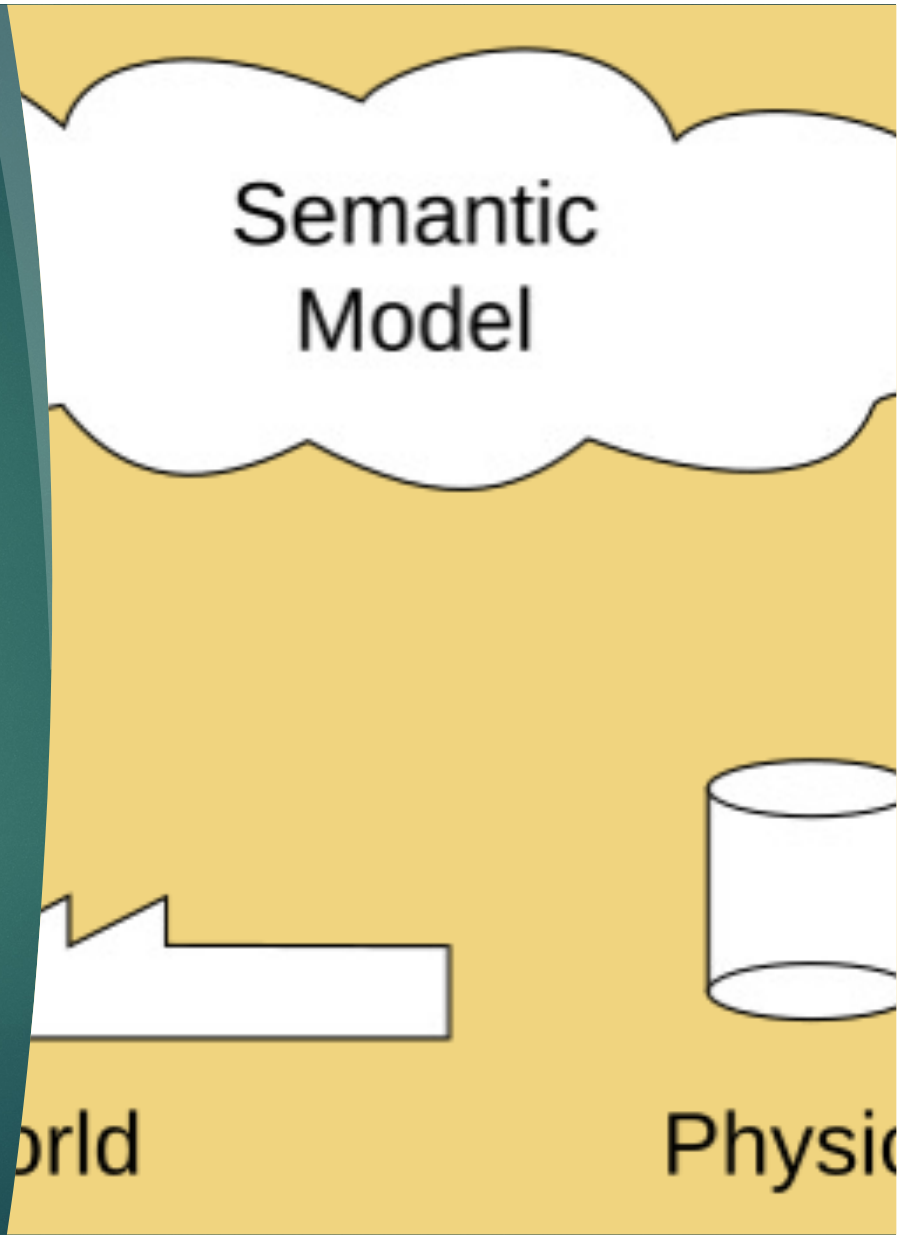
Datenbanken

- Datenbankmodelle
- Daten
- Datenaustausch und Operationen
- Big Data
- Relationale Datenbanken
- ER-Modelle
- Architekturen
- Datenbankentwicklung

Datenbankmodelle

- Überblick
- Hierarchisches Datenbankmodell
- Netzwerkdatenmodell
- Relationales Datenbankmodell
- Objektorientiertes Datenbankmodell
- Dokumentenorientiertes Datenbankmodell
- Graphdatenmodell
- Semantisches Datenmodell
- NoSQL



Datenbankmodelle

Modell	Struktur	Beziehungen	Geeignet für
Hierarchisch	Baum	1:n	Strukturierte feste Hierarchien
Netzwerk	Netz/Graf	n:m	Komplexe, verknüpfte Daten
Relational	Tabellen	1:n, m:n über Joins	Standard-Anwendungen
Objektorientiert	Objekte	Vererbung möglich	OO-Software, CAD, Simulation
Dokumentenorientiert	JSON/XML	Nested/embedded	Webanwendungen, Content
Graph	Knoten + Kanten	Direkt	Soziale Netzwerke, Beziehungen

Hierarchisches Datenbankmodell

- Daten werden in einer baumartigen Struktur gespeichert (Parent-Child-Beziehung)
- Jeder Datensatz hat genau einen übergeordneten Knoten

- **Vorteile:**
  - Schneller Zugriff auf hierarchisch strukturierte Daten
  - Einfaches Datenmodell bei festen Beziehungen
- **Nachteile:**
  - Unflexibel bei komplexen Beziehungen
  - Nur 1:n-Beziehungen möglich

Netzwerkdatenmodell

- Daten sind netzartig miteinander verknüpft, ähnlich einem Graphen
- Unterstützt mehrere Beziehungen pro Knoten (n:m möglich)

- **Vorteile:**
  - Flexible Verknüpfungen
  - Effiziente Datenzugriffe mit Navigationslogik
- **Nachteile:**
  - Komplexe Modellierung
  - Kaum verwendet in modernen Systemen

Relationales Datenbankmodell

- Daten werden in Tabellen (Relationen) gespeichert
- Beziehungen über Primär- und Fremdschlüssel
- Basierend auf mathematischer Relationenlogik

- **Vorteile:**
  - Standardisiert (SQL)
  - Flexibel, skalierbar, weit verbreitet
  - Einfache Abfrage durch SQL
- **Nachteile:**
  - Relativ schlechter bei unstrukturierten Daten
  - Performance bei großen Joins kann kritisch werden

Objektorientiertes Datenbankmodell

- Kombination von Objektorientierung und Datenhaltung
- Daten werden als Objekte mit Attributen und Methoden gespeichert, durch Objektverweise verbunden

- **Vorteile:**
  - Ideal für objektorientierte Programmierung
  - Daten und Verhalten in einer Einheit
- **Nachteile:**
  - Komplexere Umsetzung
  - Geringere Verbreitung als relationale Modelle

Dokumentenorientiertes Datenbankmodell

- Daten als strukturierte Dokumente, meist im JSON-Format
- Schemafrei: unterschiedliche Felder pro Dokument möglich

- **Vorteile:**
  - Flexibel, gut für unstrukturierte oder semi-strukturierte Daten
  - Horizontale Skalierung einfach möglich
- **Nachteile:**
  - Kein fester Schema → Validierung muss manuell erfolgen
  - Kein SQL-Standard → unterschiedliche Abfragesprachen

Graphdatenmodell

- Daten bestehen aus Knoten (Entitys), Kanten (Beziehungen) und Eigenschaften (Properties)
- Optimal für Beziehungsdaten (z. B. soziale Netzwerke)

- **Vorteile:**
  - Hochperformant bei Beziehungsabfragen
  - Ideal für Empfehlungssysteme, soziale Netze etc.
- **Nachteile:**
  - Weniger geeignet für klassische Geschäftsdaten
  - Abfragen komplexer

Semantisches Datenmodell

- beschreibt Daten nicht nur strukturell, sondern auch in Bezug auf ihre Bedeutung (Semantik) und Zusammenhänge
- was bedeuten Daten, wie verhalten sie sich zueinander und in welchem Kontext stehen sie

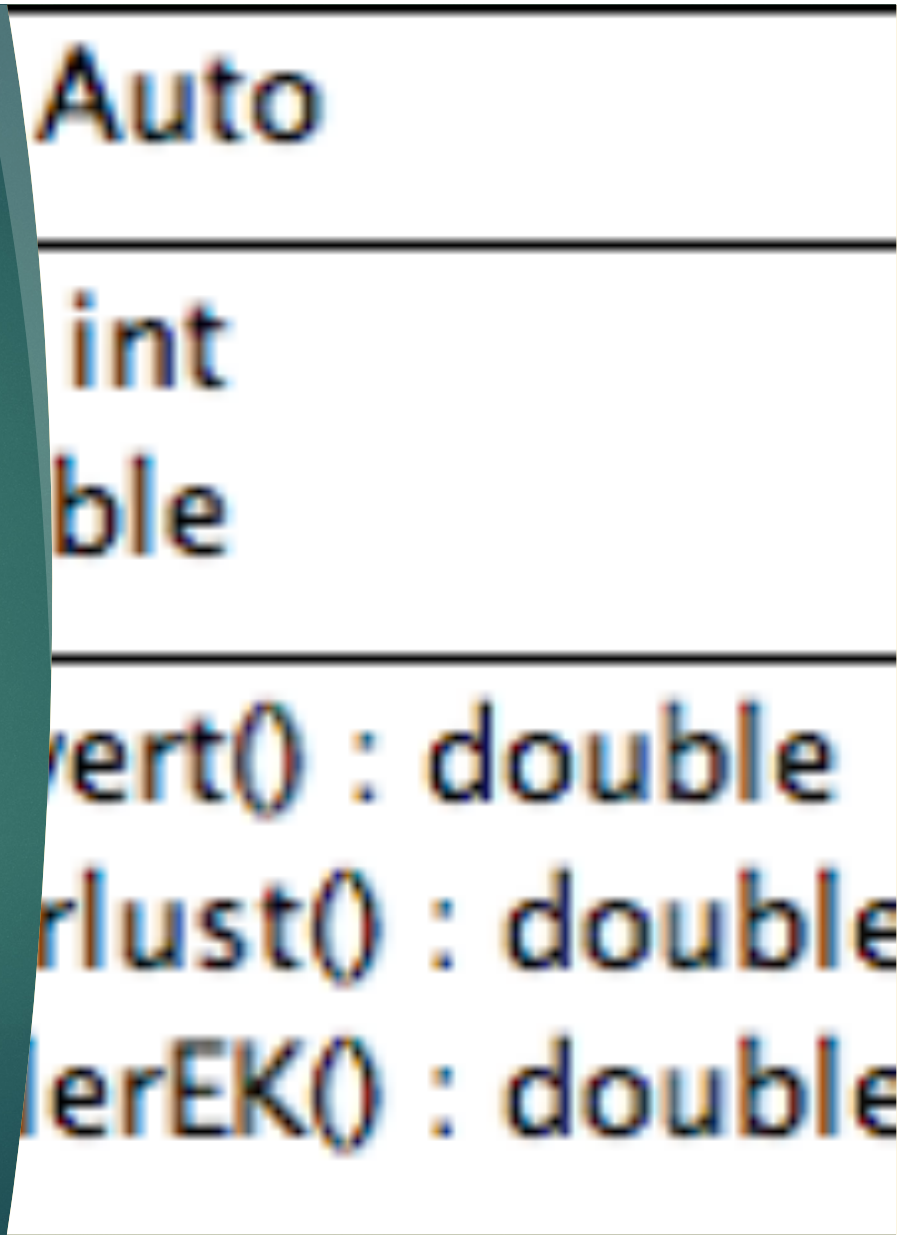
- **Vorteile:**
  - Mehr Ausdruckskraft, nah an der Realwelt
  - Ideal für Empfehlungssysteme, soziale Netze etc.
- **Nachteile:**
  - Komplexe Tools nötig
  - Nur konzeptionell

Begriffsklärung: NoSQL

- Ansatz zur Datenbankverwaltung
- Bezeichnung für alle Nicht-Relationalen Datenbanken
- Speicherung der Daten auf natürlichere und flexiblere Weise

Daten

- Datentypen
- Open Data
- Klassische Datenquellen
- Dateinbasierte Datenquellen
- Echtzeit- und Streaming-Datenquellen
- Web- und Cloudbasierte Datenquellen
- Weitere Datenquellen



Datentypen

Kategorie	SQL-Datentyp (Beispiel)	Beschreibung
Boolesche Werte	BOOL / BOOLEAN	Wahrheitswerte: TRUE oder FALSE
Ganzzahl	INT / INTEGER / SMALLINT / BIGINT	Ganzzahlige Zahlenwerte (positiv/negativ)
Gleitkommawerte	FLOAT / REAL / DOUBLE	Gleitkommazahlen (für z. B. wissenschaftliche Berechnungen)
Währung	DECIMAL(10,2) / NUMERIC	Genauer Datentyp für Währungsangaben (mit fester Nachkommastelle)
Datumswerte	DATE / TIME / DATETIME / TIMESTAMP	Datum und/oder Uhrzeit
Text (feste Länge)	CHAR(n)	Texte mit exakt definierter Länge
Text (variable Länge)	VARCHAR(n)	Texte beliebiger Länge bis zur Obergrenze
BLOB	BLOB / BYTEA	Binäre große Objekte wie Bilder, Dateien, Audio
Geokoordinaten	GEOGRAPHY / GEOMETRY / POINT	Speichert Koordinaten wie Längen-/Breitengrad

Open Data

- Daten, die:
  - **frei verfügbar, nutzbar, veränderbar und weiterverteilbar** sind – ohne Einschränkungen durch Urheberrechte, Patente oder andere Mechanismen
  - idealerweise **maschinenlesbar**, **strukturiert** und **standardisiert** bereitgestellt werden
- Kriterien für Open Data (nach Open Definition):
  - **Zugänglichkeit**: Daten sind vollständig, kostenlos und ohne technische Hürden verfügbar
  - **Nutzungsfreiheit**: Jeder darf die Daten verwenden – auch kommerziell
  - **Weiterverwendung und Verbreitung**: Veränderung und Weitergabe sind erlaubt
  - **Namensnennung** (bei Bedarf): Quelle muss ggf. genannt werden
  - **Maschinenlesbarkeit**: Daten sollten automatisiert verarbeitet werden können

CSV
XML
JSON
RDF
…

Klassische Datenquellen

Relationale Datenbanken

- Strukturierte Daten mit festem Schema (Tabellen, Spalten, Schlüssel)
- Beispiele: MySQL, PostgreSQL, Oracle, MS SQL Server
- Einsatz: Geschäftsanwendungen, ERP, CRM, Buchhaltung

Schemafreie (NoSQL) Datenbanken

- Flexiblere Struktur, oft dokumentbasiert (z. B. JSON)
- Beispiele: MongoDB, CouchDB, Firebase
- Einsatz: Webanwendungen, IoT, Big Data

Dateibasierte Datenquellen

CSV, Excel, XML, JSON-Dateien

- Werden oft manuell oder automatisch erzeugt/exportiert
- Einsatz: Datenimporte, Schnittstellen, Reportings
- Vorteile: leicht lesbar, plattformunabhängig
- Herausforderung: Fehleranfällig bei manueller Pflege

Bilder, Audio, Video (z. B. BLOBs)

- Rohdaten in Binärform
- Einsatz: Dokumentenmanagement, Medienarchive

Echtzeit- und Streaming-Datenquellen

Sensoren (IoT-Geräte)

- Temperatur, Bewegung, Füllstand, GPS etc.
- Verbindung über Protokolle wie MQTT, HTTP, Bluetooth
- Daten oft unstrukturiert, kontinuierlich

Live-Streams (z. B. Video, Maschinendaten)

- Echtzeit-Daten, große Datenmengen
- Einsatz: Überwachung, Industrie 4.0, Verkehrssysteme

Web- und Cloudbasierte Datenquellen

APIs (REST, SOAP, GraphQL)

- Zugriff auf externe Dienste (z. B. Wetter, Logistik, Open Data)
- Strukturierte Daten in Echtzeit über das Internet

Cloud-Speicher / Dienste

- Google Drive, Dropbox, Azure Storage
- Verwendung für geteilte Datenhaltung, automatisierte Workflows

Weitere Datenquellen

Datenquelle	Beschreibung
Manuelle Eingaben	Webformulare, Eingabemasken
Maschinendaten	Produktionsanlagen, Roboter, SCADA-Systeme
Logdateien	Server- oder Anwendungsmeldungen
Social Media	Daten über APIs von Twitter, Instagram etc.
E-Mail	Inhalte und Metadaten automatisiert verarbeitbar

Datenaustausch und Operationen

- API
- ACID-Prinzipien für Transaktionen in Relationalen Datenbanken
- BASE
- Begriffsklärung: Streaming Analytics
- Maßnahmen bei Lösch-/Aktualisierungsvorgängen
- Tiefergehende Datenbankobjekte
- Replikation

API

Application Programming Interface / Programmierschnittstelle

- ermöglicht es verschiedenen Programmen, Systeme oder Dienste, miteinander zu kommunizieren, Daten auszutauschen und Funktionen bereitzustellen, ohne dass die internen Abläufe bekannt sein müssen
- REST, SOAP, GraphQL, Webhooks

Bestandteil	Bedeutung
Endpoint	URL oder Pfad, unter dem eine Funktion aufgerufen wird
Request	Anfrage an die API, z. B. per HTTP (GET, POST, PUT, DELETE)
Response	Antwort der API (meist JSON, XML oder Text)
Parameter	Zusätzliche Daten zur Anfrage (Query-Parameter, Body, Header etc.)
Statuscode	Antwortcode wie 200 OK, 404 Not Found, 500 Server Error

ACID-Prinzipien für Transaktionen in Relationalen Datenbanken

Atomicity (Atomarität)	Consistency (Konsistenz)	Isolation (Isolation)	Durability (Dauerhaftigkeit)
• Eine Transaktion ist eine atomare Einheit – entweder sie wird vollständig durchgeführt, oder gar nicht. Teilausführungen sind nicht erlaubt. • Transaktionssteuerung, Rollback	• Eine Transaktion überführt die Datenbank von einem konsistenten Zustand in einen anderen – alle Integritätsbedingungen bleiben gewahrt. • Constraints, Validierungen	• Transaktionen beeinflussen sich nicht gegenseitig, selbst wenn sie gleichzeitig ablaufen. Zwischenergebnisse dürfen nicht sichtbar sein. • Sperren	• Nach einem erfolgreichen Commit bleiben alle Änderungen dauerhaft erhalten – auch bei Systemabstürzen oder Stromausfall. • Logs, Commit-Protokoll, persistente Speicherung

Problemfall ohne Isolation:
„Dirty Read“: Eine Transaktion liest Daten, die eine andere noch nicht abgeschlossen hat
„Phantom Read“: Wiederholte Abfragen liefern unterschiedliche Ergebnisse

BASE

- Gegenentwurf zu den ACID-Prinzipien klassischer relationaler Datenbanken
- vor allem bei verteilten NoSQL-Datenbanken verwendet, um hohe Verfügbarkeit und Skalierbarkeit zu ermöglichen

Basically Available	Soft State	Eventual Consistency
• Das System ist grundsätzlich verfügbar – auch bei Teilausfällen	• Der Zustand des Systems kann sich über Zeit auch ohne neue Eingaben ändern (z. B. durch Replikation)	• Die Daten werden nicht sofort, aber schlussendlich konsistent – also mit Verzögerung

Begriffsklärung: Streaming Analytics

- Analyse von Datenströmen in Echtzeit
- während die Daten erzeugt und empfangen werden – nicht erst im Nachhinein

- **Realisierung:**
  - Daten stammen meist aus kontinuierlichen Quellen (z. B. Sensoren, Logs, GPS, Social Media)
  - Daten werden „on the fly“ verarbeitet, ohne sie vorher dauerhaft zu speichern
  - Systeme analysieren, filtern, aggregieren und reagieren innerhalb von Sekundenbruchteilen

Maßnahmen bei Lösch-/Aktualisierungsvorgängen

Aktionstyp	Wirkung bei Löschung oder Änderung im referenzierten Datensatz
CASCADE	Automatisches Löschen abhängiger Datensätze
SET NULL	Fremdschlüssel wird auf NULL gesetzt
RESTRICT/DENY	Verhindert das Löschen, wenn abhängige Daten existieren
NO ACTION	Verhalten wie RESTRICT, aber wird zur Laufzeit geprüft

Tiefergehende Datenbankobjekte

Index

- Beschleunigt Lesezugriffe auf Spalten
- B-Tree, Hash-Index, Fulltext-Index

Stored Procedure

- Ausführung vorgefertigter SQL-Prozeduren auf dem Datenbankserver
- Kapselung von Geschäftslogik

Trigger

- Ausführung automatischer Aktionen bei bestimmten Ereignissen auf Tabellen

Sequence

- Generiert fortlaufende IDs (z. B. für Primärschlüssel)

Replikation

- Kopie von Datenbankinhalten auf andere Server zur Leistungssteigerung, Lastverteilung und Ausfallsicherheit

- **Typen:**

Typ	Beschreibung
Master-Slave	Nur Master schreibt, Slaves lesen
Multi-Master	Alle können schreiben → Konfliktlösung nötig
Snapshot	Periodische Momentaufnahme
Log-basierte	Überträgt Änderungen inkrementell

- **Herausforderungen:**
  - Konsistenzsicherung
  - Replikationsverzögerung
  - Konflikte bei Multi-Master-Szenarien

Big Data – optional in APT2

- Begriffsklärung
- Merkmale
- Herausforderungen
- 3-V-Modell
- 4-V-Modell
- Begriffsklärung: Map/Reduce
- Data Lake

Big Data

- bezieht sich auf riesige und komplexe Datensätze, die mit herkömmlichen Methoden der Datenverarbeitung nicht mehr effizient bearbeitet werden können
- Big Data Analysen helfen Unternehmen, Muster und Trends zu erkennen, fundierte Entscheidungen zu treffen und neue Geschäftsmöglichkeiten zu erschaffen

- **Anwendungsbereiche:**

Kundenanalyse	Produktentwicklung	Risikomanagement	Gesundheitswesen	Logistik und Transport
• Verhaltensmuster erkennen, personalisierte Angebote erstellen, Kundenbindung verbessern	• Neue Produkte und Dienstleistungen entwickeln, die auf Kundenbedürfnissen basieren	• Potenzielle Risiken erkennen und minimieren, z.B. im Finanzwesen oder der Versicherungsbranche	• Krankheitsmuster erkennen, personalisierte Behandlungen entwickeln, Effizienzsteigerung im Gesundheitswesen	• Routen optimieren, Lagerbestände verwalten, Lieferketten effizienter gestalten

Big Data - Merkmale

Volumen	Vielfalt	Geschwindigkeit	Verlässlichkeit
• enorme Menge an Daten, die oft petabyte (10^15 Bytes) oder Exabyte (10^18 Bytes) umfassen	• Big Data umfasst strukturierte (z.B. Datenbanken), unstrukturierte (z.B. Texte, Bilder, Videos) und semi-strukturierte Daten	• Daten werden in Echtzeit oder nahezu Echtzeit generiert und müssen schnell verarbeitet werden	• Qualität und Zuverlässigkeit der Daten ist ein wichtiges Kriterium, da Fehler oder Ungenauigkeiten zu falschen Schlussfolgerungen führen können

Big Data - Herausforderungen

- **Speicherung und Verarbeitung:**
  - Die Datenmengen erfordern spezielle Infrastruktur und Technologien
- **Datenschutz und Sicherheit:**
  - Umgang mit sensiblen Daten erfordert strenge Datenschutzrichtlinien und Sicherheitsmaßnahmen
- **Analyse und Interpretation:**
  - Komplexe Daten erfordern spezialisierte Tools und Fachwissen

3V-Modell

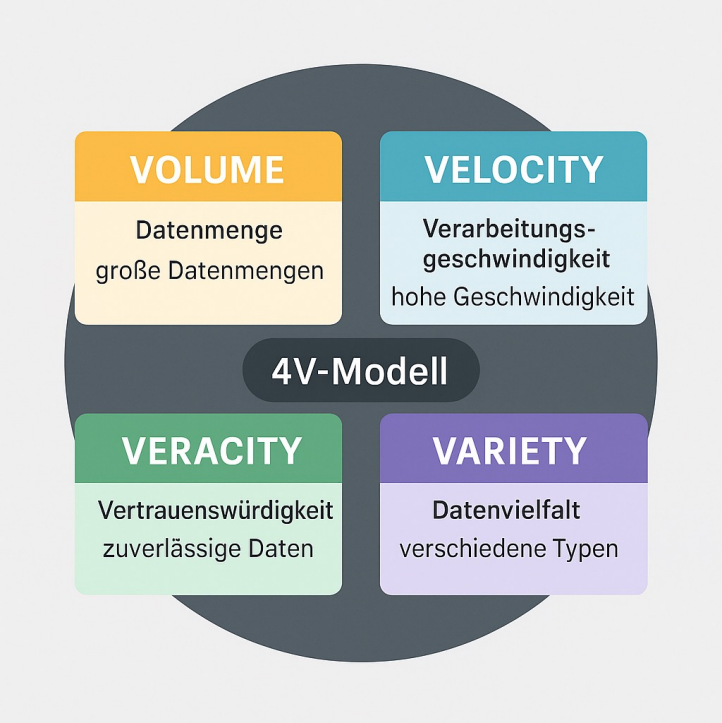
- Big-Data-Modell
- beschreibt die wesentlichen Merkmale von Datenmengen im Kontext großer und komplexer Datenverarbeitungssysteme
- dient zur **Einordnung, Bewertung und Planung** von Systemen, die mit sogenannten **Big Data** umgehen

- **Ziel**:
  - Klassifikation von Datenprojekten (z. B. für die Auswahl geeigneter Technologien)
  - Grundlage für Architekturentscheidungen (z. B. klassische Datenbank vs. NoSQL vs. Data Lake)
  - Bewertung von Herausforderungen und Anforderungen im Datenmanagement

4V-Modell

- Erweiterung des klassischen 3V-Modells aus dem Big-Data-Kontext
- ergänzt die ursprünglichen Dimensionen um einen wichtigen Aspekt: **Veracity** (Wahrhaftigkeit/Vertrauenswürdigkeit der Daten)

- **Vorteile gegenüber V3:**
  - Realistischere Betrachtung moderner Datenprobleme
  - Fokussiert auf Datenqualität und Vertrauenswürdigkeit
  - Hilft bei Risikoabschätzung in datenbasierten Systemen (z. B. KI)



Begriffsklärung: Map/Reduce

- Programmiermodell und Ausführungsprinzip zur Verarbeitung und Aggregation großer Datenmengen in verteilten Systemen
- besteht aus zwei aufeinanderfolgenden Schritten:

- **Map (Zuordnung)**
  - Zerlegt Daten in kleinere Teile und ordnet sie Schlüssel/Wert-Paaren zu
    → Ziel: Verteilen & Vorstrukturieren
- **Reduce (Reduktion)**
  - Fasst Werte mit gleichem Schlüssel zusammen (Aggregation)
    → Ziel: Zusammenfassen & Berechnen

Data Lake

- zentrales, flexibles Datenspeichersystem, das große Mengen an rohen, unstrukturierten, semi-strukturierten und strukturierten Daten aus verschiedensten Quellen aufnimmt und speichert
- Speicherung der Daten in ihrem ursprünglichen Format
- Analyse, Strukturierung oder Transformation erfolgt je nach Bedarf („Schema on Read“)

Merkmal	Beschreibung
Datentypen	Strukturiert (SQL), semi-strukturiert (JSON, XML), unstrukturiert (PDFs, Bilder, Videos)
Schema-Verwendung	Schema on Read – Struktur wird erst bei Analyse festgelegt
Speicherarchitektur	Häufig objektbasiert
Skalierbarkeit	Sehr hoch – für Petabyte große Datenmengen ausgelegt
Kosten	Geringe Speicherkosten, da keine aufwändige Strukturierung nötig ist

Data Lake - Einsatzszenarien

Datensammlung für Data Science/Machine Learning

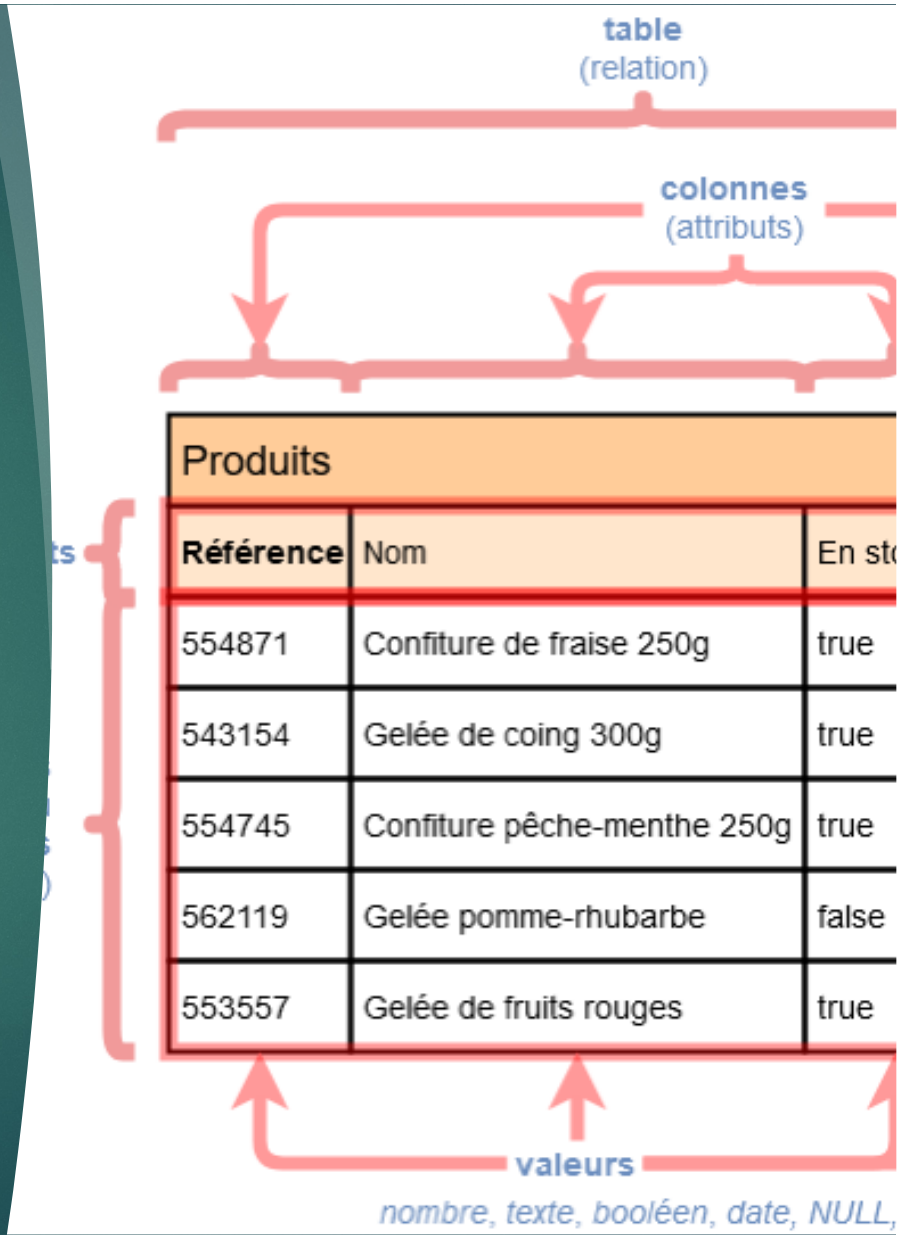
IoT-Daten, Logdateien, Sensorströme

Archivierung großer Mengen historischer Daten

Flexible Analyse in der Forschung oder Produktentwicklung

Relationale Datenbanken

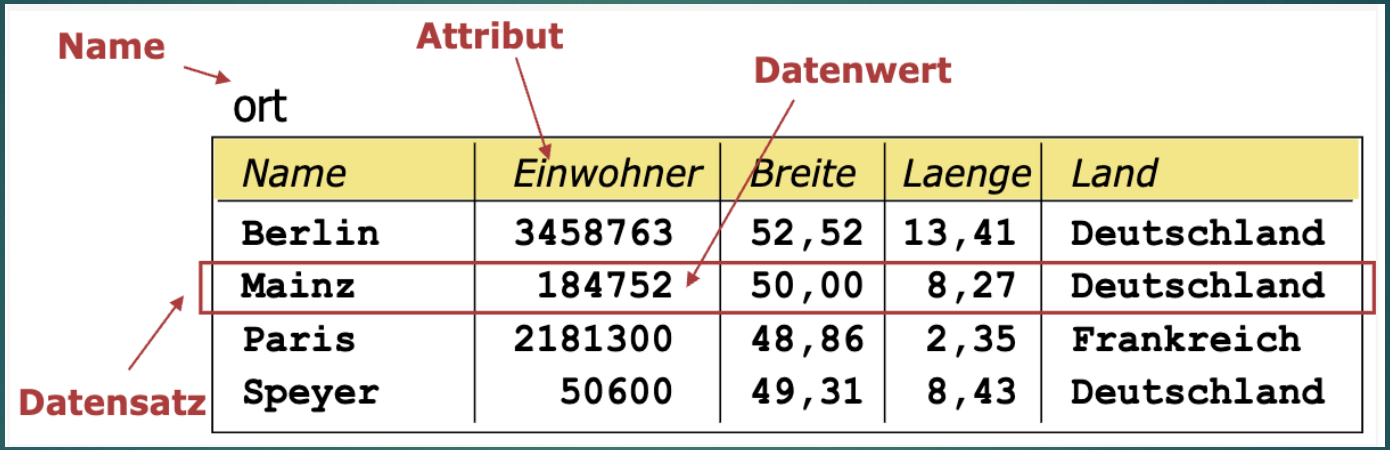
- Begriffsklärung
- Herausforderungen im Design
- Anomalien
- Kardinalitäten
- Schlüsseltypen
- Referentielle Integrität
- Grenzen einer Relationalen DB
- Datenbankentwurf



Relationale Datenbanken

- besteht aus einer oder mehreren Tabellen (Relationen), in denen Daten abgespeichert werden können
- jede Zeile (Tupel) stellt einen eigenen Datensatz (Record) dar
- die Spalten jeder Zeilen sind Attribute, also Eigenschaften der abgespeicherten Daten
- Die Domäne/Domain ist dabei der Wertebereich, den die Attribute annehmen könne
- durch ein Relationenschema lassen sich sowohl die Anzahl als auch die Typen der Attributen einer Relation festlegen

Relationale Datenbanken



Herausforderungen im Design

- **Redundanzen**
  - Daten liegen mehrfach (redundant) in einer Datenbank oder über mehrere Tabellen verteilt vor
- **Inkonsistenzen**
  - mehrere Dinge, die als gültig angesehen werden sollen, nicht miteinander vereinbar sind
- **Anomalien**
  - Einfügeanomalie
  - Änderungsanomalie
  - Löschanomalie

Anomalien

Anomalie	Beschreibung	Beispiel
Einfügeanomalie (Insert Anomaly)	Neue Daten können nicht gespeichert werden, ohne andere unnötige Daten mit einzufügen	Ein neuer Kurs kann nicht eingetragen werden, weil noch kein Teilnehmer bekannt ist
Änderungsanomalie (Update Anomaly)	Eine Änderung muss an mehreren Stellen vorgenommen werden – sonst entsteht Inkonsistenz	Die Telefonnummer eines Dozenten steht in mehreren Zeilen und wird nur an einer Stelle geändert
Löschanomalie (Delete Anomaly)	Durch das Löschen von Daten gehen zusätzliche, relevante Informationen verloren	Wenn der letzte Teilnehmer eines Kurses gelöscht wird, verschwindet der Kurs gleich mit

Kardinalitäten

Kardinalität	Bedeutung	Beispiel
1:1	Ein Datensatz A gehört zu einem B	Person ↔ Personalausweis
1:n	Ein A zu mehreren B	Kunde → mehrere Bestellungen
m:n	Mehrere A zu mehreren B	Schüler ↔ Kurse (über Zwischentabelle)

- **Zu beachten:**
  - m:n-Beziehungen müssen aufgelöst werden (z. B. durch Zwischentabellen)
- **Herausforderung**:
  - Komplexe Beziehungen machen Modellierung und Abfragen aufwändiger

Schlüsseltypen

Primärschlüssel (Primary Key)	Fremdschlüssel (Foreign Key)	Künstlicher Schlüssel:	Natürlicher Schlüssel:	Anonymer Schlüssel:
• Eindeutiger Identifikator eines Datensatzes (z. B. KundenID)	• Verweis auf einen Primärschlüssel in einer anderen Tabelle – bildet Relationen ab	• Künstlich erzeugt, z. B. KundenID = 1001 • Stabil, anonymisiert • Kein inhaltlicher Bezug	• Echter Wert, z. B. E-Mail-Adresse • Verständlich • ann sich ändern → ungeeignet als Primärschlüsse	• Dient nur zur internen Identifikation, keine Rückschlüsse auf Person/Daten

Referentielle Integrität

- Beziehungen zwischen Tabellen sind konsistent
- Ein Fremdschlüssel muss auf gültigen Datensatz zeigen oder leer sein

- Wenn nicht: Anomalien

Grenzen einer Relationalen DB

Nachteil	Bedeutung
Komplexe JOINs	Abfragen über viele Tabellen können langsam oder unübersichtlich werden
Nicht ideal für unstrukturierte Daten	Medien, Text, Sensoren oft schwer modellierbar
Horizontal schlecht skalierbar	Verteilung auf viele Server ist schwieriger als bei NoSQL

3 Phasen im Datenbankentwurf

- Konzeptioneller DB-Entwurf
  - Modell der Miniwelt in einem konzeptionellen Datenmodell (z.B. dem Entity-Relationship Modell)
  - Statt “konzeptionell” kann man auch “konzeptuell” sagen
- Logischer DB-Entwurf
  - transformiert das Schema in das Datenmodell des DBMS
  - Z. B. relationales Modell
- Physischer DB-Entwurf
  - Verbesserung der Performance des endgültigen Systems

ER-Modelle

- Begriffsklärung
- Arten von ER-Modellen
- Crow‘s Foot Notation

Begriffsklärung

Entity-Relationship-Modell

- grundlegendes Werkzeug zur konzeptionellen Datenmodellierung relationaler Datenbanken
- Diagramm zur Beschreibung der Struktur von Daten in einer Datenbank
- Zeigt Entitäten (oder „Objekte“) einer Datenbank, deren Attribute und die Beziehungen zwischen ihnen

Arten von ER-Modellen

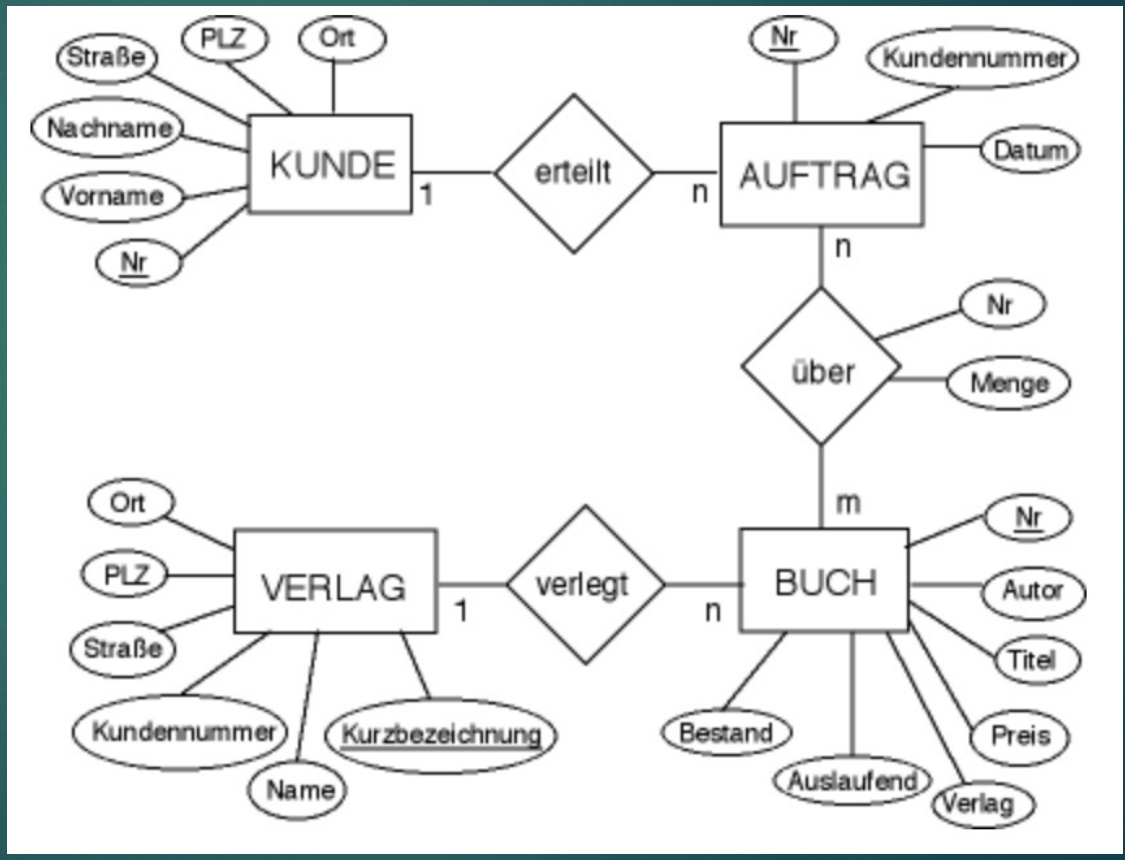
ER-Diagramm:

- Darstellung der Entitäten und Beziehungen zwischen ihnen in Form von Symbolen und Linien
- Hohes Abstraktionsniveau

Tabellenmodelle:

- Darstellung der Entitäten und Beziehungen zwischen ihnen in Form von Tabellen
- Tieferes Abstraktionsniveau
- Dient der Planung einer konkreten Implementierung einer Datenbank

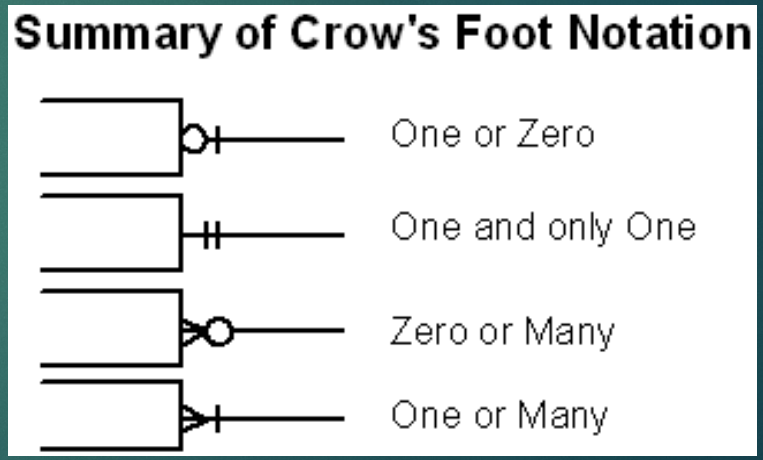
ER-Diagramm - CHEN-Notation



Quelle

Crow‘s Foot Notation

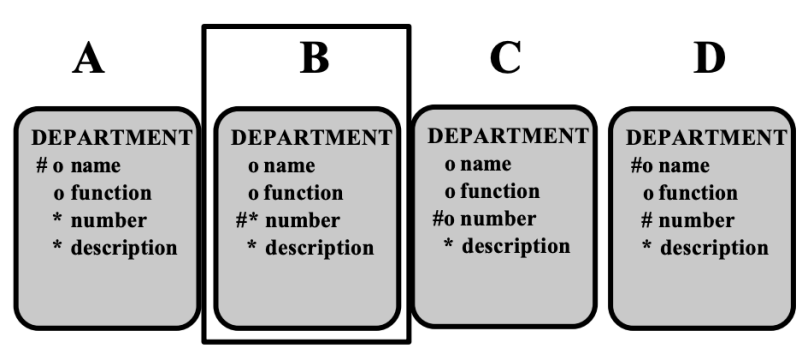
- weit verbreitete grafische Notation zur Darstellung von ER-Diagrammen
- Siehe Zusatzdokument



die "Füße" **(„<“)** stehen für **„viele“**
das Strich-Symbol **(|)** steht für **„genau eins“**
der Kreis **(O)** steht für **„keins“**

Oracle Notation

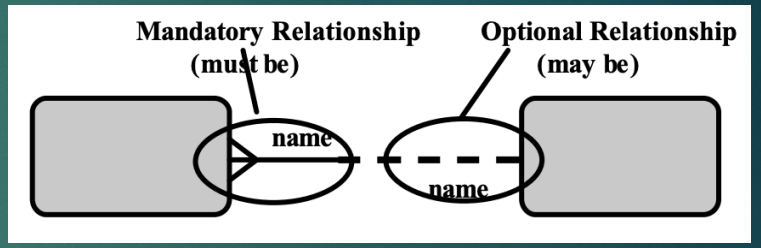
- Entitäten mit Attributen werden wie folgt dargestellt:



  - # - Primary Key
  - * - Pflicht-Attribut
  - ° - optionales Attribut

Quelle

- Beziehungen werden über Crow`s Foot Notation und dem Relationen-Name dargestellt:

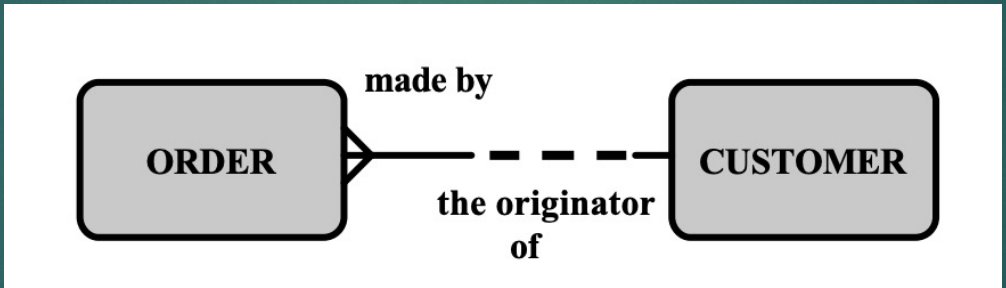


Quelle

  - Durchgezogene Linie
    - Must be-Beziehung
  - Gestrichelte Linie
    - May be-Beziehung

Oracle Notation - Beispiel

- „Each ORDER must be made by one and only one CUSTOMER.“
- „Each CUSTOMER may be the originator of one or more ORDERs.“



Quelle

Architektur

- 3 Ebenen Architektur
- Zentral versus verteilt
- CAP-Theorem
- Cloud-Datenbankarchitekturen

3 Ebenen Architektur

- basiert auf mehreren Schichten, die zusammenarbeiten, um Daten effizient und sicher zu verwalten:
  - **Physische Schicht:** Speicherort der Daten auf der Hardware
  - **Logische Schicht:** Datenstruktur, wie Tabellen und Indizes
  - **Benutzer-Schicht:** Schnittstelle, über die Benutzer auf die Daten zugreifen
- Innerhalb dieser Schichten arbeiten zusätzliche Komponenten wie das Datenbank-Management-System (DBMS) zur Verwaltung der Daten

- Vorteile einer durchdachten Architektur:
  - **Effizienz**: Optimierte Abfragen und Datenzugriffe.
  - **Sicherheit**: Schutz sensibler Daten vor unbefugtem Zugriff.
  - **Skalierbarkeit**: Handhabung wachsender Datenmengen und Benutzender
  - **einfache Wartung und Anpassung**

Zentral versus verteilt

Zentrale Datenbank

- Alle Daten liegen auf einem einzigen Server/System
- **Vorteile:** Einfach zu verwalten, konsistent
  Nachteile: Single Point of Failure, begrenzte Skalierbarkeit

Verteilte Datenbank

- Daten sind über mehrere physische Standorte verteilt
- Nutzer merken von der Verteilung nichts
- **Typen:**
  - **Homogen verteilt** (gleiches DBMS an allen Knoten)
  - **Heterogen verteilt** (verschiedene DBMS, z. B. Oracle + PostgreSQL)
- **Vorteile:** Ausfallsicherheit, Nähe zum Nutzer, horizontale Skalierung
  Nachteile: Komplexität bei Konsistenz, Synchronisation

CAP-Theorem

- fundamentales Konzept der verteilten Systeme, insbesondere von verteilten Datenbanksystemen
- beschreibt, dass in einem verteilten System 2, jedoch niemals alle drei der folgenden Eigenschaften gleichzeitig garantiert werden können:

Kürzel	Begriff	Bedeutung
C	Consistency (Konsistenz)	Alle Knoten liefern immer den gleichen Datenstand nach einer Schreiboperation
A	Availability (Verfügbarkeit)	Jeder Anfrage wird innerhalb einer angemessenen Zeitspanne beantwortet – auch bei Ausfällen
P	Partition Tolerance (Partitionstoleranz)	Das System funktioniert trotz Ausfällen oder Kommunikationsunterbrechungen zwischen Knoten weiter

CAP - Klassische Kombinationen

Kombi	Bedeutung	Beispielsysteme
CA	Konsistent & verfügbar, aber nicht partitionstolerant	Klassische SQL-Datenbank auf einem Server
CP	Konsistent & partitionstolerant, aber ggf. nicht verfügbar	HBase, MongoDB (stark konfigurierbar)
AP	Verfügbar & partitionstolerant, aber ggf. nicht konsistent	CouchDB, Cassandra, DynamoDB

Viele NoSQL-Datenbanken priorisieren **Availability + Partition Tolerance (AP)**, da sie hochverfügbar über viele Server verteilt sein sollen.

Konsistenz wird dann oft **eventual consistency** genannt:
Die Daten sind irgendwann konsistent, aber nicht sofort nach dem Schreiben.

Cloud-Datenbankarchitekturen

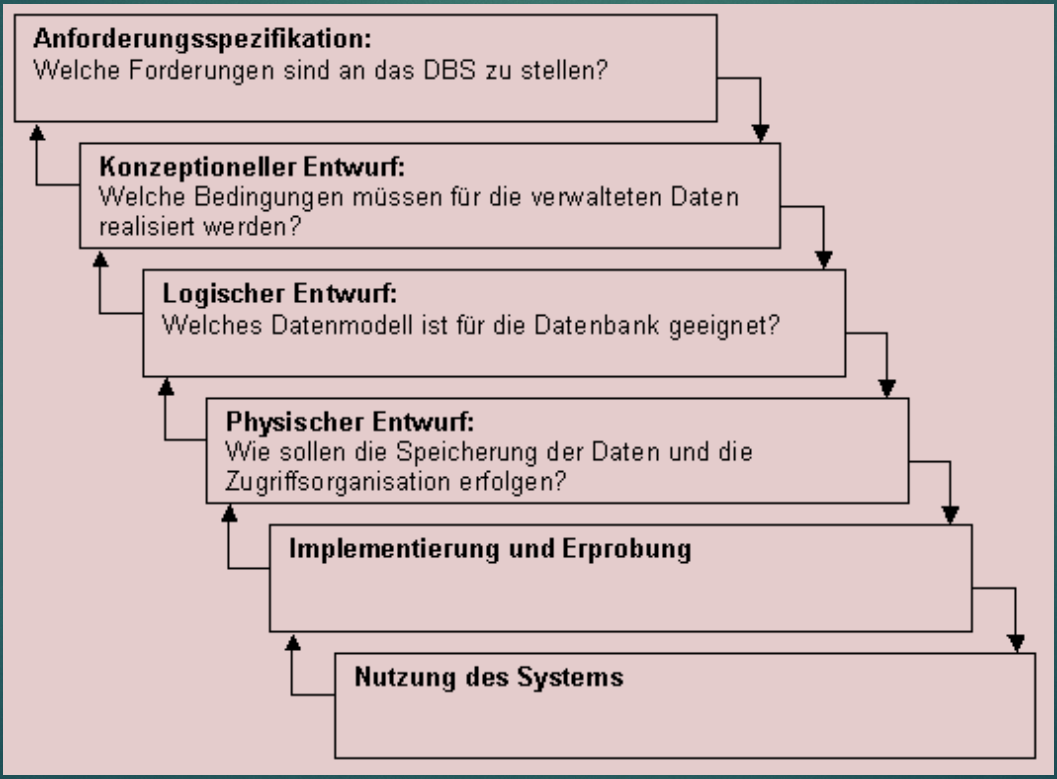
- DBaaS (Database-as-a-Service)
- Datenbank läuft in der Cloud

- **Vorteile:**
  - Skalierbar
  - Wartungsarm
  - hohe Verfügbarkeit
- **Nachteile:**
  - Abhängigkeit vom Anbieter
  - Datenschutzthemen

Datenbankentwicklung

- Phasen der DB-Entwicklung
  - Externe Phase
  - Konzeptionelle Phase
  - Logische Phase
  - Physische Phase

Phasen der DB-Entwicklung



Externe Phase (Informationsbeschaffung)

- Ermittlung der Anforderungen und Informationsbedarfe der NutzerInnen und Anwendungen
- **Inhalte:**
  - Gespräche mit Stakeholdern (z. B. Abteilungsleiter, Endanwender)
  - Sammlung der benötigten Datenarten
  - Definition von Sichten (z. B. für Vertrieb, Buchhaltung …)
  - Abgrenzung der relevanten Datenbereiche
- **Ergebnis:**
  - Anforderungskatalog
  - Beschreibung der benötigten Daten aus fachlicher Sicht

Konzeptionelle Phase (Semantisches Modell)

- Modellierung der fachlichen Datenstruktur unabhängig von der Technik
- **Inhalte:**
  - Modellierung mit semantischen Modellen, v. a. Entity-Relationship-Modell
  - Definition von Entitäten, Beziehungen, Attributen, Kardinalitäten
  - Abbildung der realen Welt in einer abstrakten, verständlichen Form
- **Ergebnis:**
  - Konzeptionelles Datenmodell (z. B. ER-Diagramm)
  - Grundlage für Kommunikation mit Fachabteilungen

Logische Phase (Datenmodell)

- Überführung des konzeptionellen Modells in ein datenbankspezifisches Format
- **Inhalte:**
  - Übersetzung des ER-Modells in ein relationales Modell (Tabellen, Primär-/Fremdschlüssel)
  - Auswahl eines passenden logischen Datenmodells (z. B. relational, dokumentenbasiert)
  - Prüfung auf Normalformen (1NF–3NF etc.)
- **Ergebnis:**
  - Logisches Datenbankschema mit allen Tabellen, Attributen, Beziehungen und Integritätsbedingungen

Physische Phase (Datenbankschema)

- Technische Umsetzung auf einem konkreten Datenbanksystem (z. B. PostgreSQL, Oracle, MongoDB)
- **Inhalte:**
  - Implementierung des Schemas
  - Festlegung von Datentypen, Indizes, Partitionierung
  - Performanceoptimierung
  - Technische Constraints
- **Ergebnis:**
  - Laufendes Datenbanksystem mit implementiertem Schema
  - Einsatzbereit für Anwendungen