



Data Mining, Data Storing und Zugriff

Schulung am 09.12.2024 bis 11.12.2024

Ihr Trainer: Nicolas Kuhaupt

Überblick Zeiten

9:00 – 10:30



10:45 – 12:00



13:00 – 14:45



15:00 – 16:30

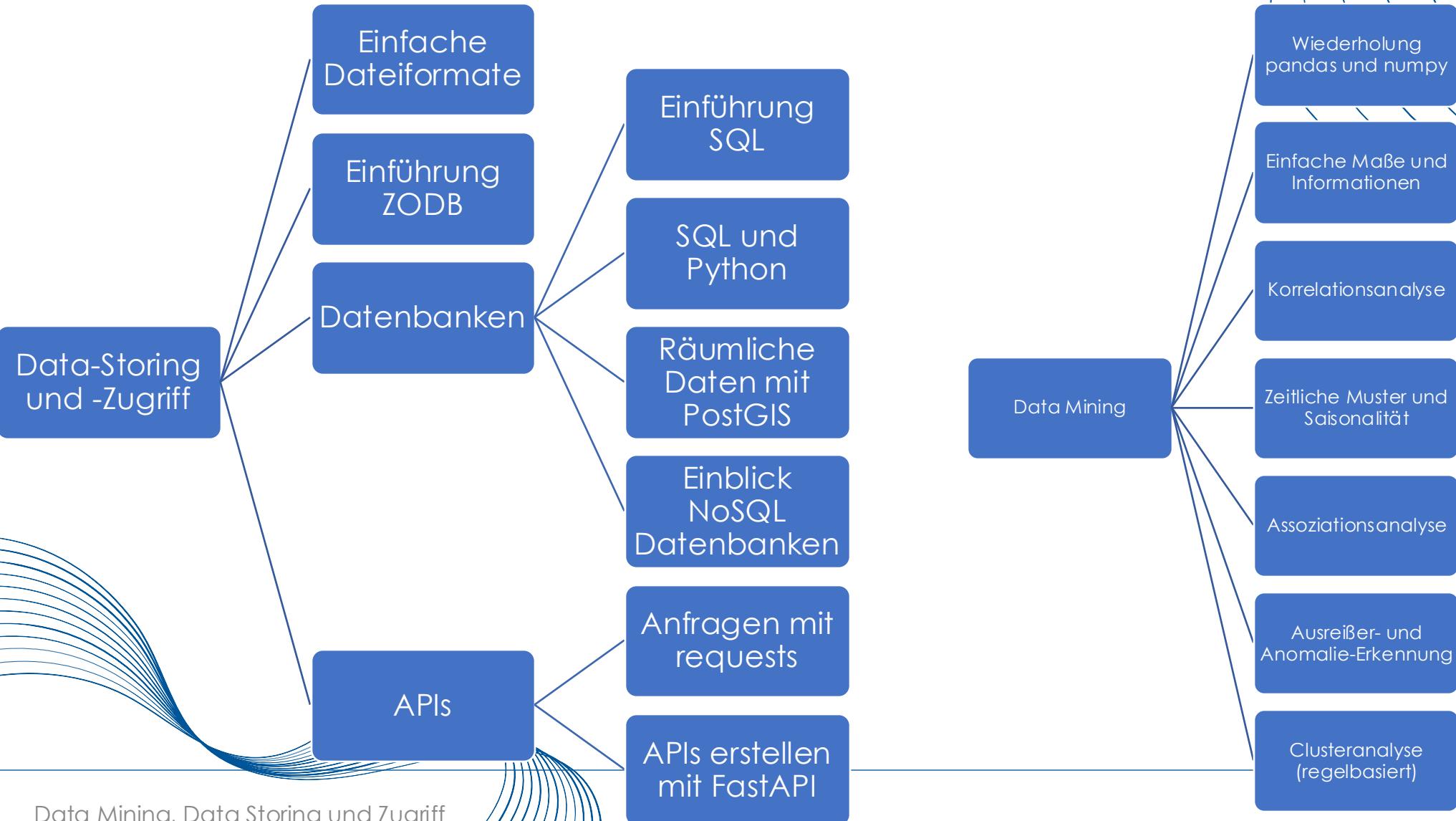
16:30 – 17:30 **OpenSpace**



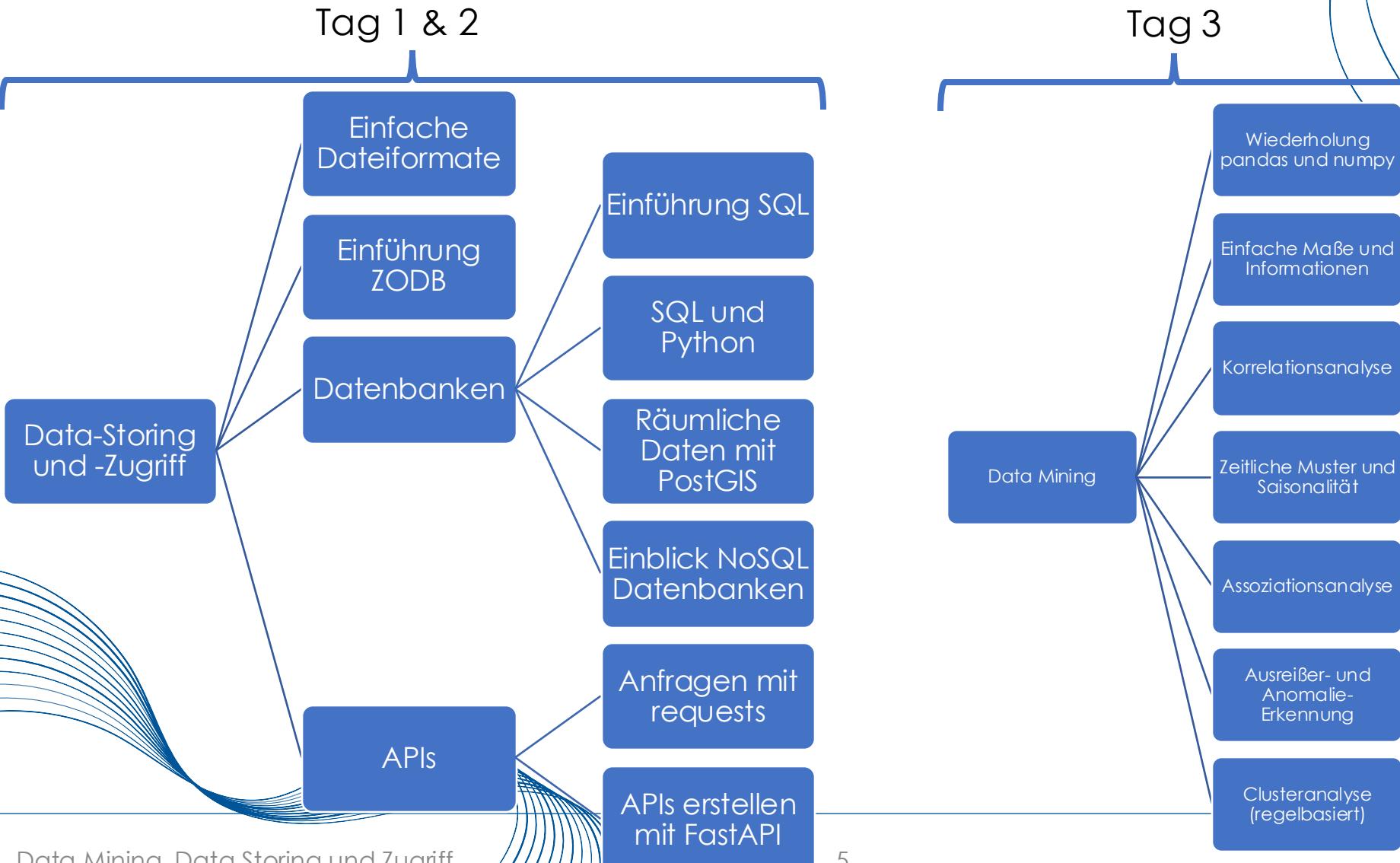
„Vorstellungs“-runde

Erwartungen an die Schulung
Vorkenntnisse mit Datenzugriff

Agenda



Agenda





Materialien im Repository + Einrichtung in Pycharm

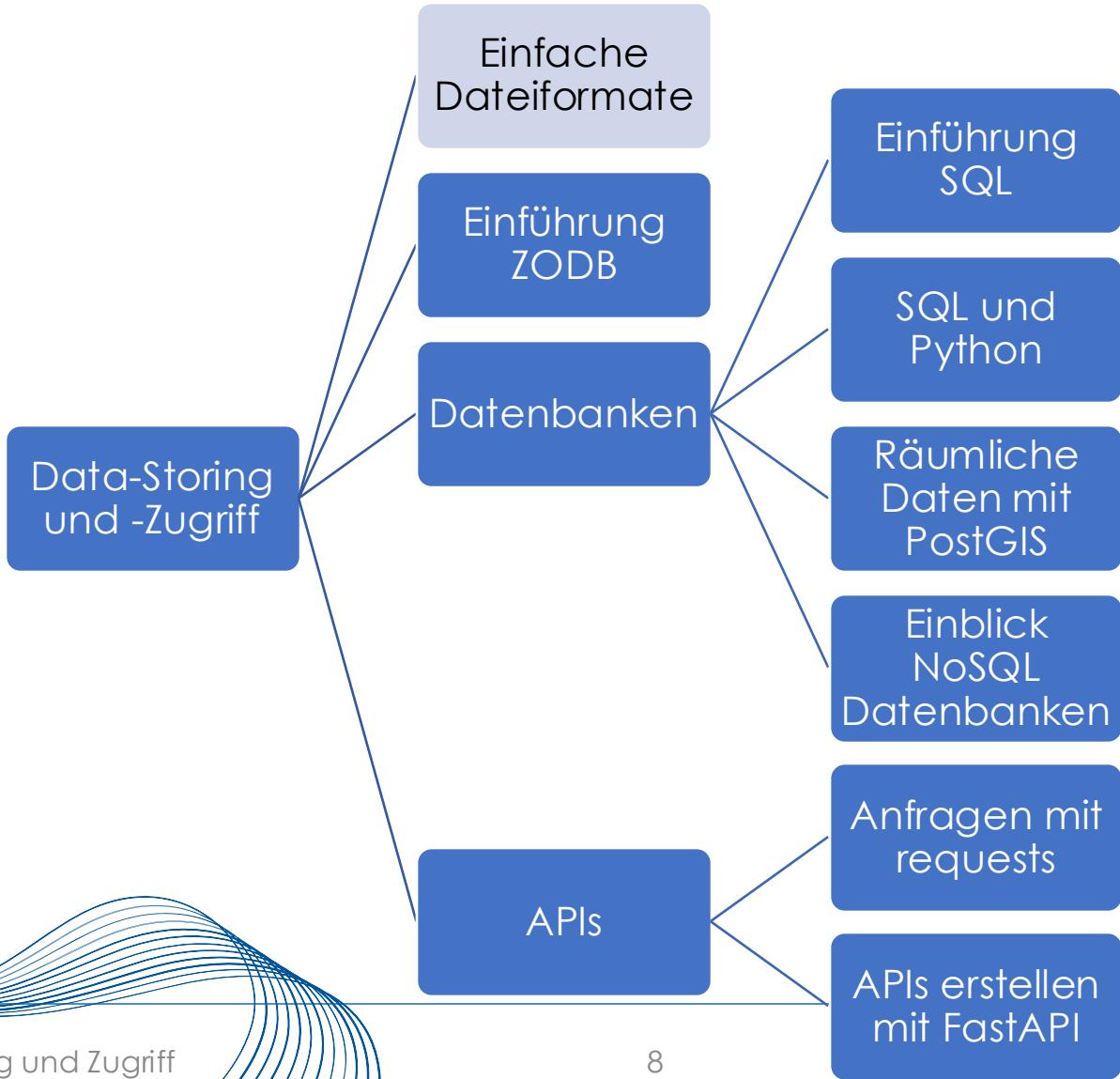
Daten, Code, Glossar

https://github.com/NKDataConv/Schulung_DataStoring

Spielregeln

- Jeder sollte coden
- Spielerisch mit Code umgehen und Dinge ausprobieren

Agenda



Serialisierung – Was ist das?

= Umwandlung eines Objekts in ein Format, das gespeichert oder übertragen werden kann

- **Daten speichern:** Speicherung von Objekten in Datenbanken oder Dateien
- **Datenübertragung:** Senden von Objekten über Netzwerke (z.B. APIs, Remote-Verbindungen)
- Anwendung in **verteilten Systemen:** Kommunikation zwischen Systemen mit unterschiedlichen Architekturen
- **Deserialisieren** = Rückumwandlung in das ursprüngliche Objekt

Comma-Separated Values - CSV

- Textdatei, in der Daten in Zeilen und Spalten strukturiert sind, getrennt durch Kommas (oder andere Trennzeichen)
 - **Leicht lesbar und bearbeitbar:** CSV-Dateien sind einfach zu öffnen und können von Menschen gelesen und editiert werden
 - **Kein komplexes Datenmodell:** Unterstützt nur flache, tabellarische Daten; keine Hierarchien oder Verschachtelungen möglich

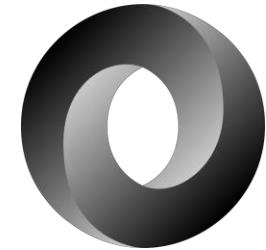
Anwendungsfälle

- Datenexport und –import
 - Einfache Datenanalyse
 - Schnittstelle zwischen Programmen

"ID"	"Manufacturer"	"Country"	"Category"	"Model"	"Year"
1	"Acura"	"Japan"	"Asia"	"RL"	8.5
2	"Acura"	"Japan"	"Asia"	"TL"	39.
3	"Acura"	"Japan"	"Asia"	"Integra"	1994
4	"Acura"	"Japan"	"Asia"	"CL"	14.
5	"Audi"	"Germany"	"Europe"	"A8"	1994

JavaScript Object Notation - JSON

- Textbasiertes Datenformat, das strukturierte Daten in einer menschen- und maschinenlesbaren Form darstellt
- **Menschen- und maschinenlesbar:** Leicht zu verstehen und zu debuggen
- **Flexibel:** Unterstützt verschachtelte Datenstrukturen, Arrays und komplexe Objekte



Anwendungsfälle

- API-Kommunikation
- Konfigurationsdateien
- Datenübertragung zwischen Systemen

```
{  
    "vorname": "Anna",  
    "nachname": "Müller",  
    "alter": 28,  
    "email": "anna.mueller@example.com",  
    "mitglied": true,  
    "interessen": {  
        "farbe": "grün",  
        "essen": "italienisch",  
        "hobbys": ["lesen", "radfahren", "fotografie"]  
    }  
}
```

Protocol Buffers - Protobuf

- Binäres Serialisierungsformat
- **Kompakt und effizient:** Geringerer Speicherbedarf und schnellere Verarbeitung als textbasierte Formate wie JSON oder XML
- **Typensicherheit:** Unterstützt definierte Datentypen und sorgt so für klare Datenstruktur und Konsistenz

Anwendungsfälle

- Echtzeitanwendungen
- Interne Systemkommunikation
- APIs mit hohem Durchsatz

Weitere Serialisierungsformate

Extensible
Markup
Language
(XML)

YAML Ain't
Markup
Language
(YAML)

Avro

Thrift

Code Demo

In /materialien_serialisierung

- `demo_csv.py`
- `demo_json.py`
- `demo_protobuf.py`

Übung

Lese die cars.csv ein und speichere sie als .JSON ab.

Python - intake

Python-Package zur einfachen Handhabung und Verwaltung von Datenquellen.

- Erlaubt das Laden, Katalogisieren und Teilen von Daten.
- Erstellung von Datenkatalogen (Data Catalogs) mit YAML-Dateien.
- Unterstützt verschiedene Datenformate wie CSV, Parquet, SQL, JSON, etc.
- Vereinheitlichte API für den Zugriff auf unterschiedliche Datenquellen.
- Erleichtert das Arbeiten mit großen und komplexen Datensätzen.
- Integrierbar in Data-Science-Workflows und Tools wie pandas und Dask.



INTAKE

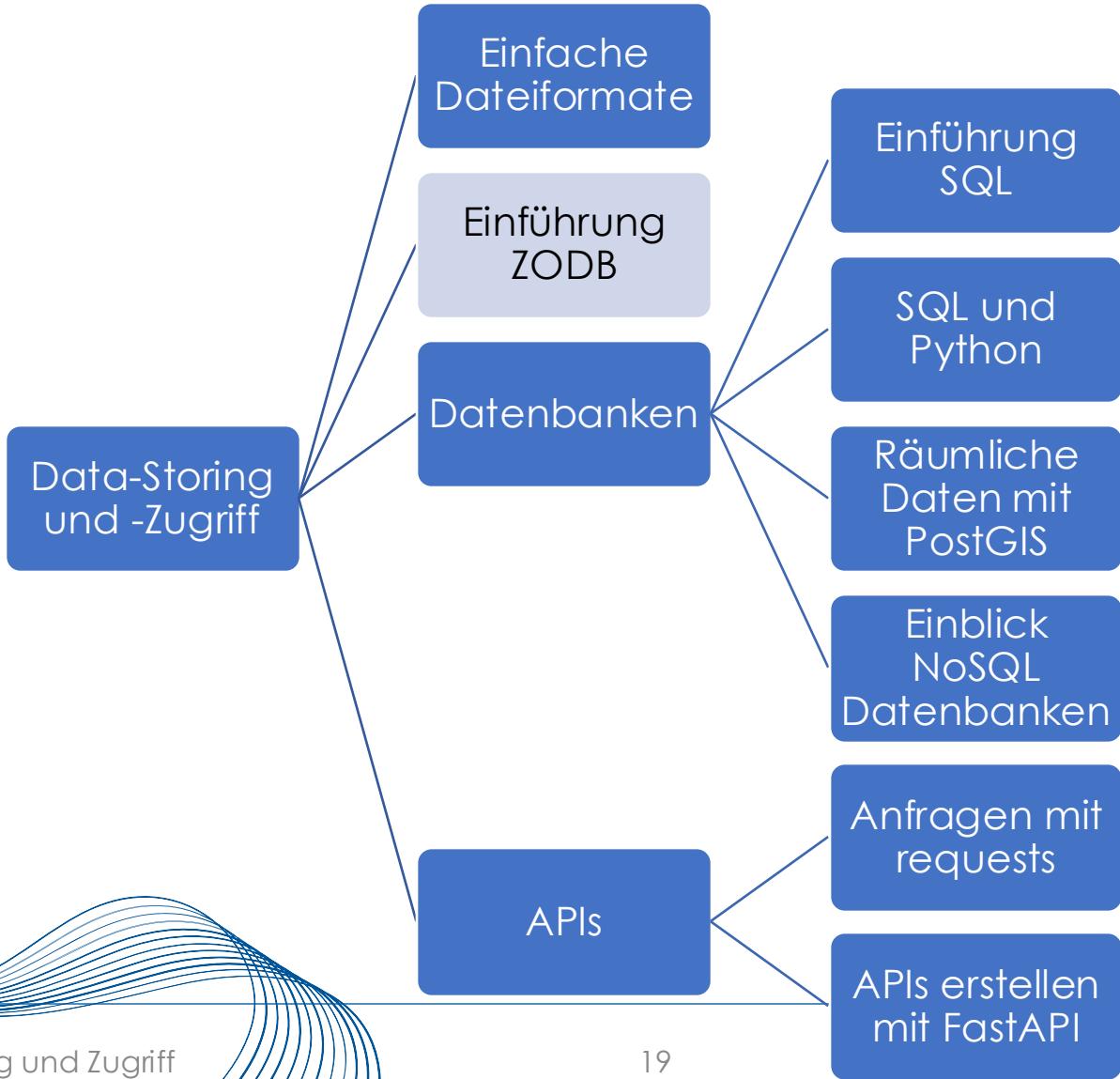
Code Demo

In /materialien_intake/demo.py
Und data_catalog.yml

Übung

Füge die .csv Datei „cars.csv“ als Quelle im Daten Katalog hinzu. Teste das Lesen der Dateien anschließend.

Agenda



ZODB – Zope Object Database

= Eine objektorientierte Datenbank für Python.

- Ermöglicht die direkte Persistenz von Python-Objekten.

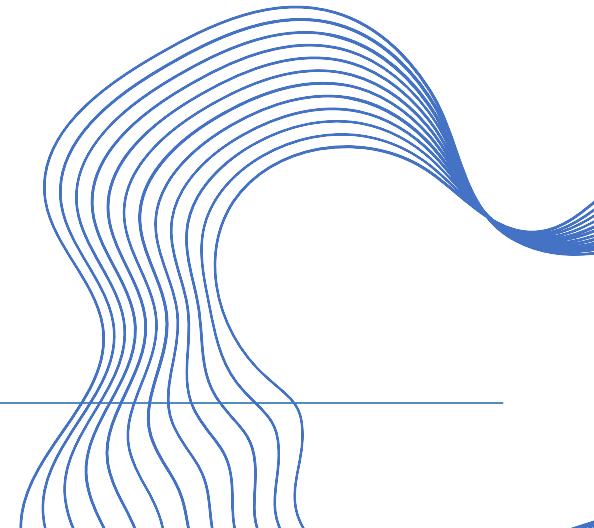


Hauptmerkmale:

- Keine manuelle Übersetzung zwischen Objekt und Datenbank (z. B. ORM).
- Unterstützt Transaktionen und Versionierung.
- Integrierte Datenbank für Hierarchiestrukturen.

Vorteile:

- Nahtlose Integration mit Python.
- Flexibilität durch objektorientiertes Speichern.
- Einfach zu implementieren, ideal für Prototyping.



Code Demo

material_zodb/demo_1.py

Was bedeutet ACID?

- **Atomarität:** Operationen sind unteilbar; entweder vollständig oder gar nicht ausgeführt.
- **Konsistenz:** Datenbank bleibt in einem konsistenten Zustand, bevor und nachdem eine Transaktion abgeschlossen ist.
- **Isolation:** Gleichzeitige Transaktionen beeinflussen sich nicht gegenseitig.
- **Dauerhaftigkeit:** Nach Abschluss einer Transaktion bleiben Änderungen erhalten, auch bei Systemausfällen.

Atomarität

```
BEGIN TRANSACTION;  
UPDATE Konten SET Saldo = Saldo - 100 WHERE KontoNr = 123;  
UPDATE Konten SET Saldo = Saldo + 100 WHERE KontoNr = 456;  
COMMIT;
```

Datenbank garantiert, dass Transaktion komplett oder gar nicht ausgeführt wird.
Bei Fehlschlagen des zweiten Updates, wird auch das erste zurückgerollt.

Konsistenz

- Keine Verletzung von Eindeutigkeits-Bedingungen (Primary Key, Unique)
- Keine Ungültige Fremdschlüssel
- Keine Verletzung von NOT NULL Spalten

Isolation

- Datenkonsistenz trotz Parallelität:
Transaktionen dürfen keine Zwischenzustände anderer Transaktionen sehen
- Keine unzulässigen Überschneidungen:
Änderungen durch eine Transaktion werden erst dann sichtbar, wenn diese abgeschlossen ist (also nach COMMIT).
- Die Eigenschaft geht auf Kosten der Performance



Dauerhaftigkeit

Nicht dauerhaft sind beispielsweise:

- Memory Datenbanken
 - Memcached, Redis
- Streams
 - Kafka
- Transaktionslogs

ACID in ZODB

- ✓ **Atomarität:** Transaktionen sind atomar, Änderungen werden vollständig oder nicht übernommen.
- ✓ **Konsistenz:** Unterstützt durch Python-Objektstrukturen; garantiert Datenintegrität.
- ✓ **Isolation:** Optimistische Sperrmechanismen für parallele Transaktionen.
- ✓ **Dauerhaftigkeit:** Änderungen werden persistent auf Datenspeicher geschrieben.

Übung ZODB

Führe die Datei „material_zodb/demo_2.py“ aus. Was überrascht am Verhalten?

Persistenzregeln in ZODB

Nur persistente Objekte werden gespeichert

- Nur Objekte, die von persistent.Persistent erben, können von der ZODB verwaltet werden.

Explizite Attributänderungen

- Änderungen an Attributen eines persistenten Objekts werden von ZODB erkannt und markiert das Objekt automatisch als "geändert".

Persistente Container: Änderungen an Unterobjekten

- Änderungen an Python-Containern (z. B. dict, list), die innerhalb eines persistenten Objekts gespeichert sind, werden nicht automatisch erkannt. Lösung:
 - `container._p_changed = True # Markiert das Objekt als "geändert".`
 - Alternativ Verwendung spezieller Container wie PersistentMapping (statt dict) und PersistentList (statt list)

Transaktionen und transaction.commit()

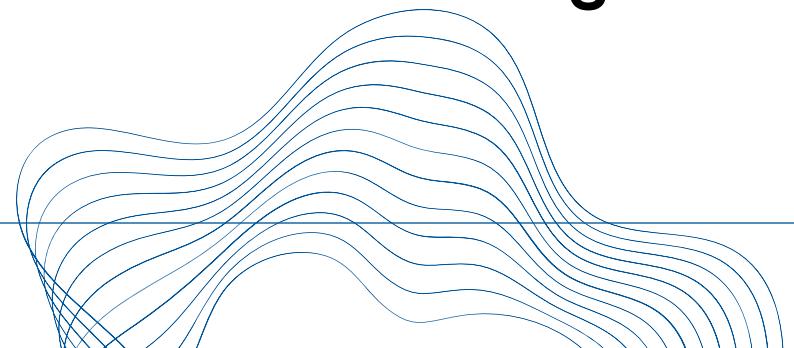
- Änderungen an persistierten Objekten werden erst gespeichert, wenn eine Transaktion explizit mit `transaction.commit()` abgeschlossen wird

Verwendung des OOBTee in ZODB

Anders als ein normales Python-Dictionary, das alle Daten im Speicher hält, speichert OOBTee seine Daten in einer **Baumstruktur**. Dies ermöglicht:

- **Effiziente** Suche, Einfügen und Löschen von Elementen.
- **Speicheroptimierung**, da nur die Teile des Baums im Speicher gehalten werden, die tatsächlich verwendet werden.
- Im Gegensatz zu einem Dictionary garantiert OOBTee, dass die Schlüssel immer **sortiert** bleiben. Das ist hilfreich für Anwendungen, die geordnete Daten erfordern.

„OO“ bedeutet, dass sowohl Schlüssel, als auch Wert ein **beliebiges Python Object** sein können.



Code demo

material_zodb/demo_4.py

Übung

Erstelle eine Klasse „Person“, die von Persistent erbt und mit den Attributen „name“, „adresse“ und „alter“. Speichere 3 Instanzen der Klasse Person mit OOBTree in ZODB.

ZODB – weitere Features

Mounting:

- Ermöglicht das Einhängen mehrerer unabhängiger Datenbanken in eine gemeinsame Objektstruktur.
- Ideal für die logische Trennung von Daten oder modularen Aufbau.

Distributing:

- Ermöglicht verteilte Anwendungen mit mehreren Clients und Skalierbarkeit.
- Die Datenbank kann auf mehreren Servern repliziert werden, um Ausfallsicherheit zu gewährleisten.

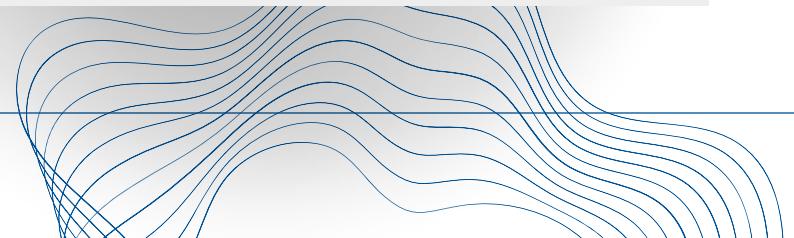
Einsatz von ZODB



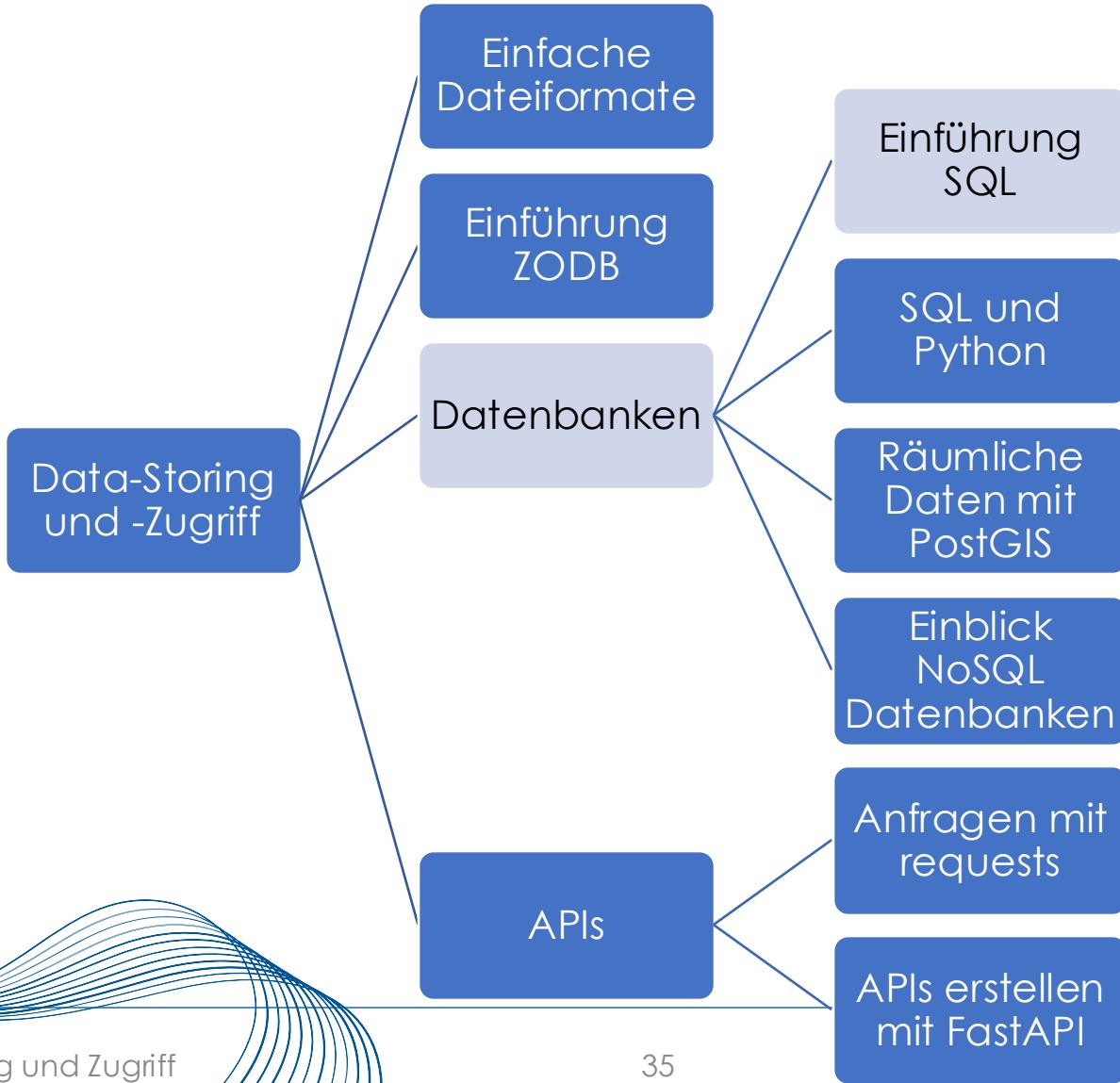
- **Python-basierte Anwendungen**, bei denen die Daten als Objekte verwaltet werden.
- **Anwendungen mit komplexen und verschachtelten Datenstrukturen**, die sich schwer auf relationale Tabellen abbilden lassen.
- **Schnelles Prototyping und nahtlos in Python integrierbar**



- Einsatz **nur in Python**, nicht mit anderen Sprachen kompatibel.
- Große Datenmengen: ZODB **skaliert nicht gut** bei extrem großen Datenmengen.
- **Hochgradig parallele Schreiboperationen**: Wenn viele parallele Schreibzugriffe nötig sind



Agenda



Was ist SQL?

=**S**tructured **Q**uery **L**anguage.

- Dient der Verwaltung und Abfrage von relationalen Datenbanken.
- Ermöglicht das Erstellen, Lesen, Aktualisieren und Löschen von Daten (CRUD).
- Unterstützt komplexe Operationen wie Joins, Aggregationen und Datenfilterung.
- Ein zentraler Standard für Datenmanagement in IT-Systemen.

Relationale Datenbank Management Systeme (RDBMS)

Relationale Datenbank:

- Speichert Daten in Tabellen mit Zeilen (Datensätze) und Spalten (Attribute).
- Daten sind ggf. durch Beziehungen (=Relationen) zwischen Tabellen miteinander verknüpft.
- Geeignet für strukturierte Daten.

RDBMS (Relational Database Management System) = **Software**, die die Verwaltung relationaler Datenbanken ermöglicht.

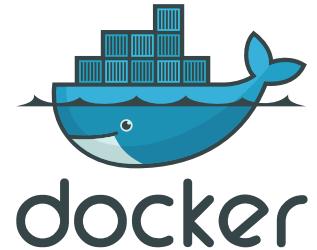
- Führt Operationen wie Speichern, Abrufen und Manipulieren von Daten aus.
- Unterstützt Funktionen wie Datenintegrität, Zugriffskontrolle und Transaktionen.

SQL Datenbank Technologien



Intermezzo Docker

Welches Problem löst Docker?



Herausforderung der Softwareentwicklung

- „Es funktioniert auf meinem Rechner“: Unterschiede in Umgebungen führen zu Problemen.
- Schwierige Reproduktion von Abhängigkeiten und Konfigurationen.

Was Docker bietet

- Standardisierte Container für Anwendungen und deren Abhängigkeiten.
- Konsistente Ausführung in jeder Umgebung: Lokal, Testing, Produktion.
- Isolierte und portable Entwicklungsumgebungen.

Wichtige Vorteile

- Schnellere Bereitstellung von Anwendungen.
- Effiziente Ressourcennutzung.
- Erleichterung der Zusammenarbeit zwischen Teams.

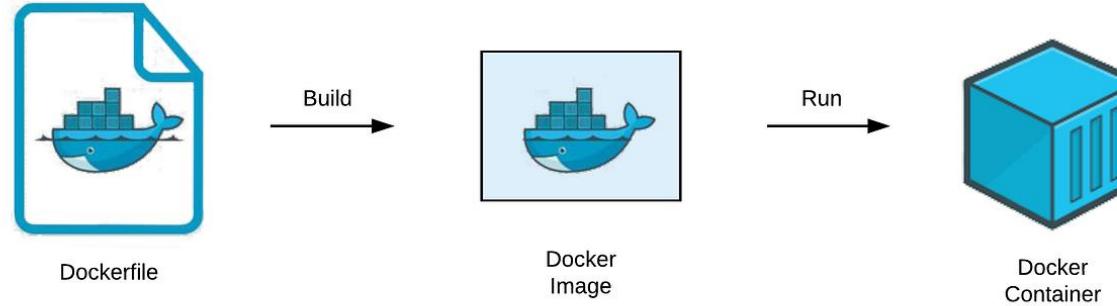
Docker – die wichtigsten Begriffe

Dockerfile: Eine Textdatei, die Bauanleitungen für ein Docker-Image enthält.

Image: Eine Vorlage, die alle notwendigen Dateien, Abhängigkeiten und Befehle enthält.

Container: Ausführbare Instanz eines Docker-Images, isoliert und portabel.

Registry: Ein Repository, um Docker-Images zu speichern und zu teilen (z. B. Docker Hub).



<https://medium.com/swlh/understand-dockerfile-dd11746ed183>

Docker Referenz: die wichtigsten Befehle

- Erstellt ein Docker-Image aus einem Dockerfile:

```
docker build [OPTIONS] PATH
```

- Startet einen neuen Container basierend auf einem Docker-Image:

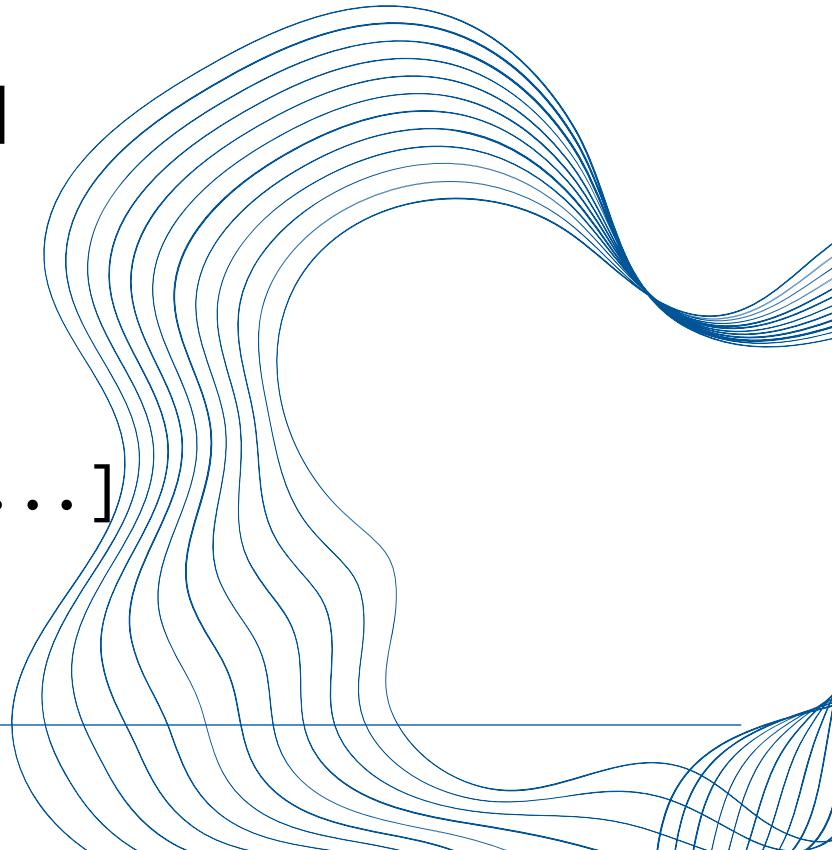
```
docker run [OPTIONS] IMAGE [COMMAND] [ARGS]
```

- Zeigt laufende Container an:

```
docker ps [OPTIONS]
```

- Beendet einen laufenden Container:

```
docker stop [OPTIONS] CONTAINER [CONTAINER...]
```



Demo PostgreSQL mit Docker starten

- PostgreSQL Container starten:
`material_postgresql/start_postgresql.txt`

SQL – Tabellen erstellen und Daten einfügen

Tabellen erstellen:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype
);
```

datatype gibt den Datentyp (z. B. INT, VARCHAR, DATE) an.

Daten einfügen:

```
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

SQL – einfache Select Anweisungen

- Grundlegende Syntax:

```
SELECT column_1, column_2 FROM table_name;
```

- Alle Spalten auswählen:

```
SELECT * FROM table_name;
```

- Alle Spalten, limitierte Anzahl Zeilen auswählen:

```
SELECT * FROM table_name LIMIT 10;
```

Demo in PostgreSQL

In materialien_postgresql/create_table.txt

Übung

Erstelle eine Tabelle user_data mit Spalten „name“ und „alter“ und füge 2 Beispiel-Datenpunkte ein.

Demo PostgreSQL pgadmin starten

In material_postgresql/pgadmin_start.txt

SQL – Primär- und Fremdschlüssel



Primärschlüssel (Primary Key):

- Eindeutige Kennung für jede Zeile in einer Tabelle.
- Besteht aus einer oder mehreren Spalten.
- Keine doppelten oder NULL-Werte erlaubt.
- Beispiel: ID in einer Kundentabelle.



Fremdschlüssel (Foreign Key):

- Verweist auf den Primärschlüssel einer anderen Tabelle.
- Stellt Beziehungen zwischen Tabellen her.
- Beispiel: Kunden_ID in einer Bestelltablette.

Zweck:

- Datenintegrität und Vermeidung von Inkonsistenzen.
- Erleichtert komplexe Abfragen und Verknüpfungen.

SQL - Indices



Was ist ein Index?

- Sortierung der Tabelle anhand eines Wertes, sodass schneller gesucht werden kann
- Aufbau einer Separaten Struktur mit Verweisen auf die entsprechenden Speicherorte

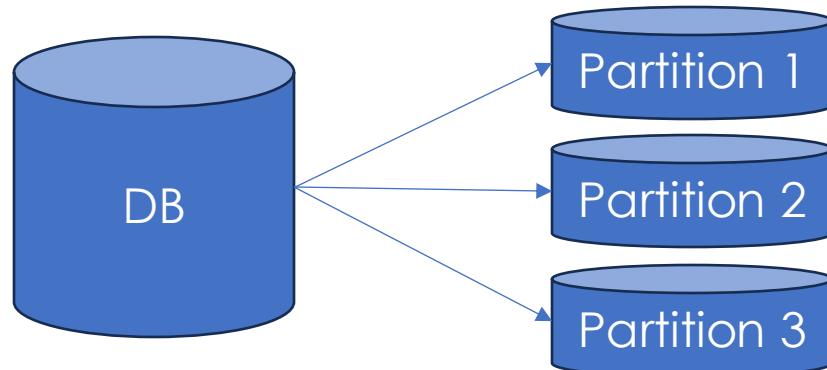


Nachteil von Indices: Schreibgeschwindigkeit in Tabelle ist geringer, da zusätzlich der Index geschrieben werden muss.

Datenbank – Skalierung mit Partitionen

Partitionierung ist das **Aufteilen** einer großen Tabelle in kleinere, handhabbare Teile, die „Partitionen“ genannt werden. Jede Partition speichert einen Teil der Daten und kann unabhängig behandelt und abgefragt werden.

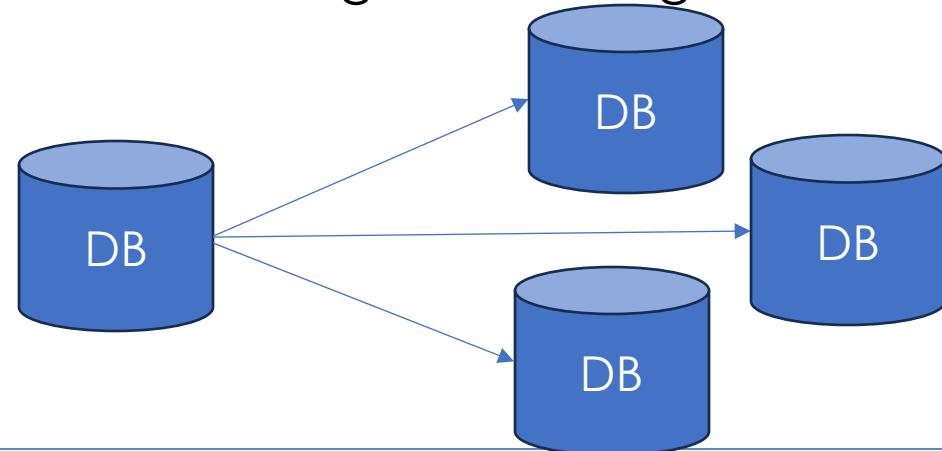
- **Schnellere Abfragen:** Nur die relevanten Partitionen werden bei einer Abfrage durchsucht, was die Performance verbessert.
- **Einfache Wartung:** Partitionen können einzeln archiviert, gelöscht oder optimiert werden, ohne die gesamte Tabelle zu beeinflussen.
- **Parallelisierung:** Abfragen können parallel über mehrere Partitionen hinweg erfolgen, wodurch die Verarbeitungsgeschwindigkeit erhöht wird.



Datenbank – Skalierung mit Replikationen

Replikation kopiert Datenbanken oder Tabellen auf mehrere Server, was die Skalierbarkeit und Ausfallsicherheit erhöht. PostgreSQL bietet verschiedene Replikationsarten:

- **Lastverteilung:** Leseoperationen können auf mehrere Standby-Server verteilt werden, wodurch die Last auf dem Master-Server sinkt.
- **Ausfallsicherheit:** Bei einem Ausfall des Master-Servers kann ein Standby-Server sofort übernehmen, was die Verfügbarkeit erhöht.
- **Geografische Verteilung:** Replikation ermöglicht verteilte Systeme über verschiedene Standorte hinweg, um die Zugriffszeiten für lokale Benutzer zu reduzieren.



Datenbank – Backups und Wiederherstellung

Base Backups

- Komplett-Backups der Datenbank, die als Ausgangspunkt für die Wiederherstellung dienen. Diese können regelmäßig durchgeführt und bei Bedarf schnell wieder eingespielt werden.

WAL-Archiving

- Die Write-Ahead Logs (WALs) werden kontinuierlich gespeichert und können bei einem Ausfall genutzt werden, um eine Datenbank auf den neuesten Stand zu bringen. Dies ist essenziell für das PITR.

Point-in-Time Recovery (PITR)

- Ermöglicht es, eine Datenbank zu einem bestimmten Zeitpunkt wiederherzustellen. Dabei werden regelmäßige Backups mit sogenannten Write-Ahead Logs (WAL) kombiniert, um alle Transaktionen bis zum gewünschten Zeitpunkt wiederherzustellen.

Datenbank – Backups und Wiederherstellung

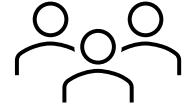
pg_dump und pg_restore

- Tools, die es ermöglichen, Daten auf Tabellenebene zu sichern und wiederherzustellen. Sie bieten Flexibilität für den Export und Import bestimmter Datensätze oder Datenbankteile.

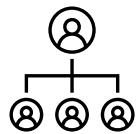
Replikation und Failover

- Bei der Verwendung von Replikation können Ausfallzeiten minimiert werden. Ein Standby-Server kann im Fehlerfall zum Hauptserver umschalten, was eine nahtlose Wiederherstellung ermöglicht.

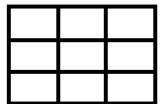
Datenbank – Nutzerverwaltung und Rechtemanagement



- **Benutzerrollen** definieren: PostgreSQL unterstützt vordefinierte Rollen (z. B. SUPERUSER, READ ONLY) und benutzerdefinierte Rollen.
- Rechteverwaltung: **Präzise Steuerung** von Zugriffsrechten über GRANT und REVOKE.



- **Rollenhierarchie**: Rollen können andere Rollen beinhalten, um Berechtigungen zu vererben.
- Sicherheit durch **Passwort und Authentifizierung**: Unterstützung verschiedener Authentifizierungsmethoden wie md5, scram-sha-256.

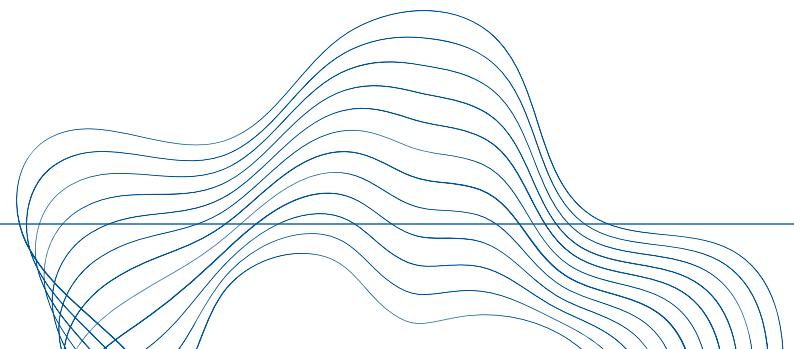


- Schema- und Tabellenzugriff: Zugriffsrechte können **auf Schema-, Tabellen- und Spaltenebene** festgelegt werden.
- Best Practices: Verwenden von rollenbasiertem Zugriff anstelle direkter Benutzerrechte.

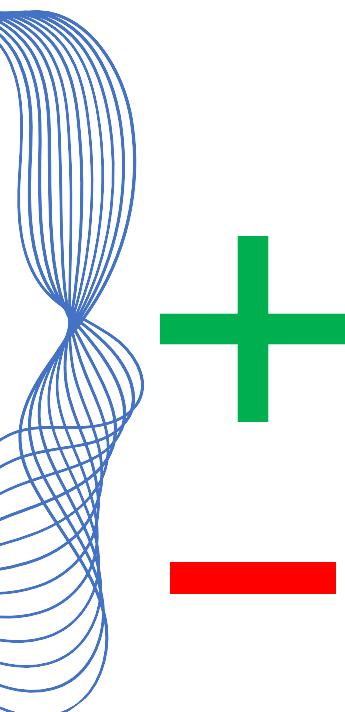
Datenbank - Verschlüsselung

- Transportverschlüsselung (TLS/SSL): Sicherung der Datenübertragung zwischen Client und Server.
- Verschlüsselung der Daten auf Festplattenebene, um vor physischem Diebstahl zu schützen.
- Spaltenbasierte Verschlüsselung: Verschlüsselung sensibler Daten auf Anwendungsebene.

```
SELECT pgp_sym_encrypt('Sensitive Data',  
'encryption_key');
```



(Postgre)SQL vs. ZODB



SQL-Datenbanken

Anwendung bei
Strukturierten Daten

- weit verbreitet
 - Komplexe Abfragen möglich
- Unterstützung für Python Objekte schwer

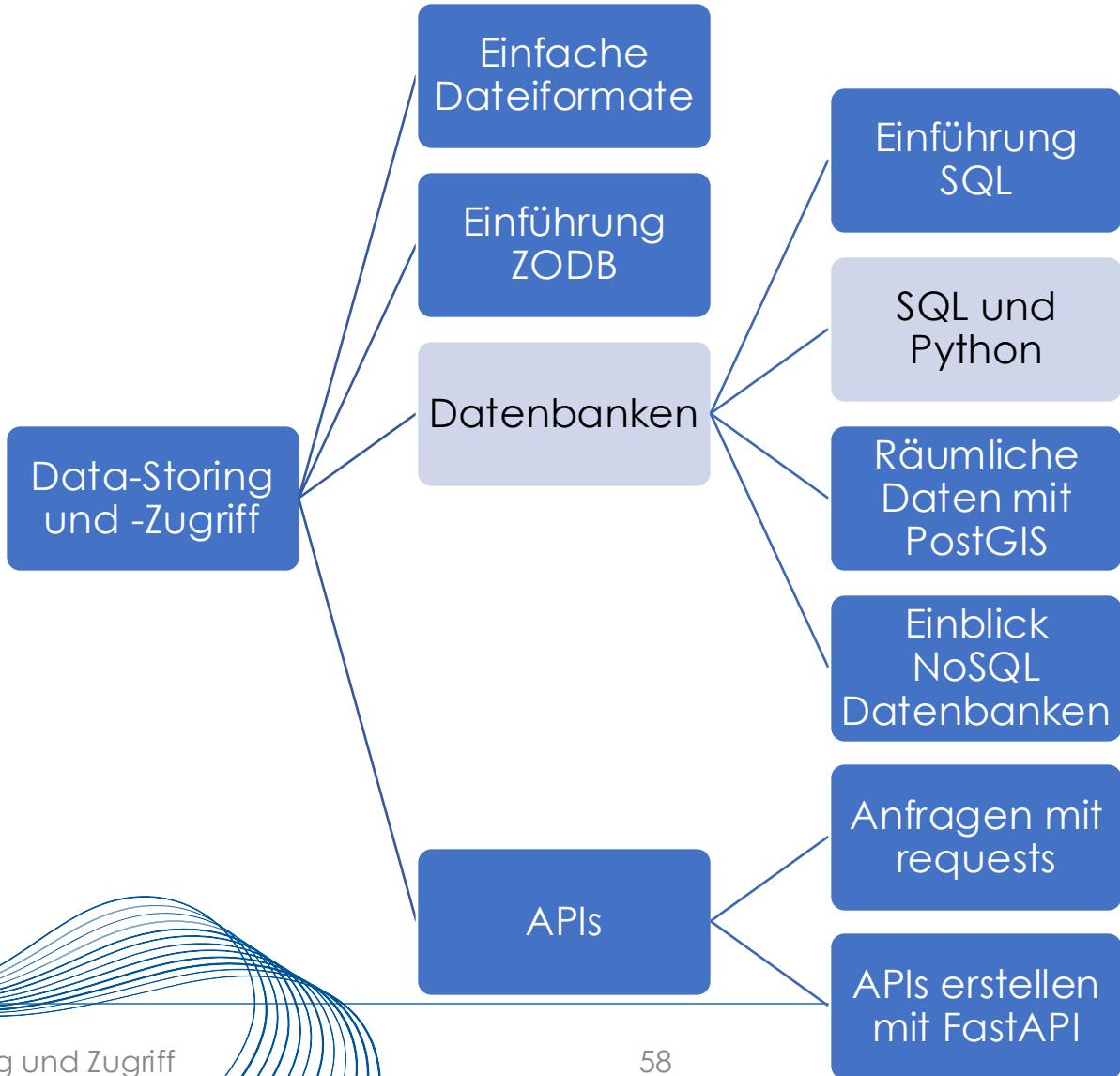
ZODB

Anwendung bei Python Objekten

- einfach in der Einrichtung
- Schnelles Prototyping
- Nur mit Python gut verwendbar



Agenda





SQLite

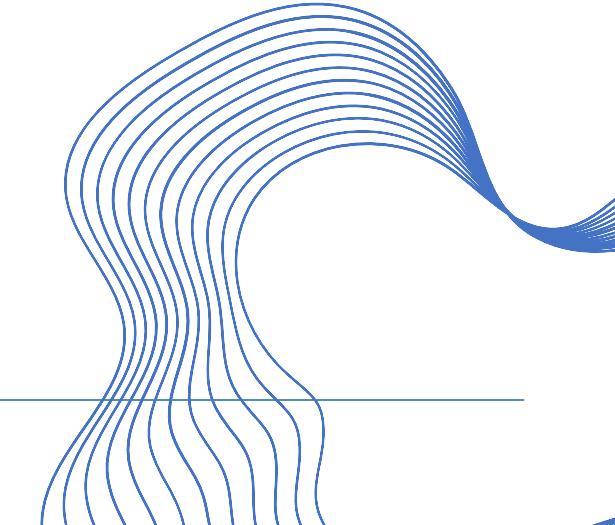
= Integrierte, serverlose, dateibasierte SQL-Datenbank.

Vorteile von SQLite:

- Plattformunabhängig und kein Setup erforderlich.
- Daten werden in einer einzigen Datei gespeichert.
- Perfekt für lokale Datenspeicherung und Tests.

Einsatzbeispiele:

- Prototyping von Anwendungen.
- Temporäre oder persistente Datenspeicherung.
- Analyse und Data Mining auf lokalen Daten.



Python DB API

= Standard für Zugriff auf relationale Datenbanken
(PEP 249: <https://peps.python.org/pep-0249/>)



Die wichtigsten Merkmale:

- **Verbindung zur Datenbank herstellen** mit `connect()`
- **Cursor-Objekt:** ermöglicht die Ausführung von SQL-Befehlen und das Abrufen von Ergebnissen
- Mit Methoden wie `fetchone()`, `fetchmany()` oder `fetchall()` **Ergebnisse von SQL-Abfragen abrufen**
- **Änderungen speichern** mit `commit()`
- Verbindung zur **Datenbank schließen** mit `close()`

Demo SQLite

Tabelle erstellen und Zugriff mit pandas und SQLAlchemy

- material_sqlite/sqlite_demo.py
- material_sqlite/demo_sqlite_load_table.py

Übung SQLite

Lade die Daten daten/weatherAUS.csv in eine SQLite Tabelle „weather“. Rufe anschließend aus der Datenbank-Tabelle alle Daten mit „Location = Sydney“ ab.

Daten im intake catalog ergänzen

Gemeinsam: PostgreSQL

Übung: SQLite Daten BTC_daily

(Hinweis: uri: "sqlite:///example.db")

- SQLAlchemy ist eine Open-Source-Bibliothek in Python für den Datenbankzugriff.
- Sie bietet sowohl eine ORM-Schicht als auch eine direkte SQL-Schnittstelle.

Hauptfunktionen:

- Erstellen und Verwalten von Datenbankverbindungen.
- Ausführen von SQL-Befehlen und Transaktionen.
- Mapping von Python-Klassen auf Datenbanktabellen (ORM).
- Unterstützung komplexer Datenbankabfragen und Beziehungen.

Was ist ORM?

Object-Relational Mapping (ORM) verbindet Objekte in einer Programmiersprache mit Datenbanktabellen

Funktionalität:

- Erzeugt automatisch Datenbankabfragen aus Python-Klassen.
- Erleichtert das Arbeiten mit Datenbanken ohne direktes SQL.

Vorteile:

- Reduziert Boilerplate-Code.
- Macht den Code lesbarer und wartbarer.
- Unterstützt verschiedene Datenbanken (z. B. MySQL, PostgreSQL, SQLite).

Demo SQLAlchemy

In material_sqlalchemy:

demo_tabellen_erstellen.py

demo_daten_filtern.py

demo_daten_sortieren.py

demo_aggregieren.py

demo_join.py

Übung SQLAlchemy

1. Erstelle ein SQLAlchemy Model für die Daten aus daten/user_data.csv.
2. Lade die Daten in eine PostgreSQL Tabelle „user_data“.
3. Erstelle darauf aufbauen folgende Queries:
 - a. Gebe alle User aus.
 - b. Welcher User hat am „1957-07-13“ Geburtstag?
 - c. Wie viele Nutzer sind älter als „1970-01-01“
 - d. Wer ist die älteste Person?
- e. Bonus: Gebe für jedes Geburtsjahr die Anzahl an Personen aus.

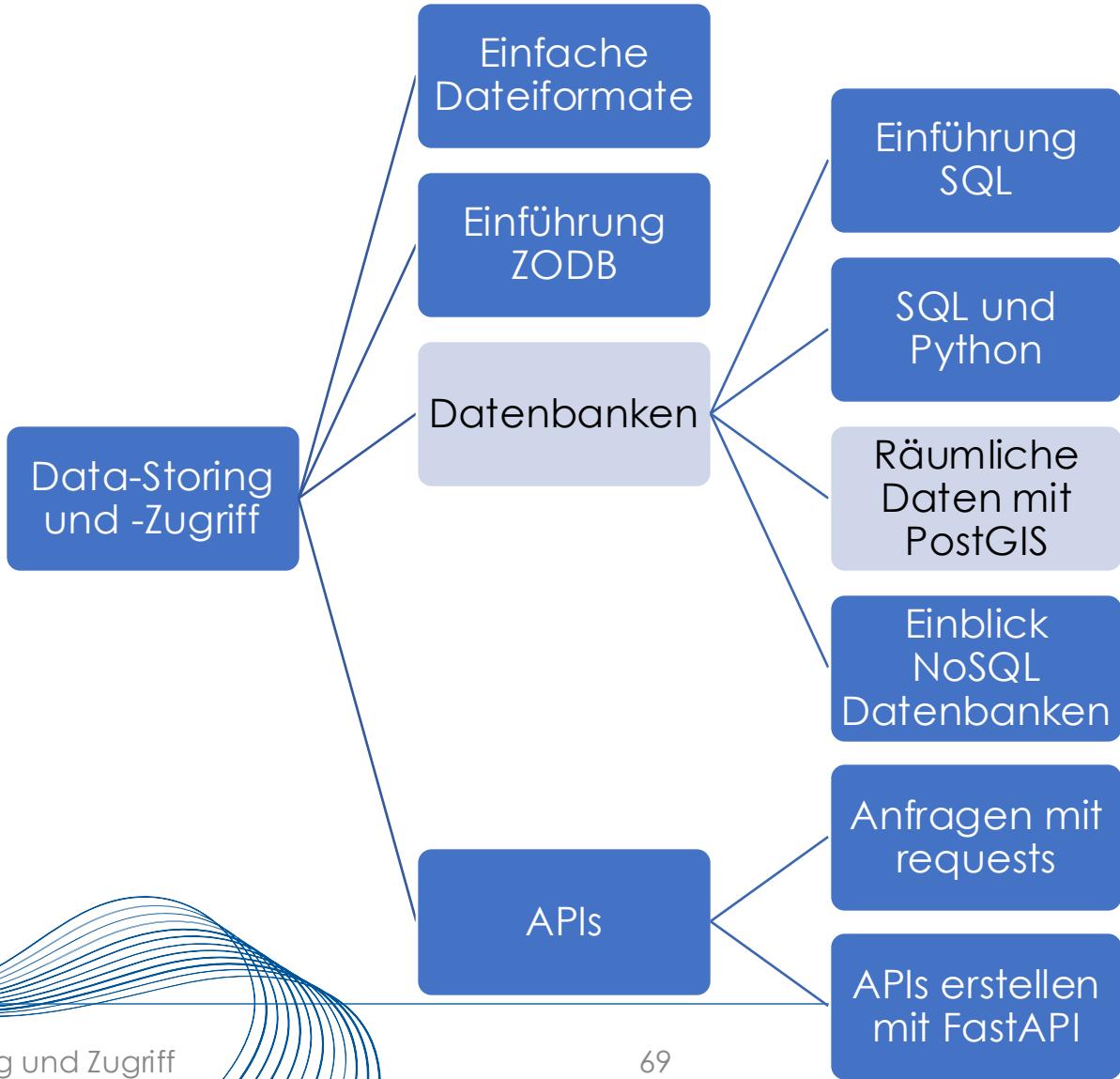
ODBC

= **Open Database Connectivity**: Ein offener Standard für den Zugriff auf verschiedene Datenbanken.

- **Datenbankunabhängigkeit**: Anwendungen können auf unterschiedliche Datenbanken zugreifen, ohne Änderungen am Code.
- **Einheitliche Schnittstelle**: Standardisierte API für die Kommunikation zwischen Anwendungen und Datenbanken.
- **Plattformunabhängigkeit**: Funktioniert auf verschiedenen Betriebssystemen (Windows, Linux, macOS).
- **Breite Unterstützung**: Kompatibel mit nahezu allen großen Datenbankmanagementsystemen.



Agenda



PostGIS

- Erweiterung für PostgreSQL zur Unterstützung geographischer Objekte
- Ermöglicht das Speichern, Abfragen und Bearbeiten von Geodaten in einer Datenbank
- Open-Source und weit verbreitet in geografisches Informationssystem (GIS)- und Spatial-Data-Anwendungen



Eigenschaften von PostGIS

- **Geodaten-Typen:** Punkt, Linie, Polygon, Multi-Polygon und mehr
- **Räumliche Abfragen:** Nahegelegene Objekte finden, Schnittmengen berechnen, Distanzen messen
- **Koordinatensysteme:** Unterstützung für Projektionen und Transformationen zwischen Systemen
- **Skalierbarkeit:** Performante Speicherung und Verarbeitung großer Mengen räumlicher Daten
- **Integration:** Kompatibel mit anderen GIS-Tools und -Bibliotheken (z. B. QGIS, GeoServer)

PostGIS - Datentypen

Geometry

- Flaches Koordinatensystem (Kartesisch)
- Distanzen in Einheiten der Projektion (z. B. Meter bei UTM)
- Flexibel in der Wahl der Projektionen
- **Optimal für lokale Anwendungen**, wo Erdkrümmung vernachlässigt werden kann
- Schneller bei Berechnungen

Geography

- Kugelförmiges Koordinatensystem (Erdkrümmung wird berücksichtigt)
- Distanzen standardmäßig in Metern (WGS 84 Koordinatensystem)
- **Ideal für globale Anwendungen** und lange Distanzen
- Langsamer bei komplexen Berechnungen, aber genaue Ergebnisse über große Entfernungen

SRID - Spatial Reference System Identifier

= identifiziert eindeutig ein Koordinatensystem.

Funktion:

- Sichert die korrekte Interpretation und Kombination geografischer Daten.

Wichtige Eigenschaften:

- **Koordinatensystem:** Geografisch (z. B. Breitengrad/Längengrad) oder projiziert (z. B. Meter).
- **Einheit:** Abhängig vom Koordinatensystem (Grad, Meter).

Bekannteste SRID:

- **SRID 4326:** WGS84, nutzt Längen- und Breitengrade in Grad.
- **SRID 3857:** Pseudo-Mercator, geeignet für Webkarten (Maßeinheit: Meter).

Wichtige Abfragen in PostGIS:

- **SRID einer Geometrie:** ST_SRID(geom)
- **SRID setzen:** ST_SetSRID(geom, 4326)
- **Transformation:** ST_Transform(geom, neues_SRID)

Demo PostGIS

In /material_postgis

- Starte PostGIS mit start_postgis.txt
- Beispiel Queries in postgis_beispiel_queries.txt

Übung PostGIS

In /material_postgis

- Was berechnen die Queries in postgis_uebung_queries.txt?
- Bonus 1: Berechne die Fläche aller Länder, die an Deutschland grenzen.
- Bonus 2: Welches Land ist am weitesten von Deutschland entfernt?

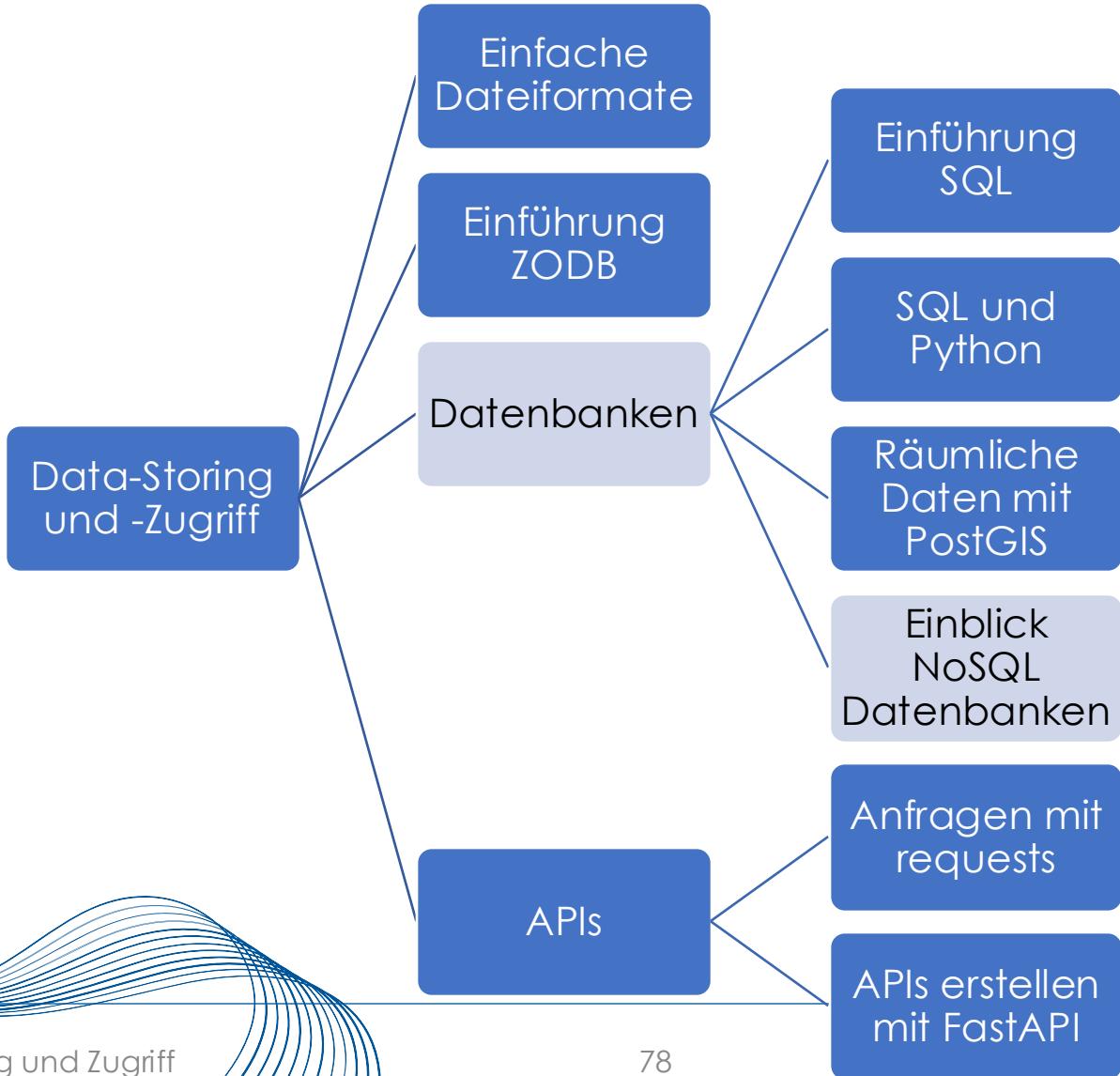
Demo PostGIS in Python

/material_postgis/demo_postgis_python.py

PostGIS – grundlegende Funktionen

- **ST_GeomFromText()**: Wandelt Text in eine Geometrie um
- **ST_Contains() und ST_Intersects()**: Prüfen, ob Geometrien sich schneiden oder enthalten sind
- **ST_Distance()**: Berechnet die Entfernung zwischen zwei Punkten oder Objekten
- **ST_Buffer()**: Erzeugt einen Puffer um ein Objekt
- **ST_Transform()**: Transformiert Geometrien in verschiedene Koordinatensysteme

Agenda



Was bedeutet NoSQL?

- = Not Only SQL – Datenbanken, die über das traditionelle relationale Modell hinausgehen.
- Entwickelt für flexible, skalierbare und schnelle Datenspeicherung und -abfragen.

Charakteristika von NoSQL-Datenbanken

- **Schemafrei:** Keine festen Tabellenstrukturen erforderlich, ideal für unstrukturierte oder semi-strukturierte Daten.
- **Hohe Skalierbarkeit:** Unterstützt horizontale Skalierung durch verteilte Systeme.
- **Flexibilität:** Verschiedene Datenmodelle (z. B. Dokumente, Key-Value, Spalten, Graphen).
- **Optimiert für Performance:** Schnelle Lese- und Schreiboperationen, oft in Echtzeit.



NoSQL DB Technologien



elasticsearch



CouchDB

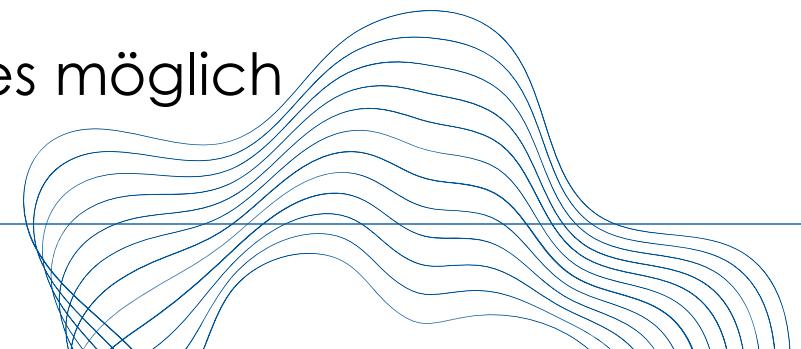
Kurzeinführung MongoDB

= **Open-Source, dokumentenorientierte NoSQL-Datenbank**

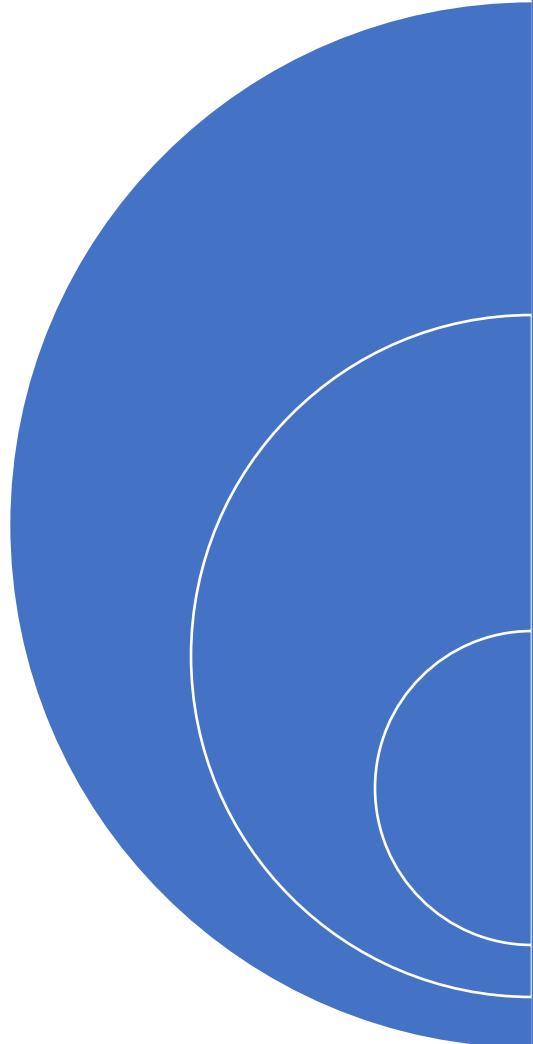
- Speichert Daten in **JSON**-ähnlichen BSON-Dokumenten
- Designed für flexible Schema und hohe Skalierbarkeit

Eigenschaften:

- **Schemafrei:** Anpassbar an sich ändernde Datenstrukturen
- **Hohe Performance:** Optimiert für große Datenmengen und schnelle Abfragen
- **Horizontale Skalierung:** Unterstützt Sharding (Datenverteilung über mehrere Server)
- Reichhaltige **Abfragefunktionen:** Unterstützt Filter, Aggregationen und Geodatenabfragen
- **Indexierung:** Volltext-, Feld- und Geodaten-Indizes möglich



MongoDB – die wichtigsten Begriffe



Datenbank

- Container für Sammlungen
- Mehrere Datenbanken können in einer MongoDB-Instanz existieren

Sammlung (Collection)

- Äquivalent zu einer Tabelle in relationalen Datenbanken
- Enthält Dokumente mit ähnlicher Struktur

Dokument

- Grundlegende Dateneinheit in MongoDB
- JSON-ähnliches Format (BSON)
`{"name": "Max", "alter": 30, "beruf": "Entwickler"}`

ACID in MongoDB



MongoDB unterstützt **ACID**-Eigenschaften, aber es gibt Szenarien und Einstellungen, in denen diese Unterstützung eingeschränkt ist oder besondere Bedingungen gelten.

- Verletzung von Durability bei Einstellung “Write Concern w: 1“, mehreren Replikaten und Absturz des Primär-Nodes.
- MongoDB garantiert standardmäßig keine strikte Isolation für parallele Operationen.

Demo MongoDB

In material_mongo/

- Mongo in Docker starten: start_mongo.txt
 - demo_mongo_1.py
 - demo_mongo_2.py

Übung MongoDB

- Was machen die Anfragen in `demo_mongo_3.py`?
- Bonus 1: Zähle die Einträge in der collection „crew“.
- Bonus 2: Zähle die Anzahl der Filme im genre Drama mit `isAdult = 0`.

SQL vs. NoSQL

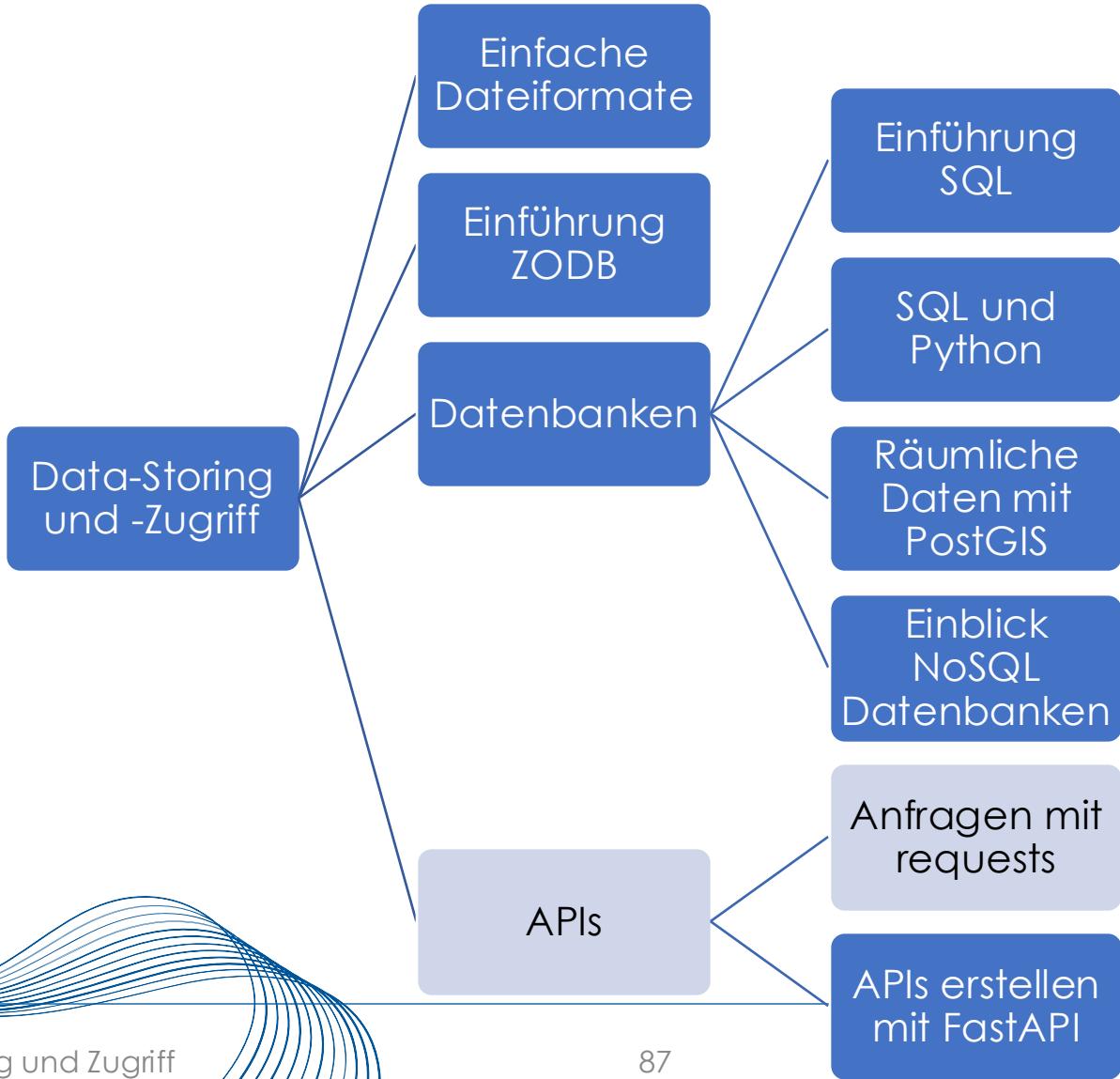
SQL-Datenbanken

- Strukturierte und damit besser zugängliche Daten
- Starres Schema

NoSQL Datenbanken

- Kein Schema notwendig
- Größere Flexibilität und schneller anpassbar
- Freiheit beim Schema kann zu hoher Komplexität führen

Agenda



Application Programming Interface (API)

= eine Schnittstelle zur Interaktion zwischen verschiedenen Softwareanwendungen.

- **Funktion:** APIs ermöglichen den Zugriff auf Daten und Funktionen eines anderen Systems, ohne die zugrunde liegende Logik offenzulegen.
- **Kommunikation:** Durch APIs können Anwendungen über festgelegte „Anfragen“ und „Antworten“ miteinander kommunizieren.

RESTful API

- REST = Representational State Transfer
- **Ressourcenorientierte Architektur:** Jede Ressource (z. B. ein Benutzer, ein Artikel) wird durch eine eindeutige URL adressiert.
- **HTTP-Methoden:** RESTful APIs nutzen die HTTP-Methoden zur Interaktion mit Ressourcen:
- **Repräsentationen:** Ressourcen können in verschiedenen Formaten dargestellt werden, z. B. JSON, XML oder HTML.
- **Zustandslosigkeit:** Jede Anfrage an die API enthält alle notwendigen Informationen, sodass der Server keinen Zustand über verschiedene Anfragen hinweg speichern muss.

Die 4 Basis Methoden CRUD

- C – Create mit POST
 - Anfrage zum Senden von Daten an den Server.
 - Beispiel: Hochladen eines Formulars oder Übermitteln neuer Datensätze.
- R – Read mit GET
 - Anfrage zum Abrufen von Daten vom Server.
 - Beispiel: Abrufen von Benutzerinformationen oder öffentlichen Daten.
- U – Update mit PUT
 - Anfrage zum Aktualisieren von bestehenden Daten auf dem Server.
 - Beispiel: Aktualisieren von Profilinformationen.
- D - DELETE
 - Anfrage zum Löschen von Daten auf dem Server.
 - Beispiel: Löschen eines Benutzerkontos oder eines Datensatzes.

Datenübermittlung in RESTful-APIs

Abfrageparameter (Query Parameters)

- Schlüssel-Wert-Paare, die an die URL einer Anfrage angehängt werden, um zusätzliche Daten zu übermitteln.
- Beispiel: /users?age=25&status=active

URL-Parameter (Path Parameters):

- Direkt in der URL definiert.
- Werden genutzt, um Ressourcen eindeutig zu identifizieren.
- Beispiel: /users/{id} → /users/123

HTTP-Body (Payload):

- Übertragung von Daten im Nachrichtenkörper.
- Besonders bei POST und PUT-Methoden relevant.
- Beispiel: {"name": "John", "email": "john@example.com"}

Datenübermittlung in RESTful-APIs

	Abfrageparameter	URL-Parameter	HTTP-Body
Beispiel	/users?age=25&status=active	/users/123	{"name": "John", "email": "john@example.com"}
Verwendung für	Für optionale Angaben und komplexere Anforderungen.	Nur für zwingend notwendige Ressourcen verwenden	wenn umfangreiche Daten mitgegeben werden müssen
Methoden	GET, POST, PUT, DELETE	GET, POST, PUT, DELETE	POST, PUT

Datenübermittlung im Header



Header-Daten dienen vor allem der Steuerung, Identifikation und Sicherheit, sind jedoch nicht für die direkte Übertragung der Hauptdaten (wie Nutzdaten in JSON oder XML) vorgesehen.

Zum Beispiel:

- Übermittlung von **Anmelddaten** oder **Token**
 - Festlegen oder Aushandeln des **Datenformats** zwischen Client und Server
 - Anpassung der Antwort basierend auf der **Region oder Sprache** des Benutzers.
- 

Einsatz bei allen CRUD-Methoden

Requests

- weitverbreitetste und beliebteste Bibliothek für HTTP-Anfragen in Python
- Ermöglicht einfache, intuitive Nutzung des HTTP-Protokolls
- Gut geeignet für Web-Scraping, API-Kommunikation und Datenzugriff



Demo requests

/material_requests/demo_requests.py

Übung

1. Rufe die Dokumentation auf
<https://date.nager.at/swagger/index.html>
2. Welche Länder sind in der API verfügbar und welches ist der entsprechende Ländercode?
3. Welche Feiertage hatte Deutschland in 2024?
4. Rufe die Dokumentation auf <https://catfact.ninja/>
5. Lasse 5 Katzen Rassen (breeds) ausgeben.

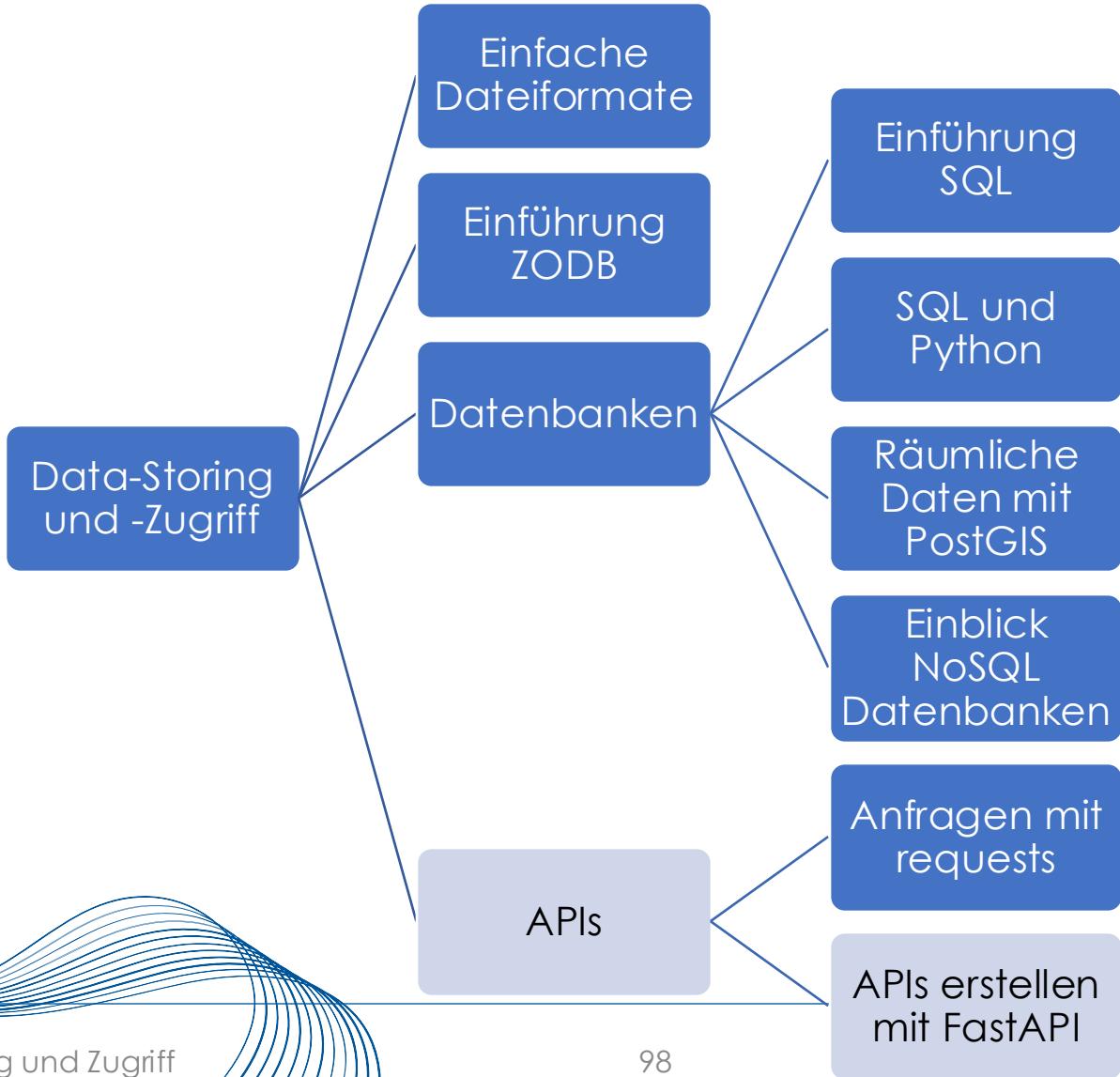
Authentifizierung in APIs

- Schutz sensibler Daten und Ressourcen
- Kontrolle und Nachverfolgung von API-Zugriffen

Authentifizierungsmethoden:

- API-Schlüssel (API Keys)
- Basic Authentication
- OAuth 2.0
- Bearer Tokens

Agenda



- modernes Python-Web-Framework für APIs.
- Entwickelt für schnelles Arbeiten und hohe Leistung.
- Unterstützt asynchrone Programmierung (async/await).

Wieso FastAPI?

- Schnelle Entwicklung: Automatische API-Dokumentation mit Swagger und ReDoc.
- Einfach & Benutzerfreundlich: Intuitive Syntax, weniger Boilerplate-Code.
- Hoch performant: Nutzt Starlette und Pydantic für maximale Geschwindigkeit.
- Eingebaute Validierung: Automatische Validierung von Eingaben und Typen.
- Modularität: Perfekt geeignet für skalierbare und modulare Anwendungen.

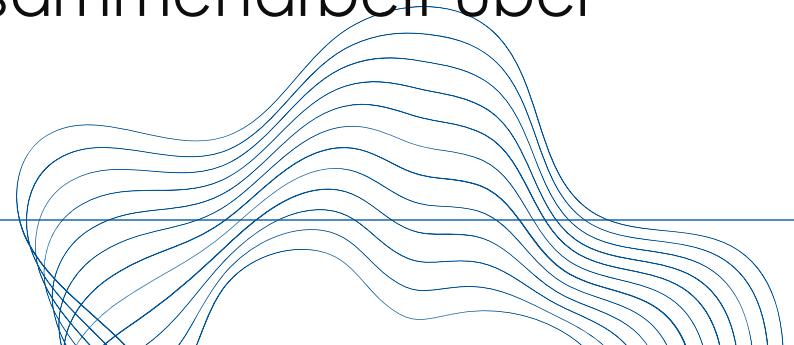
Demo FastAPI

/materialien_fastapi/demo_fastapi_1.py

Dokumentation mit OpenAPI



- OpenAPI-Dokumentation wird **automatisch** aus dem Code erstellt
- **Swagger UI** (/docs) und **ReDoc** (/redoc) sind direkt verfügbar.
- Benutzerfreundliches Webinterface für die API-Exploration.
- Interaktive Tests der Endpunkte ohne zusätzliche Tools.
- Nutzt den OpenAPI-Standard (**einheitliches, maschinenlesbares Format** zur Beschreibung von RESTful APIs)
- Ermöglicht effiziente Kommunikation und Zusammenarbeit über Teamgrenzen hinweg.



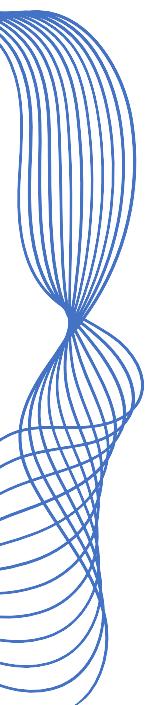
Demo FastAPI Dokumentation

/materialien_fastapi/demo_fastapi_1.py

Übung FastAPI

Erstelle eine get-Route, die per Default auf Deutsch grüßt.
Optional kann ein query Parameter „language“ mitgegeben werden, sodass bei Wert „en“ auf englisch grüßt wird.

Dependency Injection als Design Pattern



Dependency Injection = Ein Entwurfsmuster, bei dem Abhängigkeiten (Dependencies) von außen bereitgestellt werden, anstatt sie innerhalb einer Klasse zu instanziiieren.

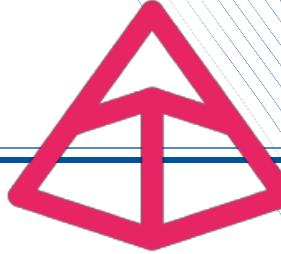
- Ziel: **Entkopplung** von Komponenten für bessere Wartbarkeit und Testbarkeit.
 - **Modularität**: Komponenten können unabhängig voneinander entwickelt und getestet werden.
 - **Flexibilität**: Einfache Austauschbarkeit von Implementierungen (z. B. Mocking für Tests).
 - **Wartbarkeit**: Weniger Verknüpfung zwischen Modulen, was Änderungen erleichtert.
 - **Klarheit**: Abhängigkeiten sind explizit und leicht nachvollziehbar.
- 

Demo Dependency Injection für DB Zugriff

In /material_fastapi

- `demo_fastapi_2.py`
- `demo_fastapi_3.py`

Pydantic



Python-Bibliothek für **Datenvalidierung** und **Datenparsing**.

- Validierung durch Typen: Unterstützt strikt typisierte Modelle mit Python-Typannotationen.
- Datenqualität sicherstellen: Automatische Validierung von Feldern, wie E-Mail-Adressen, Datumsformaten oder Zahlenbereichen.
- Performance: Nutzt Cython für **extrem schnelle** Validierung.
- Einfache Nutzung: **Klar und intuitiv** dank einer API, die auf Dataclasses basiert.
- Flexibilität: Unterstützt verschachtelte Modelle und komplexe Validierungslogik.
- JSON-Handling: Integrierte Unterstützung für JSON und Dict-Datenkonvertierung.
- Fehlermeldungen: Liefert detaillierte, verständliche Fehlermeldungen bei ungültigen Eingaben.
- Einsatzmöglichkeiten: **Perfekt für API-Entwicklung**, Eingabeverarbeitung und Datenmodelle.

Demo pydantic

In /material_fastapi/demo_pydantic.py

Übung pydantic

Erstelle eine Klasse „User“ mit pydantic, die folgende Felder besitzt:

Name (str), E-Mail (str), Adresse (str) und Geburtsdatum (date, per Default None)

Demo pydantic in FastAPI

In /material_fastapi

- demo_fastapi_4.py

Was ist Async?

- **Asynchrones Programmieren:** Programmierparadigma, das es ermöglicht, mehrere Aufgaben gleichzeitig zu bearbeiten, ohne auf die Fertigstellung anderer Aufgaben zu warten.
- **Event-Loop basiert:** Verwaltet Aufgaben effizient, indem es Leerlaufzeiten wie das Warten auf I/O-Operationen minimiert.

Vorteile von async:

➤ Höhere Performance:

- Verarbeitet mehrere Anfragen gleichzeitig, ideal für APIs mit hohem Traffic.
- Reduziert die Blockierung bei langsamem I/O-Operationen (z. B. Datenbankzugriffe).

➤ Effiziente Ressourcen-Nutzung:

- Spart Server-Ressourcen durch nicht blockierende Operationen.
- Weniger Threads/Prozesse notwendig.

➤ Schnellere Antwortzeiten:

- Minimiert Verzögerungen durch parallele Bearbeitung von Anfragen.

Demo FastAPI Produktiv

In /material_fastapi_prod

Übung

Erweitere die API um eine zusätzliche Route. Diese soll als Argument den Nachnamen besitzen. Anschließend wird in der Datenbank nach dem Nachnamen gesucht und alle Treffer zurückgegeben.

= Open-Source Remote Procedure Call (RPC)-Framework von Google zur effizienten Kommunikation zwischen Systemen.

- Ermöglicht die direkte Interaktion zwischen Client und Server durch definierte Methodenaufrufe.
- Basierend auf **Protocol Buffers** (Protobuf) zur Serialisierung von Daten.
- Unterstützt mehrere Programmiersprachen und Plattformen.
- **Hohe Performance** dank HTTP/2 (schnelle und parallele Datenübertragung).

Demo gRPC

In /material_grpc:
start_grpc.txt

gRPC vs. FastAPI

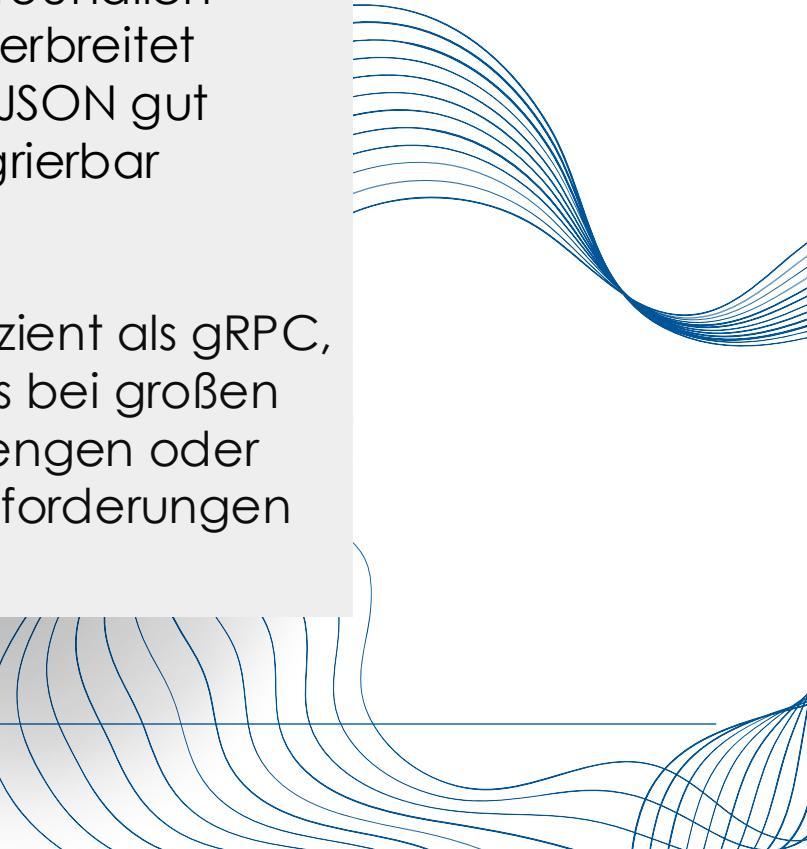


gRPC

- Schnell und effizient (durch HTTP2 und Protobuf)
- Weit verbreitet
- Im Vergleich komplexer in der Umsetzung

FastAPI

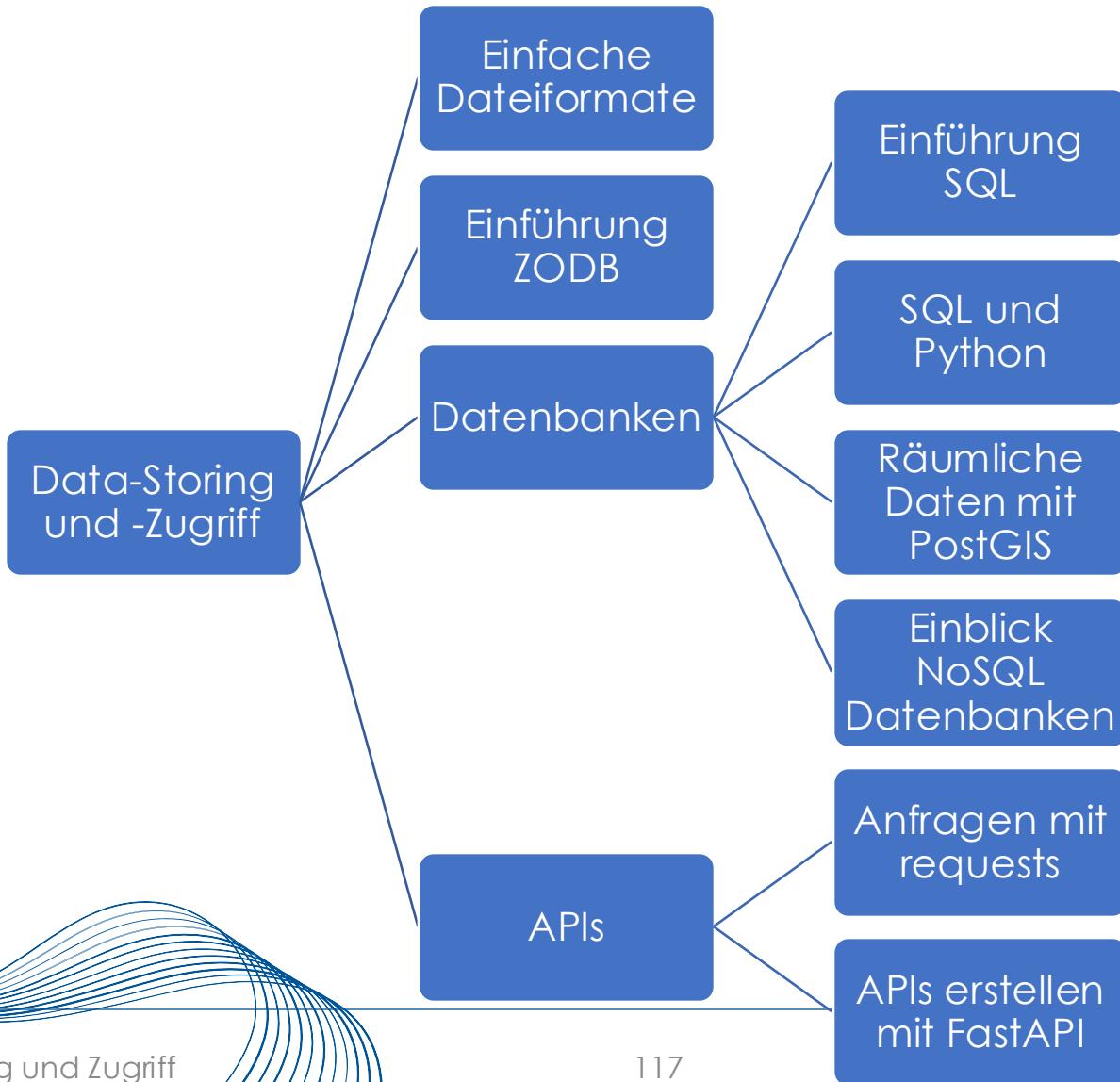
- Einfach und nutzerfreundlich
- Weit verbreitet
- Durch JSON gut integrierbar
- Weniger effizient als gRPC, besonders bei großen Datenmengen oder Echtzeitanforderungen



Ausblick Datenspeicherung

- In Memory Datenbank
 - z.B. Redis, Memcached
- Streams
 - z.B. Kafka
- Transaktionslogs

Rückblick



Glossar

Glossar I

ACID: ACID steht für Atomarität, Konsistenz, Isolation und Dauerhaftigkeit und beschreibt die Eigenschaften von Transaktionen in Datenbanken.

- Atomarität: Operationen sind unteilbar und werden entweder vollständig oder gar nicht ausgeführt.
- Konsistenz: Die Datenbank bleibt vor und nach einer Transaktion in einem konsistenten Zustand.
- Isolation: Gleichzeitige Transaktionen beeinflussen sich nicht gegenseitig.
- Dauerhaftigkeit: Änderungen bleiben nach Abschluss einer Transaktion auch bei Systemausfällen erhalten.

Assoziationsanalyse: Die Assoziationsanalyse dient der Identifizierung von häufig auftretenden Item-Kombinationen (z. B. in Warenkörben). Sie basiert auf "Wenn-Dann"-Regeln, die die Wahrscheinlichkeit angeben, dass ein bestimmtes Item gekauft wird, wenn ein anderes Item bereits gekauft wurde.

Asynchrones Programmieren: Asynchrones Programmieren ermöglicht die gleichzeitige Bearbeitung mehrerer Aufgaben, ohne auf die Fertigstellung einzelner Aufgaben warten zu müssen. Dies führt zu einer erhöhten Performance und Reaktionsfähigkeit von Anwendungen, insbesondere bei I/O-intensiven Operationen wie Datenbankzugriffen oder Netzwerkkommunikation.

Ausreißer- und Anomalie-Erkennung: Ausreißer und Anomalien sind Datenpunkte, die von den übrigen Daten abweichen. Sie können auf Messfehler, Betrug oder ungewöhnliche Ereignisse hinweisen.

Glossar II

Authentifizierung: Authentifizierungsmechanismen dienen dazu, den Zugriff auf APIs zu kontrollieren und zu sichern. Sie stellen sicher, dass nur autorisierte Benutzer auf die Ressourcen der API zugreifen können. Verschiedene Methoden wie API-Schlüssel, Basic Authentication, OAuth 2.0 und Bearer Tokens können verwendet werden.

Backups und Wiederherstellung: Backups und Wiederherstellungsmechanismen sind essenziell für die Sicherheit und Verfügbarkeit von Daten. Backups erstellen Kopien der Daten, die im Falle eines Datenverlustes wiederhergestellt werden können. Die Wiederherstellung kann mithilfe verschiedener Methoden erfolgen, beispielsweise durch Point-in-Time Recovery (PITR), bei der die Datenbank zu einem bestimmten Zeitpunkt wiederhergestellt wird.

Clusteranalyse: Die Clusteranalyse dient der Gruppierung von Datenpunkten in Cluster, die ähnliche Eigenschaften aufweisen. Es gibt verschiedene Verfahren der Clusteranalyse, darunter regelbasierte Verfahren, die auf vordefinierten Regeln basieren, und Machine-Learning-Verfahren, die die Cluster automatisch aus den Daten lernen.

CRUD: CRUD steht für Create, Read, Update, Delete und beschreibt die vier grundlegenden Operationen, die in den meisten Datenbanksystemen und APIs verwendet werden.

Data Analytics: Data Analytics konzentriert sich auf die Analyse und Interpretation von Daten, um konkrete Fragen zu beantworten oder fundierte Entscheidungen zu treffen. Ziel ist es, bereits vorhandene Informationen besser zu verstehen, Trends zu erkennen und datengestützte Entscheidungen zu treffen.

Glossar III

Data Mining: Data Mining befasst sich mit dem Aufspüren verborgener Muster und Zusammenhänge in umfangreichen Datensätzen. Der Schwerpunkt liegt auf der Gewinnung neuer Erkenntnisse, die nicht auf den ersten Blick erkennbar sind. Dieser Prozess wird oft als explorativ bezeichnet. Data Mining verwendet ähnliche Verfahren wie Machine Learning, zum Beispiel K-Means oder Entscheidungsbäume. Es dient auch der Vorbereitung von Daten für Machine Learning.

Data Storing und -Zugriff: Dieser Bereich umfasst verschiedene Methoden zur Speicherung und zum Zugriff auf Daten:

- Einfache Dateiformate: CSV, JSON, Protobuf, XML, YAML, Avro, Thrift.
- Datenbanken: ZODB, SQL-Datenbanken (PostgreSQL, SQLite), NoSQL-Datenbanken (MongoDB).
- APIs: Schnittstellen zur Kommunikation zwischen Anwendungen.

Datenbank-Skalierung: Die Skalierung von Datenbanken ermöglicht es, die Leistung und Verfügbarkeit von Datenbanken zu verbessern, indem die Daten auf mehrere Server verteilt werden. Dies kann durch Partitionierung (Aufteilung einer Tabelle in kleinere Teile) oder Replikation (Kopieren der Datenbank auf mehrere Server) erreicht werden.

Dependency Injection: Dependency Injection ist ein Entwurfsmuster, das die Entkopplung von Softwarekomponenten fördert, indem Abhängigkeiten von außen bereitgestellt werden. Dies erhöht die Modularität, Flexibilität und Wartbarkeit von Software und erleichtert das Testen von Komponenten.

Fehlende Werte: Fehlende Werte sind ein häufiges Problem in Datensätzen. Sie können verschiedene Ursachen haben, z. B. Messfehler, fehlende Informationen oder bewusstes Auslassen von Daten.

Glossar III

gRPC: gRPC ist ein Framework für Remote Procedure Calls (RPC), das eine effiziente Kommunikation zwischen Systemen ermöglicht. gRPC verwendet Protocol Buffers (Protobuf) zur Serialisierung von Daten und unterstützt verschiedene Programmiersprachen.

Indizes: Indizes dienen zur Beschleunigung von Datenabfragen in Datenbanken. Sie funktionieren ähnlich wie ein Inhaltsverzeichnis in einem Buch und ermöglichen es, schnell auf bestimmte Datensätze zuzugreifen, ohne die gesamte Tabelle durchsuchen zu müssen.

Korrelation: Die Korrelation beschreibt den linearen Zusammenhang zwischen zwei Variablen. Sie kann Werte zwischen -1 und +1 annehmen, wobei -1 einen perfekten negativen Zusammenhang, 0 keinen Zusammenhang und +1 einen perfekten positiven Zusammenhang darstellt.

Machine Learning: Bei Machine Learning kommen Algorithmen zum Einsatz, die aus Daten lernen, um Vorhersagen zu treffen oder Entscheidungen zu fällen. Für das Training von Modellen werden die Daten in Trainings-, Validierungs- und Testdaten unterteilt.

Median: Der Median ist ein weiteres zentrales Lagemaß, das den Wert angibt, der die Daten in zwei gleich große Hälften teilt.

Mittelwert / Durchschnitt: Der Mittelwert (auch Durchschnitt genannt) ist ein zentrales Lagemaß in der Statistik. Er wird berechnet, indem die Summe aller Werte durch die Anzahl der Werte geteilt wird.

Glossar V

Primär- und Fremdschlüssel: In relationalen Datenbanken dienen Primär- und Fremdschlüssel zur Verknüpfung von Tabellen. Der Primärschlüssel ist eine eindeutige Kennung für jede Zeile in einer Tabelle, während der Fremdschlüssel auf den Primärschlüssel einer anderen Tabelle verweist und so die Beziehung zwischen den Tabellen herstellt.

ORM (Object-Relational Mapping): ORM ermöglicht die Verbindung von Objekten in einer Programmiersprache mit Datenbanktabellen. Dadurch können Entwickler mit Datenbanken arbeiten, ohne direkt SQL schreiben zu müssen, was die Entwicklung vereinfacht und den Code lesbarer macht.

Perzentile und Quartile: Perzentile und Quartile sind Wertschranken, die die Daten in bestimmte Anteile aufteilen. Das 25%-Perzentil (auch unteres Quartil genannt) ist der Wert, unterhalb dessen 25% der Daten liegen.

PostGIS: PostGIS ist eine Erweiterung für die relationale Datenbank PostgreSQL, die die Verarbeitung von Geodaten ermöglicht. Mit PostGIS können räumliche Daten wie Punkte, Linien und Polygone in der Datenbank gespeichert und abgefragt werden. PostGIS bietet Funktionen zur Berechnung von Distanzen, Flächen und Schnittmengen sowie zur Transformation von Koordinatensystemen.

Pydantic: Pydantic ist eine Python-Bibliothek, die Datenvielfältigung und -parsing ermöglicht. Pydantic verwendet Typannotationen, um Datenmodelle zu definieren, und validiert automatisch die Eingabedaten anhand dieser Modelle.

Python-Bibliotheken: pandas, numpy, intake, SQLAlchemy, requests, FastAPI.

Glossar VI

RESTful API: RESTful APIs sind ressourcenorientierte Architekturen für APIs, die auf dem HTTP-Protokoll basieren. Jede Ressource (z. B. ein Benutzer, ein Artikel) wird durch eine eindeutige URL adressiert. Zur Interaktion mit den Ressourcen werden die HTTP-Methoden GET, POST, PUT und DELETE verwendet.

Saisonalität: Saisonalität beschreibt regelmäßige Schwankungen in Daten über die Zeit, die auf wiederkehrende Ereignisse wie Jahreszeiten, Feiertage oder Wochentage zurückzuführen sind.

Serialisierung: Die Serialisierung beschreibt die Umwandlung eines Objekts in ein Format, das gespeichert oder übertragen werden kann. Dies ist wichtig für die Speicherung von Daten in Datenbanken oder Dateien sowie für die Übertragung von Daten über Netzwerke, beispielsweise in APIs oder Remote-Verbindungen.

Standardabweichung: Die Standardabweichung ist ein Streuungsmaß, das die durchschnittliche Abweichung der Werte vom Mittelwert angibt.