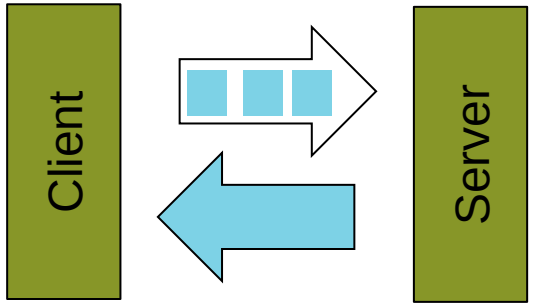
Unär



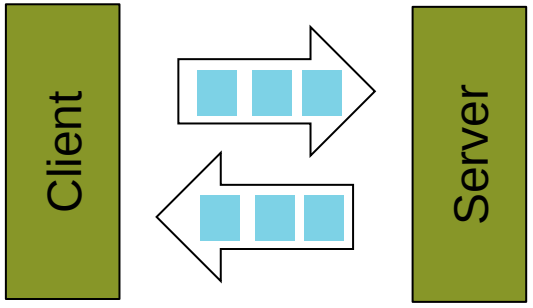
Server Streaming



Client Streaming



Bidirektionales Streaming

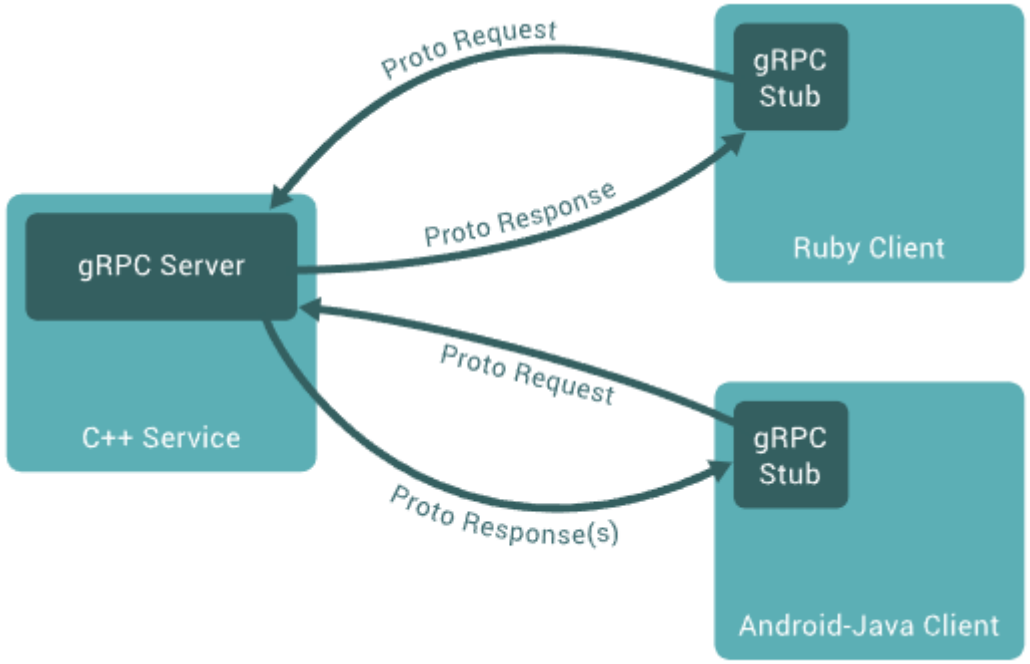



```
service GreetService {
  // Unary
  rpc Greet(GreetRequest) returns (GreetResponse) {};

  // Streaming Server
  rpc GreetManyTimes(GreetManyTimesRequest) returns (stream GreetManyTimesResponse) {};

  // Streaming Client
  rpc LongGreet(stream LongGreetRequest) returns (LongGreetResponse) {};

  // Bi Directional Streaming
  rpc GreatEveryone(stream GreetEveryoneRequest) returns (stream GreetEveryoneResponse) {};
}
```

Quelle: https://en.wikipedia.org/wiki/GRPC

Web-Services
gRPC – Ausführen von Remote Procedure Calls

- Mit gRPC kann ein Client eine Methode auf einem Server (auf einem anderen Rechner) wie ein lokales Objekt aufrufen



Quelle: https://grpc.io/docs/what-is-grpc/introduction/

Web-Services Architektur

Microservice architecture

- Aufbau einer Server-Architektur aus kleinen Services
- Nutzung von HTTP, AMQP und gRPC zur Kommunikation zwischen Services
- Jeder Service implementiert eine end-to-end-domain Fähigkeit (capability) innerhalb einer context boundary (context: Zusammenhang, boundary: Grenze)
- Jeder Service kann autonom entwickelt und unabhängig ausgeführt werden.
- Jeder Microservice bestimmt über seine Daten

Vorteile

- Skalierbarkeit
- Bessere Wartbarkeit in großen, komplexen und hoch-skalierbaren Systemen
- Voneinander unabhängig ausführbare Services
- Continuous integration and continuous delivery
- Test and run microservices in isolation
- Unabhängige Weiterentwicklung unter Berücksichtigung vorhandener Schnittstellen (interfaces and contracts)
- Einfaches Monitoring, Health checks, Watch dogs für jeden Microservice
- Skalierbare Infrastruktur (über Cloud und Orchestrators)
- Schnelle Entwicklung und Auslieferung durch unterschiedliche, parallel arbeitende Teams
- DevOps and CI/CD (Developing and Operations, Continous Integration and continuous deployment)

Domain-Driven Design

- Herangehensweise zur Modellierung komplexer Software
- Iterative Softwareentwicklung
- Enge Zusammenarbeit zwischen Entwicklern und Fachexperten

Bounded Context (Kontextgrenzen)

- beschreiben die Grenzen jedes Kontexts in vielfältiger Hinsicht wie beispielsweise Teamzuordnung, Verwendungszweck, dahinter liegende Datenbankschemata.

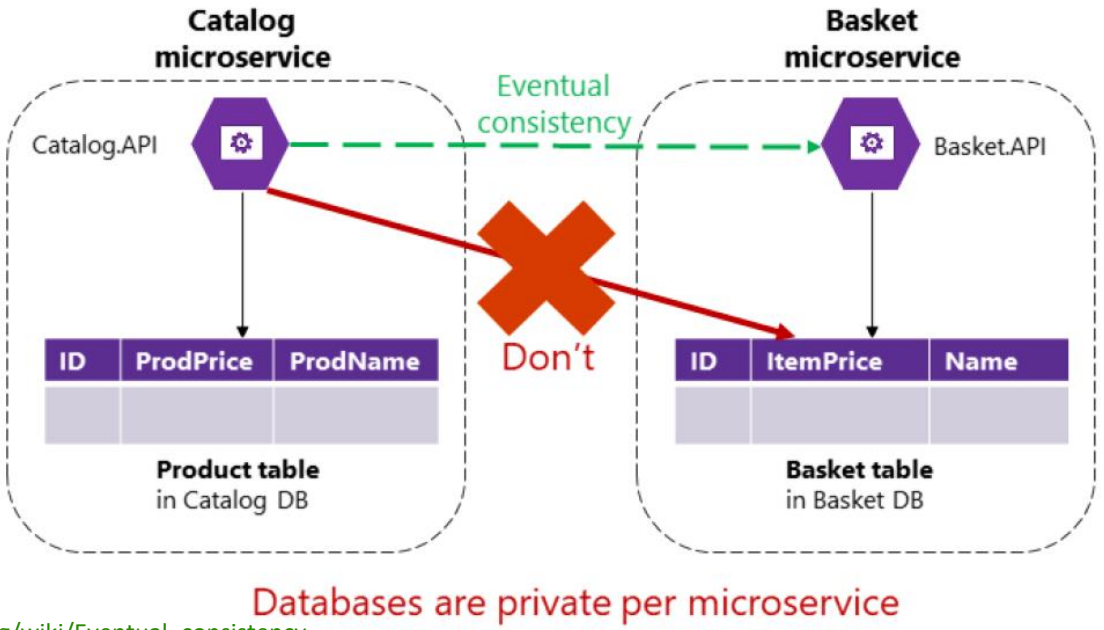
Quelle: https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf; https://de.wikipedia.org/wiki/Domain-driven_Design

Web-Services Architektur

Erhaltung der Konsistenz über verschiedene Microservices

- Jeder Microservice ist für seine Daten verantwortlich
- Daten müssen dennoch über verschiedene Microservices konsistent gehalten werden
- **Beispiel**
- Microservice 1: Katalog-Service inklusive Preise
- Microservice 2: Bestell-Service mit aktuellen Preisen
- **Herausforderung**
- Eine Änderung der Preise im Katalog muss vom Bestell-Service aktualisiert werden. Die Aktualisierung muss im UI dem Kunden transparent gemacht werden.
- CAP theorem:
  Availability contrasts with ACID strong consistency
- Most microservice-based scenarios demand availability and high scalability as opposed to strong consistency.

- **Lösung**
- eventual consistency
- informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value
- Event-driven communication und publish-subscribe-systems



Quelle: https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf; https://en.wikipedia.org/wiki/Eventual_consistency

Web-Services Architektur

Kommunikation über Grenzen von Microservices

- Komponenten können ausfallen
- Ausfälle von Teilen dürfen nicht zum Ausfall des gesamten Systems führen
- Microservices sollten nicht über Abfragen implizit gekoppelt werden
- Verkette Abfragen über Microservices hinweg sollen vermieden werden
- Nutzung asynchroner Abfragen
- Microservices sollen autonom handeln können (d.h. nicht von anderen Microservices abhängen)

Quelle: https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf

Web-Services Architektur

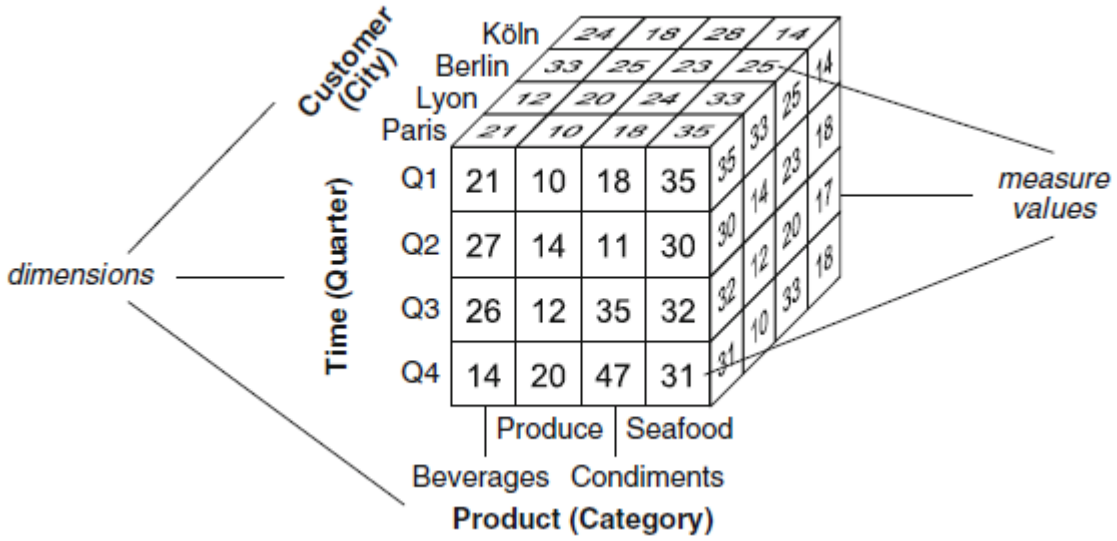
Kommunikation über Grenzen von Microservices

- Komponenten können ausfallen
- Ausfälle von Teilen dürfen nicht zum Ausfall des gesamten Systems führen
- Microservices sollten nicht über Abfragen implizit gekoppelt werden
- Verkette Abfragen über Microservices hinweg sollen vermieden werden
- Nutzung asynchroner Abfragen
- Microservices sollen autonom handeln können (d.h. nicht von anderen Microservices abhängen)

Quelle: https://dotnet.microsoft.com/download/e-book/microservices-architecture/pdf

Data Warehouse Concepts
Multidimensional Model

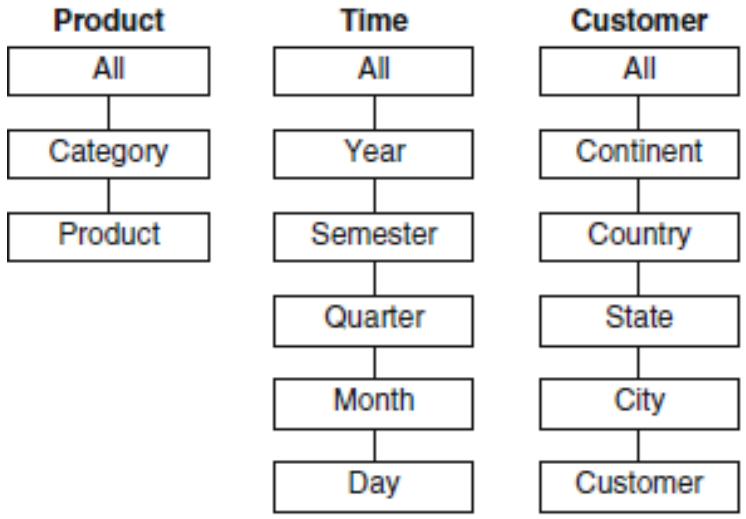
+ **OLAP**: online analytical processing
+ Beantworten von Fragestellungen wie
- Welcher Umsatz wurde im Juni 2010 in der Region Ost mit dem Produkt C220 bei dem Kundentyp Privatkunden erzielt?
+ Multidimensional Model: Daten werden multi-dimensional dargestellt, als **data cube** oder **hypercube**.
+ Ein data cube besteht aus **dimensions** und **facts**.
+ **Fakten** – mit assoziierten numerischen Werten (Measures) – sind numerische Werte, die den Mittelpunkt der Datenanalyse bilden. Aus semantischer Sicht stellen Fakten fachlich definierte Kennzahlen dar.
+ Dimensionen sind Sichtweisen (perspectives) zur Analyse der Daten.
+ Dimension level repräsentiert die Granularität (level of detail).
+ Beispiel: Cube für Verkaufszahlen basierend auf der Northwind Database.
- Drei Dimensionen: Product, Time, Customer.
- Aggregation: Product nach Kategorie, Time nach Quarter, Customer nach der Stadt
+ Instanzen einer Dimension werden **member** genannt.
- Beispiel: Seafood and Beverages sind member der Dimension Product
- **Facts / cells** eines data cubes haben assoziierte numerische Werte (**measures**).



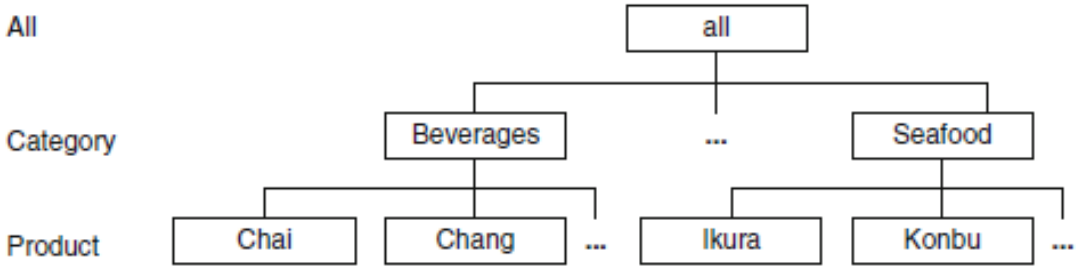
Quelle: A. Vaisman and E. Zimányi, Data Warehouse Systems, Data-Centric Systems and Applications, DOI 10.1007/978-3-642-54655-6, © Springer-Verlag Berlin Heidelberg 2014
Baars, Kemper, Business Intelligence & Analytics – Grundlagen und praktische Anwendungen, 4. Auflage, SpringerVieweg, 2021

Data Warehouse Concepts
Multidimensional Model

- **Hierarchien**



- **Members einer Hierarchie**

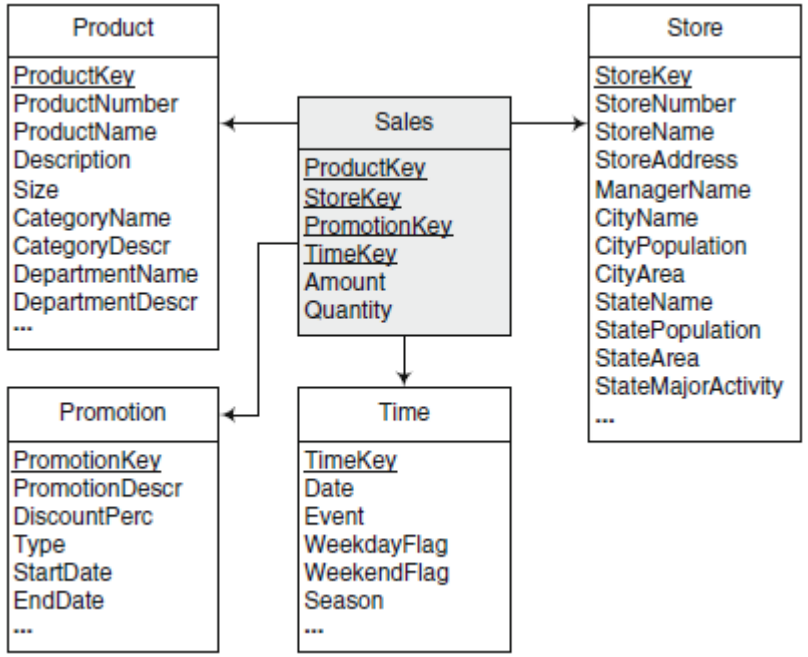


Quelle: A. Vaisman and E. Zimányi, Data Warehouse Systems, Data-Centric Systems and Applications, DOI 10.1007/978-3-642-54655-6, © Springer-Verlag Berlin Heidelberg 2014
Baars, Kemper, Business Intelligence & Analytics – Grundlagen und praktische Anwendungen, 4. Auflage, SpringerVieweg, 2021

Relational Data Warehouse Design

Star Schema

- Zentrale Faktentabelle (fact table) `Sales`
- Dimensionen `Product`, `Promotion`, `Time` und `Store`
- Die Faktentabelle hat als foreign keys die Schlüssel der Dimensions-Tabellen
- Die Faktentabelle ist normalisiert
- Die Dimensions-Tabelle sind i.d.R. nicht normalisiert, d.h. sie können redundante Daten enthalten.
- Beispiel für die Dimension `Product`: Alle Produkte der gleichen Kategorie haben redundante Informationen für die Attribute, die die Kategorie und das Department beschreiben.
- Beispiel in WideWorldImportersDW, Dimension.Supplier, (ORDER BY CATEGORY)



	Supplier Key	WWI Supplier ID	Supplier	Category	Primary Contact	Supplier Reference
1	7	4	Fabrikam, Inc.	Clothing Supplier	Bill Lawson	293092
2	17	4	Fabrikam, Inc.	Clothing Supplier	Bill Lawson	293092
3	3	3	Consolidated Messenger	Courier	Kerstin Pam	209340283
4	16	3	Consolidated Messenger	Courier	Kerstin Pam	209340283
5	27	3	Consolidated Messenger	Courier Services Supplier	Kerstin Pam	209340283
6	26	13	Woodgrove Bank	Financial Services Supplier	Hubert Helms	028034202

Data Warehouse
Data Cube Abfrage

SQL-Befehl zur Abfrage des zwei-dimensionalen Facts

+ Rollup-Operator
- group subtotals in the order given by a list of attributes

```
SELECT ProductKey, CustomerKey, SUM(SalesAmount) as
SumSalesAmount
FROM SalesAmountFact
GROUP BY ROLLUP(ProductKey, CustomerKey)
```

	ProductKey	CustomerKey	SumSalesAmount
1	1	1	100
2	1	2	105
3	1	3	100
4	1	NULL	305
5	2	1	70
6	2	2	60
7	2	3	40
8	2	NULL	170
9	3	1	30
10	3	2	40
11	3	3	50
12	3	NULL	120
13	NULL	NULL	595

+ Cube-Operator
- computes all totals of such a list

```
SELECT ProductKey, CustomerKey, SUM(SalesAmount) as
SumSalesAmount
FROM SalesAmountFact
GROUP BY CUBE(ProductKey, CustomerKey)
```

	ProductKey	CustomerKey	SumSalesAmount
1	1	1	100
2	2	1	70
3	3	1	30
4	NULL	1	200
5	1	2	105
6	2	2	60
7	3	2	40
8	NULL	2	205
9	1	3	100
10	2	3	40
11	3	3	50
12	NULL	3	190
13	NULL	NULL	595
14	1	NULL	305
15	2	NULL	170
16	3	NULL	120

Data Warehouse
Data Cube Abfrage – Window Functions

Window Framing

- Definition der Größe einer Partition zur Berechnung statistischer Größen
- Beispiel 1: Berechnung des Moving Average über drei Monate für die Verkaufszahlen pro Produkt

```
SELECT ProductKey, [Year], [Month], SalesAmount,
AVG(SalesAmount) OVER
(PARTITION BY ProductKey ORDER BY [Year], [Month]
ROWS 2 PRECEDING) AS MovAvg
FROM SalesAmountFact
```

Product Key	Year	Month	Sales Amount	MovAvg
p1	2011	10	100	100
p1	2011	11	105	102.5
p1	2011	12	100	101.67
p2	2011	12	60	60
p2	2012	1	40	50
p2	2012	2	70	56.67
p3	2012	1	30	30
p3	2012	2	50	40
p3	2012	3	40	40

- Beispiel 2: Kumulierte Summe der Verkaufszahlen pro Produkt im jeweiligen Jahr (year-to-date (YTD) sum)

```
SELECT ProductKey, [Year], [Month], SalesAmount,
SUM(SalesAmount) OVER
(PARTITION BY ProductKey, [Year] ORDER BY [Month]
ROWS UNBOUNDED PRECEDING) AS YTD
FROM SalesAmountFact
```

Product Key	Year	Month	Sales Amount	YTD
p1	2011	10	100	100
p1	2011	11	105	205
p1	2011	12	100	305
p2	2011	12	60	60
p2	2012	1	40	40
p2	2012	2	70	110
p3	2012	1	30	30
p3	2012	2	50	80
p3	2012	3	40	120

```
SELECT ProductKey, [Year], [Month], SalesAmount, SUM(SalesAmount) AS YTD
FROM SalesAmountFact S1, SalesAmountFact S2
WHERE S1.ProductKey = S2.ProductKey AND
S1.[Year] = S2.[Year] AND S1.[Month] >= S2.[Month]
```

Quelle: A. Vaisman and E. Zimányi, Data Warehouse Systems, Data-Centric Systems and Applications, DOI 10.1007/978-3-642-54655-6, © Springer-Verlag Berlin Heidelberg 2014

MapReduce

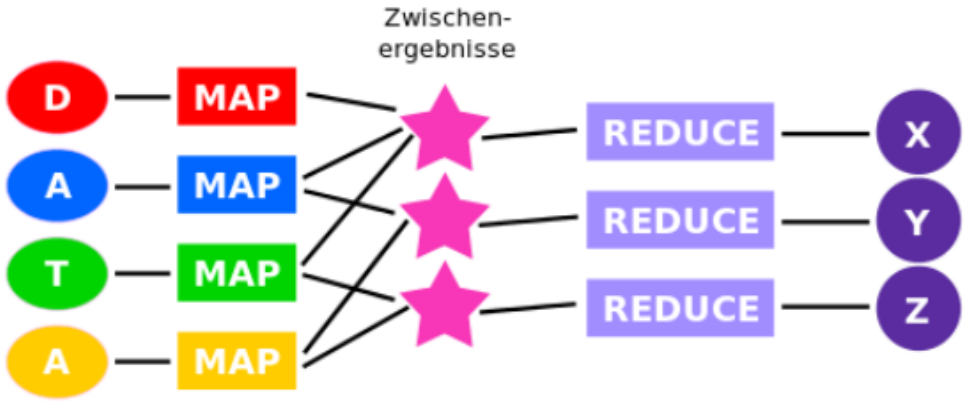
MapReduce

+ Programmiermodell zur Auswertung großer Datenmengen (mehrere Petabyte) verteilt auf Clustern
+ Verarbeitung in drei Phasen
- Map
- Shuffle
- Reduce
+ Map und Reduce werden vom Anwender spezifiziert

Quelle: https://de.wikipedia.org/wiki/MapReduce

MapReduce

Arbeitsweise



+ **Map-Phase:**
- Die Eingabedaten (D, A, T, A) werden auf eine Menge von Map-Prozessen verteilt (illustriert durch bunte Rechtecke), welche jeweils die vom Nutzer bereitgestellte Map-Funktion berechnen.
- Die Map-Prozesse werden idealerweise parallel ausgeführt.
- Jede dieser Map-Instanzen legt Zwischenergebnisse ab (illustriert durch pinkfarbene Sterne).
- Von jeder Map-Instanz fließen Daten in eventuell verschiedene Zwischenergebnisspeicher.

+ **Shuffle-Phase:**
- Die Zwischenergebnisse werden gemäß den Ausgabeschlüsseln, die von der Map-Funktion produziert wurden, neu verteilt, sodass alle Zwischenergebnisse mit demselben Schlüssel im nächsten Schritt auf demselben Computersystem verarbeitet werden.

+ **Reduce-Phase:**
- Für jeden Satz an Zwischenergebnissen berechnet jeweils genau ein Reduce-Prozess (illustriert durch violette Rechtecke) die vom Nutzer bereitgestellte Reduce-Funktion und damit die Ausgabedaten (illustriert durch violette Kreise X, Y und Z).
- Die Reduce-Prozesse werden idealerweise ebenfalls parallel ausgeführt.

Quelle: https://de.wikipedia.org/wiki/MapReduce

MapReduce Übung

Übung zu Map Reduce und Apache Spark

- 35-Apache-Spark-Count-Words
- Implementierung des Word-Count-Algorithmus nach dem MapReduce-Ansatz
- Vergleich mit Apache Spark

Ich wünsche viel Erfolg bei Ihren Klausuren!

Hochschule Karlsruhe
University of
Applied Sciences

www.h-ka.de