Fachhochschule Südwestfalen
University of Applied Sciences

Seminararbeit Projektgruppe Transparenzregister

Datenspeicherung

SoSe 2023

Bearbeitet von:

Sebastian Zeleny 10038655

Betreut durch:

Prof. Dr.-Ing. Doga Arinir

Inhaltsverzeichnis

Motivation: Warum speichern wird Daten?

Für die Speicherung von Daten gibt es verschiedene Motivationen:

- **Sammlung:** Zur Aufbewahrung von Wissen und Informationen über Objekte, Ereignisse oder Prozesse werden Daten gespeichert.
- **Historisierung:** Durch die Speicherung von Daten in einem zeitlichen Zusammenhang, wird eine Historie erstellt, mit welcher Muster, Trends oder Zusammenhänge erkannt werden können. Historische Daten helfen ausserdem bei der Entscheidungsfindung.
- **Bewertung:** Mit gespeicherten Daten können Systeme, Produkte und Prozesse nachvollzogen, bewertet und verbessert werden.

Im Projekt Transparenzregister ist die Datenspeicherung eine Kernkomponente, da die gesammelten Informationen die Grundlage für Analysen darstellen.
Mit geeigneten Pipelines werden aus diesen Daten Erkenntnisse extrahiert, um z.B. Verflechtungen zwischen Personen und Unternehmen oder den wirtschaftlichen Trend eines Unternehmens visualisieren und bewerten zu können.

1. Allgemeine Anforderungen an Datenbank

- **1.1 Speicherung/Integrität**: Das verwendete System muss Daten, wie Unternehmenskennzahlen, Stammdaten und Verflechtungen speichern. Die Daten müssen korrekt und konsistent gespeichert werden. Konsistent bedeutet in einem gültigen und widerspruchsfreien Zustand und die Transaktionen sollen den ACID-Eigenschaften entsprechen.
    - **Atomarity:** Eine Transaktion wird atomar betrachte, d.h. es ist die kleinste unteilbare Einheit, wodurch eine Transaktion entweder vollständig durchgeführt und übernommen wird (Commit) oder bei einem Fehler rückgängig gemacht wird (Rollback).
    - **Consistency:** Konsistenz bedeutet, dass eine Transaktion den Datenbankzustand von einem gültigen in einen anderen gültihgen Zustand überführt. Sollte eine Transaktion eine Konsitenzverletzung verursachen, wird diese abgebrochen und die Änderungen zurückgesetzt.
    - **Isolation:** Isolation sorgt dafür, dass alle Transaktion unabhängig voneinander ausgeführt werden, damit sich diese bei der Ausführung nicht gegenseitig beeinflussen.
    - **Durability:** Dauerhaftigkeit bedeutet, dass die Ergebnisse einer Transaktion dauerhaft in der Datenbank gespeichert werden und auch nach einem Systemneustart oder Systemfehler erhalten bleiben.
- **1.2 Skalierbarkeit:** Das System soll skalierbar sein, um zukünftige Daten weiterhin zu speichern und weitere Unternehmen hinzuzufügen. Durch Hinzufügen von Ressourcen kann das System an steigende Datenmengen und Benutzeranforderungen angepasst werden. Man spricht von horizontaler Skalierung, da die Last auf mehrere Datenbankserver verteilt wird.
- **1.3 Sicherheit:** Die Datenbank muss Mechanismen bereitstellen, um die Daten vor unbefugtem Zugriff zu schützen.
    - **Authentifizierung:** Überprüfung der Identität eines Benutzers, durch Benutzername und Passwort. Meist wird eine Zwei-Faktor-Authentifizierung verwendet, um das Sicherheitslevel zu erhöhen.

    - **Autorisierung:** Der authentifizierte Benutzer erhält bei der Autorisierung Zugriffsrechte und Ressourcen, welche auf seiner Benutzerrolle basieren. Ein Benutzer mit Administratorrechten, erhält Zugriff auf alle Systemressourcen, wohingegen ein normaler Benutzer nur beschränkten Zugriff erhält.
    - **Verschlüsselung:** Durch Verschlüsselung werden Daten in ein nicht interpretierbaren Code umgewandelt, um den Inhalt vor unbefugtem Zugriff zu schützen. Dafür wird ein Algorithmus verwendet, welcher einen Schlüssel generiert und die Daten mit diesem verschlüsselt. Um die Daten wieder lesen zu können, müssen diese mit dem Schlüssel dechiffriert werden.
- **1.4 Datensicherung- und Wiederherstellung:** Die Datenbank muss Funktionen zur Sicherung und Wiederherstellung unterstützen. Im Falle eines Ausfalls oder Fehlers muss sichergestellt sein, dass Mechanismen die Daten schützen und wiederherstellen. Die meisten Daten in einer Datenbank ändern sich nur langsam, manche allerdings schnell. Je nach Anwendungsfall muss eine geeignete Sicherungsstrategie ausgewählt werden, um nur die Daten zu sichern, die sich tatsächlich ändern. Jedes Datenbankmanagementsystem bietet unterschiedliche Mechanismen zur Datensicherung und Wiederherstellung, dessen Möglichkeiten nach Auswahl eines Systems
    - **vollständiges Backup:** Das vollständige Backup ist eine komplette Kopie der Datenbank inkl. aller Daten, Indizes, Tabellen und Metadaten. Es benötigt viel Speicherplatz und Zeit zur Erzeugung der Sicherung und auch zur Wiederherstellung.
    - **inkrementelles Backup:** Ein inkrementelles Backup sichert nur die Änderungen seit dem letzten vollständigem bzw. inkrementellen Backup. Durch den verringerten Datenbestand ist es deutlich schneller und datensparsamer, als das vollständige Backup. Zur Wiederherstellung wird das letzte vollständige und alle inkrementellen Backups benötigt. Allerdings entsteht eine Abhängigkeitskette, da jedes Backup seine Vorgänger zur Wiederherstellung benötigt.
    - **differentielles Backup:** Beim differentiellen Backup werden alle Änderungen seit dem letzten vollständigem Backup gesichert. D.h. je weiter die letzte vollständige Sicherung zurückliegt, desto größer und langsamer wird das Backup. Zur Wiederherstellung werden das letzte vollständige und differentielle Backup benötigt.

Backuphäufigkeit: Die Backuphäufigkeit ist eine Abwägung aus Risiken, Kosten und Arbeitsaufwand. Dieses muss individuell abgeschätzt werden aufgrund des Datenbankumfangs und der Änderungshäufigkeit der Daten, um eine geeignete Backup-Strategie zu entwerfen.
Beispiel:

- Vorgabe: Datenbank mit 500GB Größe
    - Anforderungen
        - min. vierfache Backupkapazität --> 2 TB
        - Backupdauer vollständig:
          USB 2.0: $\frac{500GB}{\frac{60MB/s}{1024}} = 8533sec. \approx 142Min. \approx 2,37Std.$
          USB 3.0: $\frac{500GB}{\frac{625MB/s}{1024}} = 820sec. \approx 13,6Min. \approx 0,23Std.$
          VDSL 100: $\frac{500GB}{\frac{5MB/s}{1024}} = 102400sec. \approx 1706Min. \approx 28,4Std.$
          Glasfaser: $\frac{500GB}{\frac{62,5MB/s}{1024}} = 8192sec. \approx 136,5Min. \approx 2,3Std.$
- **1.5 Leistung:** Die Performanceanforderungen an die Datenbank ergibt sich aus verschiedenen Merkmalen. Diese können kombiniert gestellt werden und sind abhängig von den Anforderungen an

das System. Eine Analyse der Anwendungsfälle ist notwendig, um die Anforderungen zu spezifizieren.

  - **Latenz:** Die Datenbank soll Anfragen effizient und in einer akzeptablen Antwortzeit verarbeiten. Typische Datenbankapplikationen, wie z.B. ein Webshop benötigen viele einzelne Zugriffe, wofür jedes Mal ein Kommunikationsprotokoll angewendet wird. Durch viele kleine Datenbankzugriffe wird die Applikation verlangsamt, da auf die Netzwerkkommunikation gewartet wird. Für das Benutzererlebnis eines Webshops ist die Latenz ein wichtiges Merkmal.
  - **Durchsatz:** Ist eine Metrik für die Anzahl an Transaktionen pro Zeiteinheit. Der Durchsatz ist wichtig bei großen Benutzeraufkommen in einem Webshop.
  - **Verfügbarkeit:** Eine hohe Verfügbarkeit, also Erreichbarkeit der Datenbank, wird durch Redundanz (mehrfaches Vorhandensein) und Wiederherstellungsmechanismen gewährleistet, damit Daten koninuierlich verfügbar sind.
  - **Wartbarkeit:** Eine einfach zu wartende Datenbank muss Funktionen zur Überwachung, Diagnose, Wartung, Datensicherung und Wiederherstellung bereitstellen. Durch diese automatisierten Pipelines können andere Eigenschaften, wie z.B. die Verfügbarkeit negativ beeinflusst werden, weil Prozesse die Datenbank blockieren.

- **1.6 Integration:** Die Datenbank muss Schnittstellen bereitstellen, um die gespeicherten Daten für eine Anwendung bzw. Systeme zur Verfügung zu stellen.

  - **API:** Das *Application Programming Interface* ist eine definierte Schnittstelle, welche Methoden und Funktionen bereit stellt, um auf die Datenbank zuzugreifen bzw. um diese zu verwalten.
    - **REST:** REpresential State Transfer beschreibt eine Schnittstelle, die das http-Protokoll verwendet, wo mit den Methoden GET, POST, PUT, DELETE die Kommunikation realisiert wird.
    - **SOAP:** Simple Object Access Protocol ist eine Schnittstelle, welche auf XML basiert.
    - **ODBC:** Open Database Connectivity ist eine standardisierte Schnittstelle zum Austausch zwischen Anwendungen und Datenbanken.
    - **JDBC:** Java Database Connectivity

2. Datenarten

Zur Beschreibung von Unternehmen, werden verschiedene Datenarten verwendet. Die folgenden Datenarten sind eine allgemeine Zusammenfassung und sollen das Brainstorming für die projektspezifischen Daten unterstützen.

- **Stammdaten:** Stammdaten beinhalten die grundsätzlichen Eigenschaften und Informationen von realen Objekten, welche für die periodische Verarbeitung notwendig sind. Ein Stammsatz für Personal besteht z.B. aus einer Personalnummer, dem Mitarbeiternamen, Anschrift und Bankverbindung.
  Je nach Anwendungsfall bzw. Geschäftsprozess muss der Inhalt definiert werden, wie z.B. bei Unternehmens-, Kunden-, Material- oder Patientenstammdaten.

- **Metadaten:** Mit Metadaten werden weitere Daten beschrieben und vereinfachen das Auffinden und Arbeiten mit diesen. Metadaten beinhalten beispielsweise den Verfasser, das Erstellungs- oder Änderungsdatum, die Dateigröße oder den Ablageort.
  Mit Metadaten können Datenbestände einfacher und effizienter verwaltet und abgefragt werden.

- **Transaktionsdaten:** Transaktionsdaten beschreiben eine Veränderung des Zustands, wie z.B. eine Kapitalbewegung oder eine Ein-/Auslieferung aus einem Lager.
- **Referenzdaten:** Referenzdaten sind eine Teilmenge von Stammdaten und beschreiben die zulässigen Daten. Diese werden nur selten geändert oder angepasst und gelten als konstant. Beispiele für Referenzdaten sind: Postleitzahlen, Kostenstellen, Währungen oder Finanzhierarchien.
- **Bestandsdaten:** Bestandsdaten sind dauerhafter Veränderung ausgesetzt, da diese z.B. die Artikelmenge in einem Lager oder das Guthaben auf einem Konto beschreiben. Diese korrelieren mit den Transaktionsdaten.

Diese Datenarten müssen im Kontext des Projektes betrachtet werden und sollen das Brainstorming unterstützen.
Stammdaten: Unternehmensname, Anschrift, Branche
Metadaten: Verfasser einer Nachricht - Veröffentlichungsdatum; Prüfungsunternehmen - Prüfdatum
Transaktionsdaten: Wer hat wann wo gearbeitet?
Referenzdaten: Einheit von Metriken (Umsatz, EBIT usw.)
Bestandsdaten: Vorstand, Geschäftsführer, Aufsichtsrat

2.1 Welche Daten erwarten wir im Projekt?

Aus den vorangehenden, allgemeinen Datenarten haben wir Cluster identifiziert, welche im Projekt benötigt werden. Die Kombination aus den folgend aufgeführten Datenclustern ermöglicht eine ganzheitliche Betrachtung und Bewertung der Unternehmen.

- **Unternehmensstammdaten:** Die Stammdaten beinhalten grundlegende Informationen zu einem Unternehmen, wie z.B. Name, Anschrift, Gesellschaftsform und Branche.
- **Sentimentdaten:** Die Sentiment- oder Stimmungsdaten beschreiben die Aussenwahrnehmung des Unternehmens hinsichtlich der Mitarbeiterzufriedenheit, Nachhaltigkeit und Umweltfreundlichkeit.

> Mit Sentimentdaten können folgende Fragen beantwortet werden:
>
> - Welchen Ruf hat das Unternehmen?
> - Wie ist die Aussenwahrnehmung?
> - Wie ist die Kundenbindung?

- **Finanzdaten:** Die Finanzdaten sind Metriken bzw, Indikatoren, um den wirtschaftlichen Erfolg des Unternehmens zu bewerten. Hierzu zählen z.B. Umsatz, EBIT, EBIT Marge, Bilanzsumme, Eigenkapitalanteil, Fremdkapitalanteil, Verschuldungsgrad, Eigenkapitalrentabilität, Umschlaghäufigkeit des Eigenkapitals.

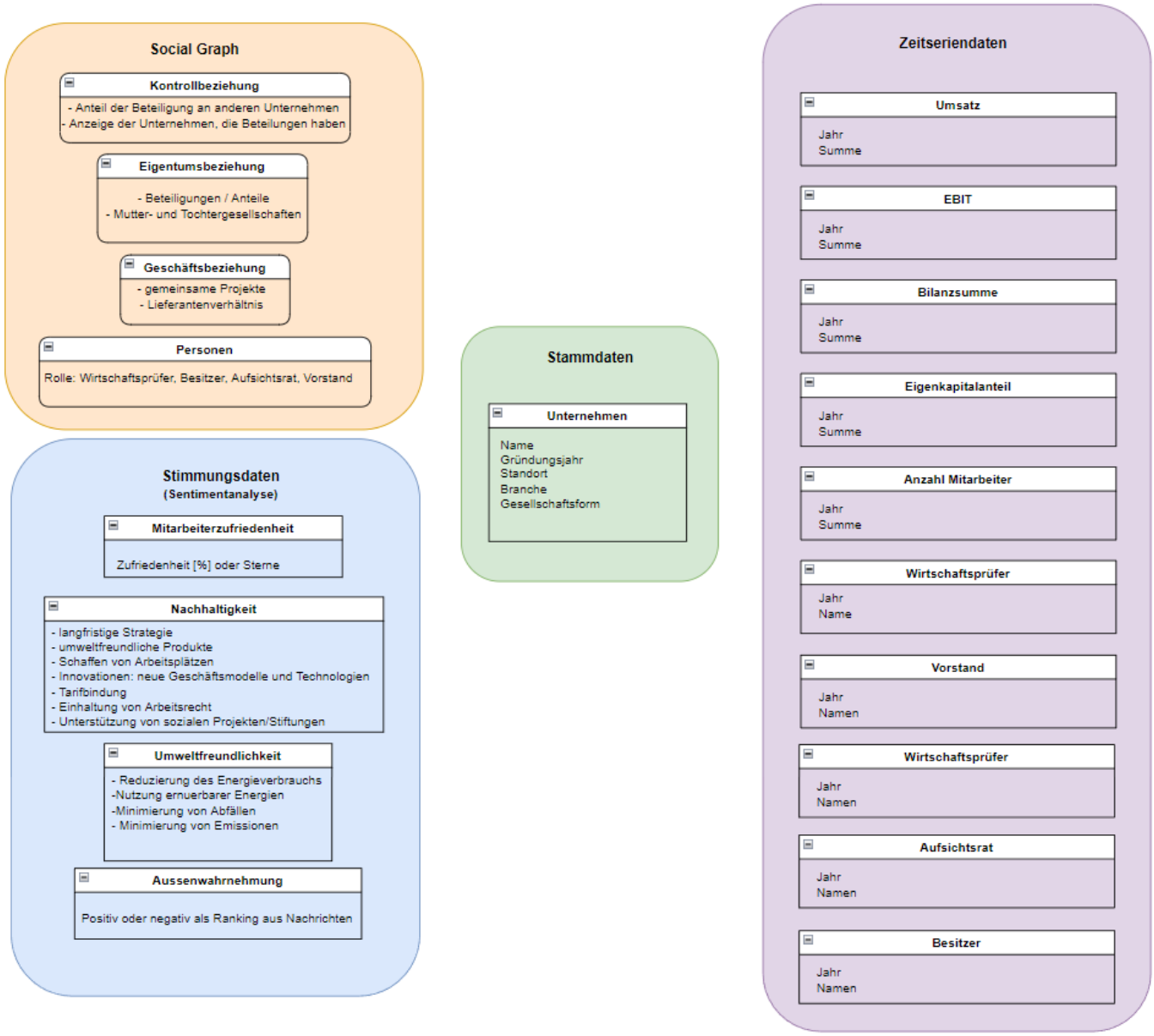
> Mit Finanzdaten können folgende Fragen beantwortet werden:
>
> - Wie rentabel wirtschaftet das Unternehmen?
> - Wie ist der wirtschaftliche Trend?
> - Bewerten anhand verschiedender Metriken.

- **Verflechtungsdaten/Social Graph:** Die Verbindungen bzw. Beziehungen zu Personen oder Unternehmen wird in den Verflechtungsdaten abgelegt. Beziehungen entstehen, wenn eine Person

Geschäftsführer, Vorstand, Aufsichtsratmitglied, Prokurist oder Auditor ist und Unternehmen z.B. gemeinsam arbeiten, beliefert wird oder Anteile an einem anderen Unternehmen besitzt.

> Mit Verflechtungsdaten können folgende Fragen beantwortet werden:
>
> - Gibt es strategische Partnerschaften?
> - Wie sind die Lieferketten aufgebaut?
> - Wie ist die Qualität der Geschäftsbeziehungen?
> - Ist das Unternehmen widerstandsfähig aufgestellt?
> - Gibt es Zusammenhänge zu Personen?

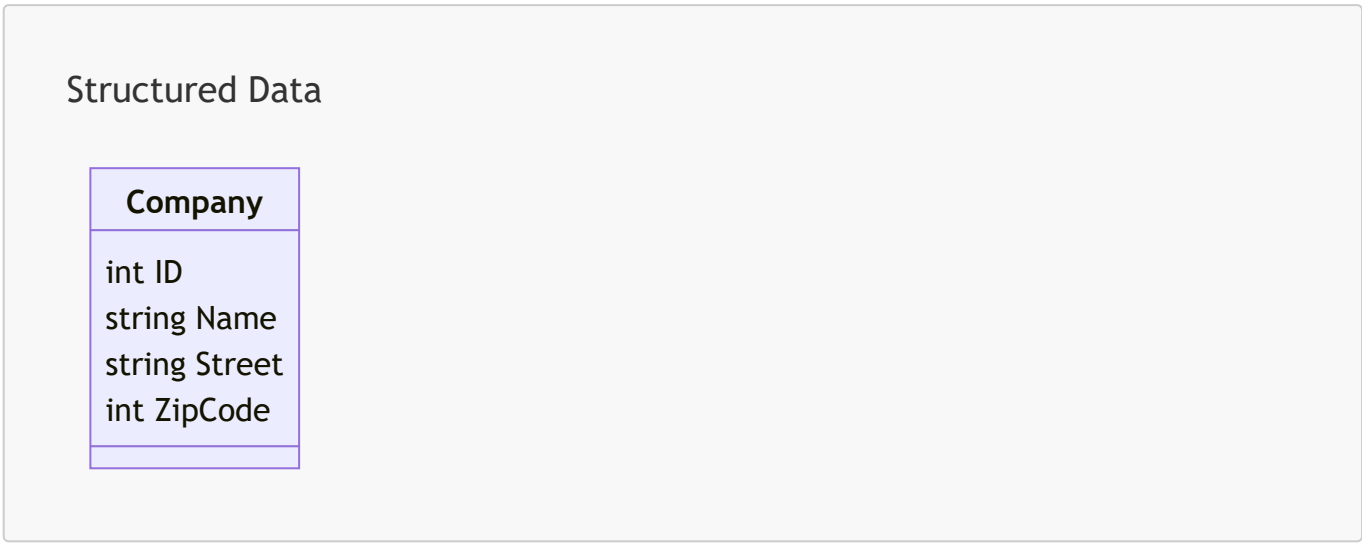
Die abgebildete Mind Map ist nicht vollständig und bildet nicht den finalen Datenumfang des Projekts ab. Es ist eine Momentaufnahme, bevor das relationale Schema entwickelt und die Implementierung begonnen wurde.



2.2 strukturierte Daten

Strukturierte Daten liegen in einem definierten Format. Vorab wird ein Schema definiert, um Felder, Datentypen und Reihenfolgen festzulegen und die Daten entsprechend abzulegen. Diese Art von Daten wird

z.B. in relationalen Datenbanken verwendet, wobei jede Zeile einer Tabelle einen Datensatz repräsentiert. Die Beziehungen untereinander sind über die Entitäten definiert. Das Beispiel unten zeigt ein einfaches Beispiel, wie die Daten für die Klasse *Company* definiert sind. Mit diesem Schema kann die Datenaufbereitung umgesetzt werden.



Vorteile	Nachteile
einfach nutzbar, da organisiert	Einschränkung der Verwendungsmöglichkeit durch Schema
bei bekannten Schema sind Werkzeuge vorhanden	begrenze Speichermöglichkeit, da starre Schemata vorgegeben sind
gut automatisierbar	

2.3 unstrukturierte Daten

Unstrukturierte Daten unterliegen keinem Schema, wie z.B. E-Mails, Textdokumente, Blogs, Chats, Bilder, Videos oder Audiodateien.

- **Textanalyse:** Aus unstrukturierten Texten werden z.B. durch Analyse und Mining Informationen gewonnen, um diese zu extrahieren. Es wird das Vorkommen von bestimmten Wörtern mittels Named Entity Recognition ermittelt oder die Stimmung bzw. das Thema in einem Artikel.
- **Audio-/Videoanalyse:** Bei der Verarbeitung unstrukturierter Audio- oder Videodateien werden Objekte, Gesichter, Stimmen oder Muster erkannt, um diese für Sprachassistenten oder autonome Fahrzeuge nutzbar zu machen.

Eine wichtige Informationsquelle sind unstrukturierte Daten für Explorations- und Analyseaufgaben. Dabei werden Datenquellen wie z.B. E-Mails, RSS-Feeds, Blogs durchsucht, um bestimmte Informationen zu finden oder Zusammenhänge zwischen verschiedenen Quellen hherzustellen. Dies ermöglicht tiefe Einsicht in die Daten zu erhalten und unterstützt die Entscheidungsfindung bei unklaren Sachverhalten und die Entdeckung neuer Erkenntnisse.

Vorteile	Nachteile
großes Potenzial Erkenntnisse zu erlangen	aufwändige Bearbeitung notwendig, um Daten nutzbar zu machen

Vorteile	Nachteile
unbegrenzte Anwendungsmöglichkeiten, da kein Schema vorhanden ist	spezielle Tools zur Aufbereitung notwendig
	Expertenwissen über die Daten und Datenaufbereitung notwendig

3. Arten von Datenbanken

3.1 Relationale Datenbank

Eine relationale Datenbank speichert und verwaltet strukturierte Daten. Dabei werden die Daten in Tabellen organisiert, welche aus Zeilen und Spalten bestehen.
In den Zeilen der Tabellen sind die Datensätze gespeichert, d.h. jede Zeile repräsentiert einen Datensatz.
Durch logisches Verbinden der Tabellen können die Beziehungen zwischen den Daten abgebildet werden.
Die wichtigsten Elemente einer relationalen Datenbank werden folgend erklärt:

Tabelle: Eine Tabelle repräsentiert eine Entität bzw. Objekt , wie z.B. Unternehmen, Kunde oder Bestellung. Die Tabelle besteht aus Spalten, welche die Attribute der Entität speichern.
Jede Zeile ist eine Instanz des Objekts und enthält konkrete Werte.

Table_Person

ID	Name	Age	Salary	Height
1	Tim	31	300.00	191.20
2	Tom	21	400.00	181.87
3	Tam	51	500.00	176.54

https://www.sqlservercentral.com/articles/creating-markdown-formatted-text-for-results-from-sql-server-tables

Primärschlüssel: Der Primärschlüssel ist ein eindeutiger Bezeichner für jede einzelne Zeile einer Tabelle und wird zur Identifikation einer einzelnen Zeile benötigt. Im oberen Beispiel ist die Spalte *ID* der Primärschlüssel.

Fremdschlüssel: Ein Fremdschlüssel verweist auf einen Primärschlüssel einer anderen Tabelle, um eine Beziehung zwischen den Tabellen herzustellen.
Im Beispiel ist bezieht sich die Spalte *customer_id* auf den Primärschlüssel der Tabelle *Table_Person*.

Table_Orders

ID	Product	total	customer_id
1	Paper	12	2
2	Book	3	2
3	Marker	5	3

Beziehungen: Wie bereits beschrieben, können mit der Verwendung von Fremdschlüsseln Beziehungen zwischen den Tabellen hergestellt werden.
Es gibt verschiedene Beziehungstypen:

Typ	Beschreibung
1:1	Jeder Primärschlüsselwert bezieht sich auf nur einen Datensatz. **Beispiel:** Jede Person hat genau eine Bestellung.
1:n	Der Primärschlüssel ist eindeutig, tritt in der bezogenen Tabelle 0..n mal auf. **Beispiel:** Jede Person kann keine, eine oder mehrere Bestellungen haben.
n:n	Jeder Datensatz von beiden Tabellen kann zu beliebig vielen Datensätzen (oder auch zu keinem Datensatz) stehen. Meist wird für diesen Typ eine dritte Tabelle verwendet, welche als Zuordnungs- bzw. Verknüpfungstabelle angelegt wird, da andernfalls keine direkte Verbindung hergestellt werden kann.

https://www.ibm.com/docs/de/control-desk/7.6.1.2?topic=structure-database-relationships

3.1.1 Anlegen von Tabellen

Der Umgang von relationalen Datenbanken erfolgt mittels SQL. Folgend ein Beispiel zum Anlegen einer Tabelle mit Attributen.

```
CREATE TABLE Bildungsstaette (
    ID INT PRIMARY KEY NOT NULL,
    Name VARCHAR(255) NOT NULL,
    Anschrift VARCHAR(255),
    Art VARCHAR(100)
);
```

3.1.2 SQL - Abfrage von relationalen Datenbanken

Für die Verwaltung und Abfrage wird SQL (Structured Query Language) verwendet. Mit dieser Syntax können Tabellen erstellt, Daten eingefügt, aktualisiert und gelöscht und Daten abgefragt werden.

Anzeige aller Attribute einer Tabelle:

```
SELECT * FROM table_name;
```

Anzeige definierter Attribute einer Tabelle:

```
SELECT column1, column2 FROM table_name;
```

Gefilterte Anzeige einer Tabelle:

```
SELECT * FROM table_name WHERE condition;
```

Daten aus mehreren Tabellen abrufen (Join):

```
SELECT t1.column1, t2.column2
FROM table1 t1
JOIN table2 t2 ON t1.id = t2.id;
```

3.2 Graphdatenbank

Eine Graphdatenbank basiert auf dem Graphenkonzept.
Ein Graph besteht aus Knoten und Kanten (Beziehungen), welche die Verbindungen zwischen den Knoten darstellen.
Die Stärke der Graphdatenbank liegt in der Darstellung von komplexen Beziehungen.

Knoten: Jeder Knoten repräsentiert eine Entität bzw. Objekt. Jeder Knoten hat eine eindeutige ID oder Bezeichner, um auf diesen zugreifen zu können. Es können auch Attribute hinterlegt werden, um zusätzliche Informationen zu speichern, wie z.B. Geburtsjahr, Wohnort einer Person.

Kanten: Die Kanten verbinden die Knoten und repräsentieren damit die Beziehungen unter den Objekten. Die Kanten können gerichtet und ungerichtet sein. Bei einer gerichteten Beziehung muss die Richtung vom Quell- zum Zielknoten beachtet werden, wohingegen eine ungerichtete Kante eine symmetrische Beziehung darstellt.
gerichtete Beziehung: Ein Unternehmen ist abhängig vom Bericht des Wirtschaftsprüfers.
*ungerichtete Beziehung:** Unternehmen A arbeitet gemeinsam mit Unternehmen B an einem Projekt.

Label: Label werden verwendet, um die Knoten zu kategorisieren/gruppieren. Ein Knoten kann auch mehrere Label besitzen, um die Zugehörigkeit an verschiedenen Kategorien darzustellen (z.B. Unternehmensbranche).

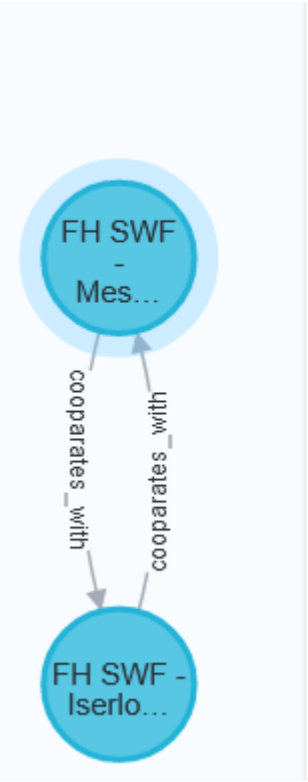
3.2.1 Erstellung eines Datensatzes

1. Knotenerstellung: Es wird zuerst ein Knoten erstellt, der die Entität repräsentiert.
2. ID: Der Knoten benötigt eine eindeutige Identifikationsnummer, welche automatisch erzeugt oder manuell festgelegt werden kann.
3. Knoten einfügen: Wenn die beiden notwendigen Elemente (Knoten und ID) festgelegt sind, kann der Knoten eingefügt werden.
4. Beziehungen/Kanten festlegen: Wenn der Knoten Beziehungen zu anderen Knoten hat, können diese hinzugefügt werden.

Beispiel: Folgender Code legt in neo4j zwei Knoten und die entsprechenden Beziehungen an.

```
CREATE (:University {id: 4711, name: 'FH SWF - Iserlohn'}),
       (:University {id: 1234, name: 'FH SWF - Meschede'})
WITH *
MATCH (u1:University {id: 4711}), (u2:University {id: 1234})
```

```
CREATE (u1)-[:cooparates_with]->(u2),
       (u2)-[:cooparates_with]->(u1)
```



Node properties

University

<id>	514
id	1234
name	FH SWF - Meschede

3.2.2 Cypher - Abfrage von Graphdatenbanken

Um Daten abzufragen wird die Abfragesprache Cypher verwendet.
Es werden folgend nur einige grundlegende Befehle gezeigt.\

Abfrage aller Knoten

```
MATCH (n)
RETURN n
```

Abfrage aller Kanten/Beziehungen

```
MATCH ()-[r]-()
RETURN r
```

Abfrage von Knoten mit definierten Eigenschaften

```
MATCH (n:Label)
WHERE n.property = value
RETURN n
```

Beziehung zwischen zwei Knoten abfragen

```
MATCH (n1)-[r]->(n2)
WHERE n1.property = value1 AND n2.property = value2
RETURN r
```

3.3 Zeitseriendatenbank

Zeitserien fallen überall dort an, wo eine Metrik zeitlich betrachtet wird, wie z.B. Umsatz oder EBIT. D.h. zu jedem Messwert gibt es einen zeitlich zugeordneten Zeitstempel, wobei die einzelnen Zeitpunkte zu einer Serie zusammengefasst werden, um den Zusammenhang zu betrachten.
Diese Datenbanken sind spezialisiert auf die Speicherung, Verwaltung und Abfrage von Zeitserien.
Die folgenden Erklärungen beziehen sich auf die InfluxDB.

Bucket: Der Bucket separiert Daten in verschiedene Speicher und ist mit der Datenbank bei relationalen Datenbanken vergleichbar.

Datapoint: Unter dem Bucket werden die Datenpunkte gespeichert. Ein Datapoint setzt sich aus mehreren Elementen zusammen, welche erorderlihc oder optional sind:

Element	Eigenschaft
Measurement	Datentyp: String Leerzeichen sind verboten Max. 64kB
Tags	Sind optional Bestehen aus einem Key/Value-Paar Datentyp: String Leerzeichen sind verboten Max. 64 kB
Fields	Min. 1 Field=value Paar wird benötigt Nicht alle Felder müssen in jedem Punkt vorhanden sein Datentypen: Float, String, Integer, Boolean
Timestamp	Sind optional Influx schreibt standardmäßig die Systemzeit als Zeitstempel Genauigkeit kann eingestellt werden (Default: Nanosekunden)

3.3.1 Erstellung eines Datensatzes

Die Einrichtung von Zeitseriendatenbanken erfolgt mit der CLI von Influx.

Anlegen eines Buckets:

```
CREATE DATABASE finance
```

3.3.2 FluxQuery

Zur Abfrage von Datenpunkten gibt es FluxQuery, welche sich stark an SQL orientiert. \

Abrufen aller Daten aus Bucket:

```
from(bucket: "my-bucket")
```

Festlegen des Zeitbereich:

```
range(start: -1h, stop: now())
```

Filtern nach Bedingungen:

```
filter(fn: (r) => r._measurement == "temperature")
```

Transformieren von Datenpunkten:

```
map(fn: (r) => ({r with temperatureF: r.temperature * 2.34 + 123}))
```

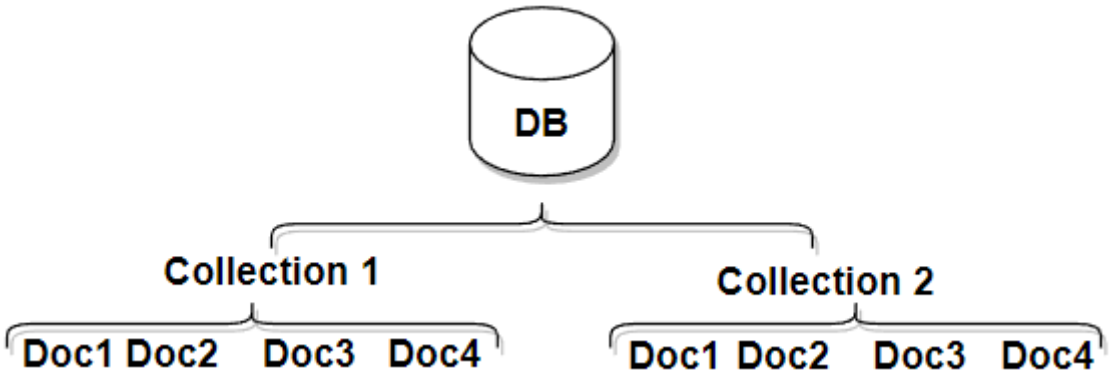
3.4 Dokumenten Datenbank

Eine Dokumentendatenbank ist ein System, welches für das Speichern von Dokumenten entwicklet wurde. Es gibt verschiedene Arten von Dokumenten, wie z.B. Textdateien (JSON, HTML, XML) oder PDF. Es muss kein Schema für die Dokumente festgelegt werden, dadurch ist es möglich Dokumente mit verschiedenen Datenfeldern zu speichern. Gleiche oder ähnliche Dokumente werden gemeinsam in *Collections* gespeichert. Die wichtigsten Elemente einer Dokumenten-Datenbank sind:

Database: Unter Database versteht man einen Container, unter welchem Dokumente gespeichert werden. Dies dient der Isolierung bzw. logischen Trennung von Daten.

Collection: Collections werden verwendet, um Dokumente mit ähnlichen Eigenschaften zusammenzufassen. Da Dokumenten-Datenbanken schemenlos sind, dienen die Collections der Organisation.

Document: Das Dokument ist ein einzelnes Datenobjekt und die kleinste Einheit in einer Dokumenten-DB. Ein Dokument kann z.B. ein JSON mit einer eigenen internen Struktur.



3.4.1 Erstellen einer Collection / Ablegen von Dokumenten

Folgend ein Code-Snippet zum Verbinden mit der Datenbank, Anlegen einer Collection und ablegen von Dokumenten.

```
from pymongo import MongoClient

# Verbindung zur MongoDB-Datenbank herstellen
client = MongoClient('mongodb://localhost:27017')

# erstelle ein Cleint-Objekt zur Datenbank
db = client['transparenz']

# Collection erstellen
collection = db['Tagesschau_API']

# Beispiel-Dokumente einfügen
doc1 = {
    'title': 'BASF wird verkauft!',
    'content': 'BASF wird an Bayer AG verkauft',
    'date': '2023-06-22'
}

doc2 = {
    'title': 'Bayer Aktie erreicht Rekordniveau',
    'content': 'Aufgrund des Zukaufs von BASF.....',
    'date': '2023-06-23'
}

# Dokumente in die Collection einfügen
collection.insert_one(doc1)
collection.insert_one(doc2)

# Verbindung zur Datenbank schließen
client.close()
```

3.5 Aufbau einer Datenbank

Vor dem Aufbau einer relationalen Datenbank sollten planerische Schritte durchgeführt werden, um ein System zu entwerfen, dass den Anforderungen gerecht wird.
Die wichtigsten Schritte sind:

Anforderungsanalyse: Identifikation und Definition von Anforderungen an die Datenbank durch Betrachtung des Anwendungsfalls.

Datenmodell: Analysieren der Strukturen und Beziehungen, die sich aus der Anforderungsanalyse ergeben. Auswahl eines Datenbankmodells, welches am besten geeignet ist.

Tabellenentwurf: Basierend auf den identifizierten Anforderungen wird die Tabellenstruktur der Datenbank entworfen. Für jede Tabelle werden Spaltennamen, Datentyp und mögliche Einschränkungen wie Primärschlüssel und Fremdschlüssel definiert.

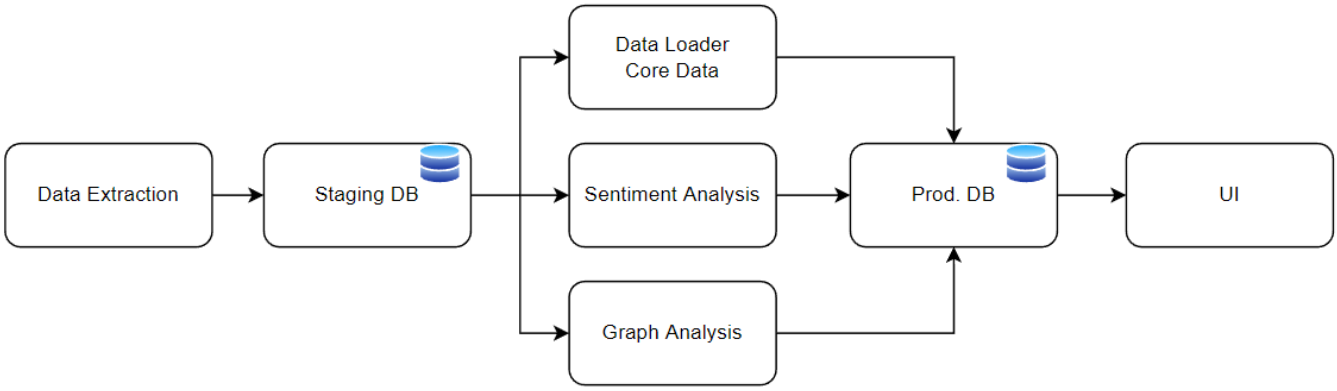
Erstellung der Tabellen: Wenn der Tabellenentwurf schlüssig ist und bereits diskutiert wurde, können die Tabellen erstellt werden. Es werden die zuvor festgelegten Bezeichner, Datenytpen und Constraints hinzugefügt.

Beziehungen festlegen: Um die Beziehungen zwischen Tabellen festzulegen, werden Fremdschlüssel verwendet. Mit Fremdschlüsseln verknüpft man Tabellen mit den Primärschlüsseln anderer, abhängiger Tabellen.

4. Datenbanken Transparenzregister

Nachdem die Datencluster identifiziert wurden, welche für das Transparenzregister notwendig sind, wurde Rechereche zu den benötigten Datenquellen betrieben.
Es gibt verschiedene Quellen, mit unterschiedlichen Schnittstellen bzw. Zugriff auf die Daten, z.B. mit API´s oder über Web Scrapping.

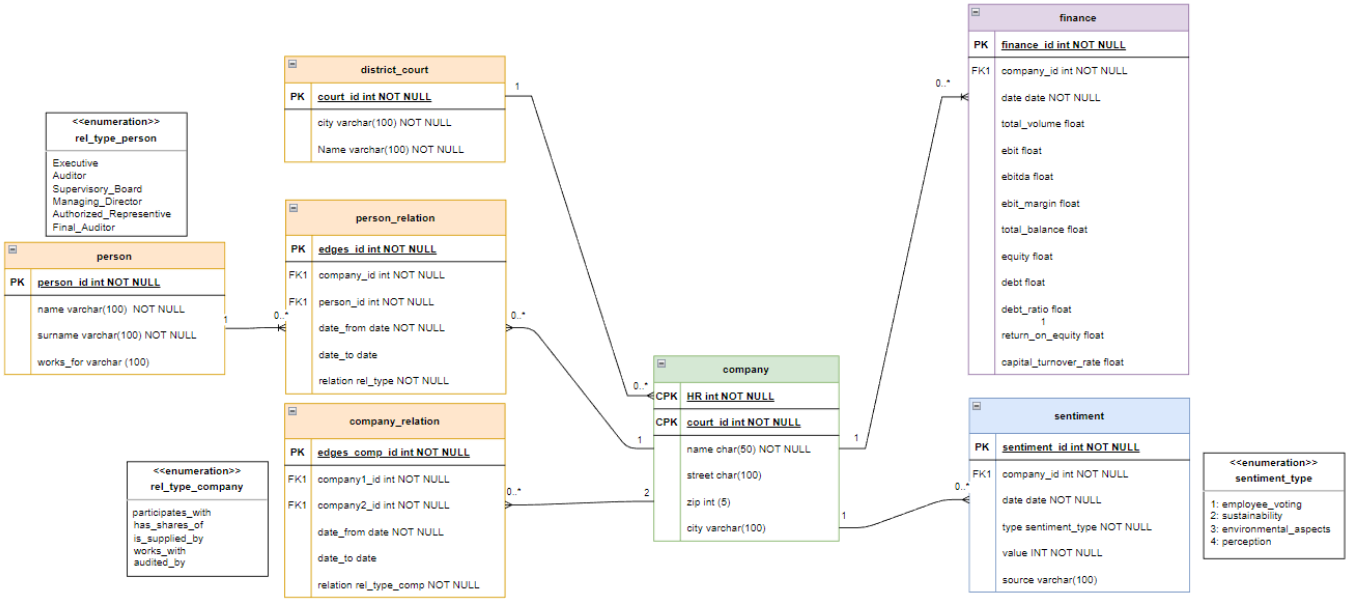
Es wurde eine Architektur definiert, welche den Aufbau der späteren Software skizziert:



Mittels geeigneter Techniken werden Daten aus diversen Quellen extrahiert (Data Extraction) und in der Staging DB gespeichert. Mit unterschiedlichen Daten-Extraktionspipelines (Dazta Loader, Sentiment Analysis, Graph Analysis) werden die Daten aus der Staging DB verarbeitet und die strukturierten und aufbereiteten Daten in der Production DB abgelegt.
Das Frontend kann auf diese strukturierten Daten zugreifen, um diese zu visualisieren.

4.1 Production DB - relationales Datenbankmodell

Für die Production DB ist eine relationale Datenbank vorgesehen, da diese die Daten organisiert und durch Verwendung von definierten Schemata strukturiert.
Diese Strukturen erleichtern die Wartung und Integration zwischen Back- und Frontend.



Zentrales Element ist die Stammdatentabelle **company**, welche einen zusammengesetzten Primärschlüssel aus der Nummer des Handelsregister und dem zuständigen Amtsgericht bildet.
Die Handelsregisternummer ist nicht eindeutig und wird deutschlandweit mehrfach vergeben, allerdings nur einfach unter einem Amtsgericht.

Es schließt sich die Tabelle **finance** an, in welcher die Finanzdaten persisitiert werden. Diese steht in einer 1:n Beziehung zur Unternehmenstabelle, da ein Unternehmen viele Finanzdaten haben kann und jeder Datensatz genau einem Unternehmen zugewiesen ist.
Die einzelnen Metriken wurden als Attribute definiert, wodurch es viele NULL-Werte in jeder Zeile gibt.
Vorteilhaft bei dieser Notation ist allerdings, dass die Metriken durch den Spalztenbezeichner eindeutig sind.

Die Tabelle **Sentiment** speichert die Stimmungsdaten zu einem Unternehmen. Auch hier besteht eine 1:n Beziehung zu der Unternehmenstabelle. Es gibt einen eigenen Enumeration-Typ, der die Art der Stimmungsdaten festlegt.

Die Tabelle **district_court** speichert die Amtsgericht, unter welchen die Unternehmen registriert sind. Diese Information ist wichtig, um mit der Handelsregisternummer und dem Amtsgericht ein Unternehmen eindeutig zu identifizieren.

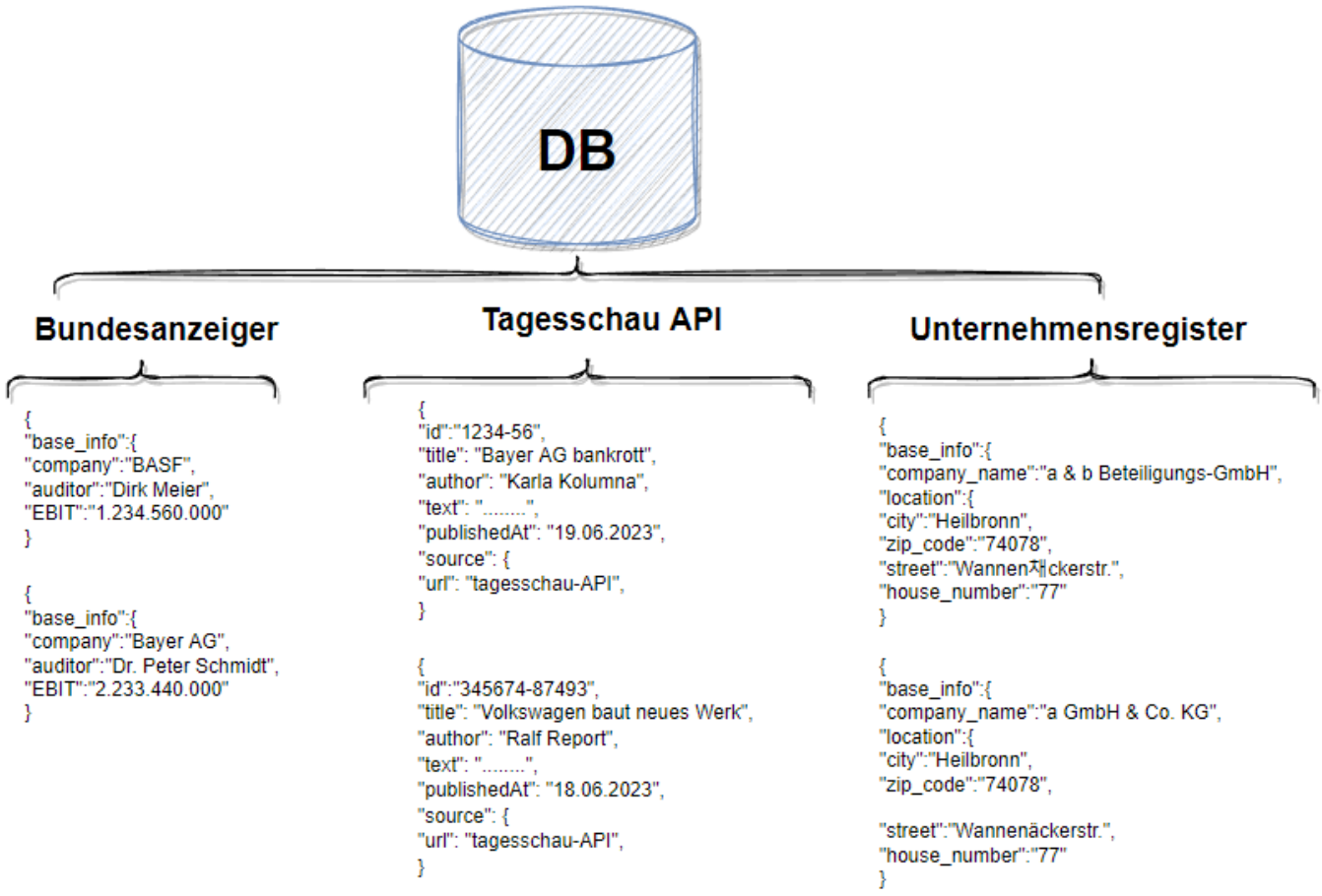
Die Tabelle **person** speichert Personen, welche unterschiedliche Beziehungen zu Unternehmen haben können. Daraus ergibt sich eine n:m Beziehung (many-to-many), da jede Person mehrere Beziehungen zu einem Unternehmen haben kann bz. jedes Unternehmen mehrfach mit einer Person in Verbindung steht.
Um diese Relation aufzulösen, wird eine Beziehungstabelle **person_relation** benötigt, um die n:m Beziehung auf zwei 1:n Beziehungen zu reduzieren. Diese enthält die Fremdschlüssel der bezogenen Tabellen, um die Beziehung zu modellieren.

Abschließend gibt es noch die Tabelle **company_relation**, welche die Verbindung zwischen Unternehmen modelliert. Hierfür wurde ein Enumaration-Typ erzeugt, welcher die Art der Beziehung angibt (wird_beliefert_von, arbeitet_mit, ist_beteiligt_an, hat_Anteile_an).

4.2 Staging DB

Die Staging DB ist eine dokumentbasierte Datenbank zu Speicherung von unstrukturierten und semi-strukturierten Daten. Sie dient als Zwischenspeicher oder "Rohdatenbank" für die Extraktions-Pipelines.
Aufgaben der Staging-DB:
1. Datenvorbereitung: Sammlung und Speicherung von Rohdaten aus verschiedenen Quellen
2. Überprüfung: Entsprechen die Daten den Anforderungen ggfs. Ermittlung von Fehlern oder Inkonsistenzen
3. Testumgebung: Die Rohdaten aus der Staging DB können mehrfach verwendet werden, um verschiedene Szenarien und Funktionalitäten der Extraktionspipelines zu erproben
4. Backup: Wenn sich im Laufe des Projekts eine Datenquelle ändert (z.B. Struktur oder Zugang zum Bundesanzeiger) sind die Daten weiterhin verfügbar oder wenn es Änderungen am Schema der Production DB gibt, kann durch eine Änderung am Data Loader das neue Tabellenschema implementiert werden

Die Staging DB erhält Collections der unterschiedlichen Quellen, unter welchen die Dokumente gespeichert werden.



4.3 SQL Alchemy

SQL Alchemy ist eine Python Bibliothek, um mit relationalen Datenbanken zu kommunizieren. Dieses ORM (Object-Relational-Mapping) Framework bildet die Datenbanktabellen als Pythonklassen an und vereinfacht damit das Erstellen, Lesen, Aktualsieren und Löschen von Daten aus Pythonanwendungen.
Wichtige Eigenschaften:

- erleichterte Entwicklung: durch die Abbildung von Datenbanktabellen als Pythonklassen wird durchgängig Pythoncode verwendet

- Flexibilität: Durch Verwendung eines Backend-Treibers für die unterschiedlichen Datenbanken, muss der Code nicht geändert werden. Wenn eine andere Datenbank zum Einsatz kommt, muss nur der Treiber ausgetauscht werden (Plattformunabhängigkeit)
- Erhöhung der Produktivität: Es werden keine Kompetenzen für SQL Programierung und Wartung benötigt.

5. Proof of Concept

5.1 Docker

Für die Umsetzung der bisher vorgestellten theoretischen Betrachtungen wird ein Docker Container verwendet. Dieser Container beinhaltet eine relationale und eine dokumentbasierte Datenbank.
Mit Jupyter Notebooks soll die Implementierung und Befüllung der Datenbank erprobt werden, um als Startpunkt für die anstehende Softwareentwicklung zu dienen.

```
version: "3.8"
services:
  db:
    image: postgres:14.1-alpine
    container_name: postgres
    restart: always
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
       - ./PostgreSQL:/var/lib/postgresql/data
  pgadmin:
    image: dpage/pgadmin4:7.2
    container_name: pgadmin4_container
    restart: always
    ports:
      - "5050:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@fh-swf.de
      PGADMIN_DEFAULT_PASSWORD: admin
    volumes:
      - ./pgadmin:/var/lib/pgadmin

  mongodb:
    image: mongo:7.0.0-rc4
    ports:
      - '27017:27017'
    volumes:
      - ./mongo:/data/db
```

Eintrag	Beschreibung
version	Version von docker-compose
services	Definition der Services, welche gestartet werden

Option	Beschreibung
image	Angabe des zu verwendenden Image
restart	Option, um Container erneut zu starten, falls dieser gestoppt wurde
environment	Umgebungsvariablen, wie z.B. Username und Passwort
Ports	Mapping des Containerports zum Port der Hostmaschine
volumes	Angabe eines Volumes zum Persistieren der Containerdaten

Beim Ausführen der docker-compose werden in diesem Verzeichnis Ordner für die Datenablage angelegt. Da zum Verfassungszeitpunkt noch nicht feststeht, wie im Projekt der Datenaustausch stattfindet, könnten diese Ordner bzw. die Volumes einfach untereinander ausgetauscht werden.

Zum Starten des Containers den folgenden Befehl ausführen:

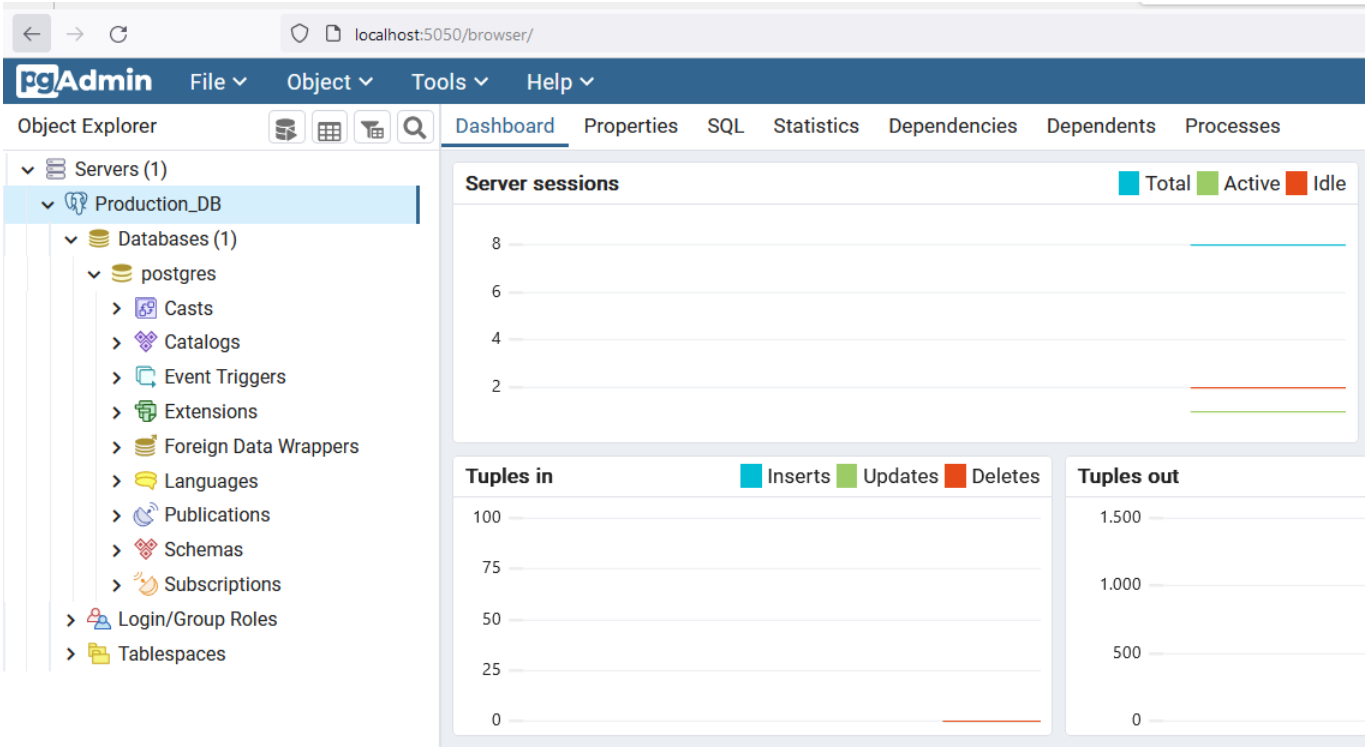
```
docker-compose -f docker-compose.yml up
```

5.2 PG Admin

PG Admin ist ein grafisches Administartionstool für Postgres. Wenn der Container gestartet ist, kann man sich über http://localhost:5050/browser/ mit dem Web-UI verbinden.
Dieses Tool dient lediglich der Überprüfung von Commits der Tabellen und daten.

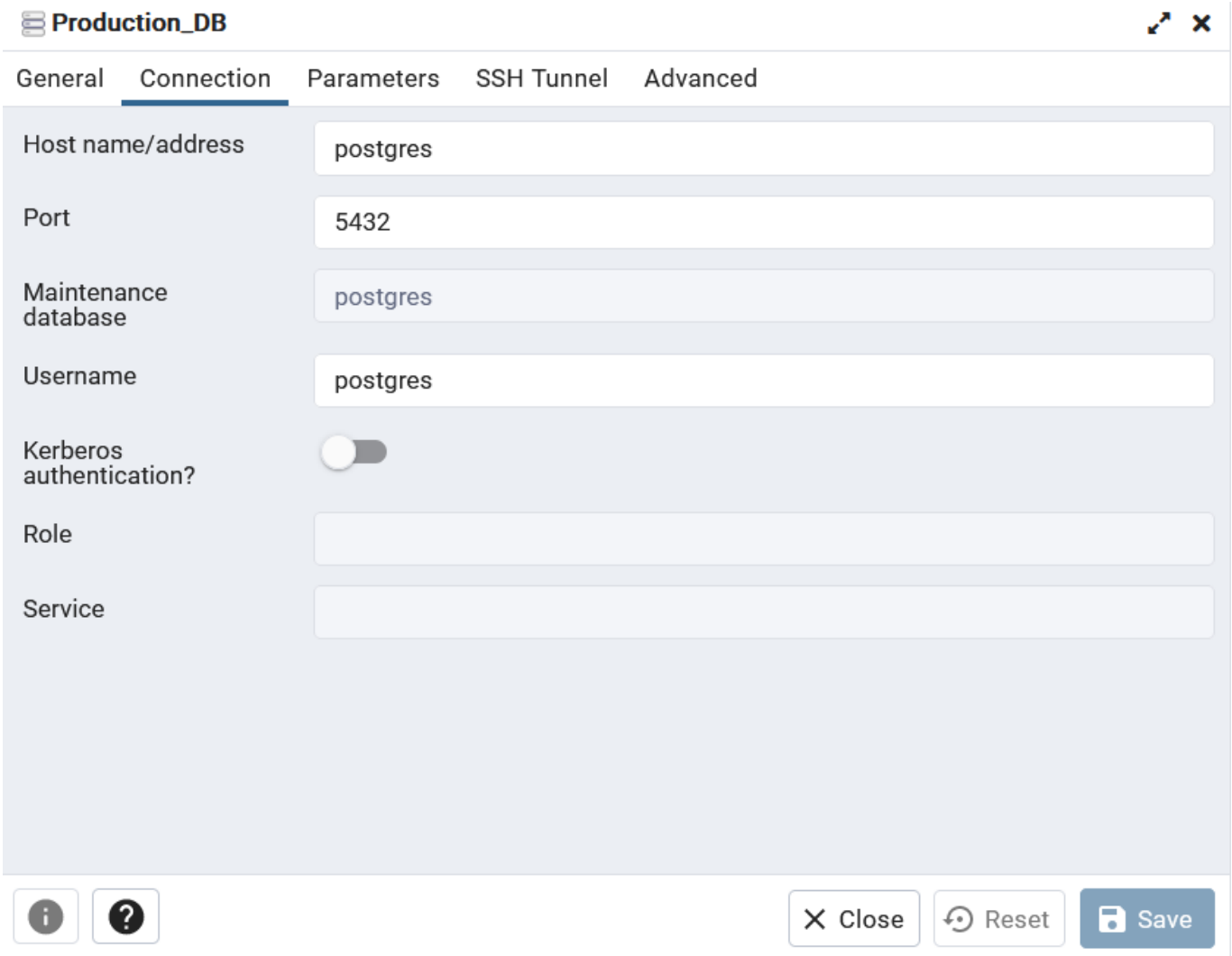
Die Anmeldedaten lauten:

> User: admin@fh-swf.de
> Passwort: admin



Zuerst muss der Server angelegt werden, dafür einen Rechtsklick auf Server und den Button „Register“ auswählen. Im geöffneten Dialog muss die Konfiguration festgelegt werden.

Reiter	Parameter	Wert
General	Name	postgres
Connection	Host name/address	postgres (siehe docker-compose)
Connection	Username	postgres (siehe docker-compose)
Connection	Password	postgres (siehe docker-compose)



5.3 Erstellen von Mock Daten

Unternehmensstammdaten:
Um das Konzept und den Umgang mit den ausgewählten Datenbanken zu überprüfen, sollen Daten in die Datenbank geschrieben werden. Hier für wurde auf Statista recherchiert, welches die größten deutschen Unternehmen sind, um einen kleinen Stamm an Unternehmensdaten zu generieren (01_Stammdaten_Unternehmen_HR.csv). / Die Relation zu den Amtsgerichten ist frei erfunden und wurde nicht recherchiert.

Amtsgerichte: Die Amtsgerichte sind aus https://www.gerichtsverzeichnis.de/ extrahiert, wobei lediglich 12 Amtsgerichte eingefügt wurden (Amtsgerichte.csv).

Finanzdaten: Es wurden für drei Unternehmen (EON, Telekom, BASF) die Finanzdaten bezüglich Umsatz, Ebit und Ebitda auf Statista ermittelt und als separate Dateien gespeichert (BASF_data.csv, Telekom_data.csv, EON_data.csv).

Personen: Die Personentabelle ist frei erfunden. Mit einer Onlinebibliothek wurde 1000 Vor- und Nachnamen erzeugt und gespeichert (Person1000.csv).

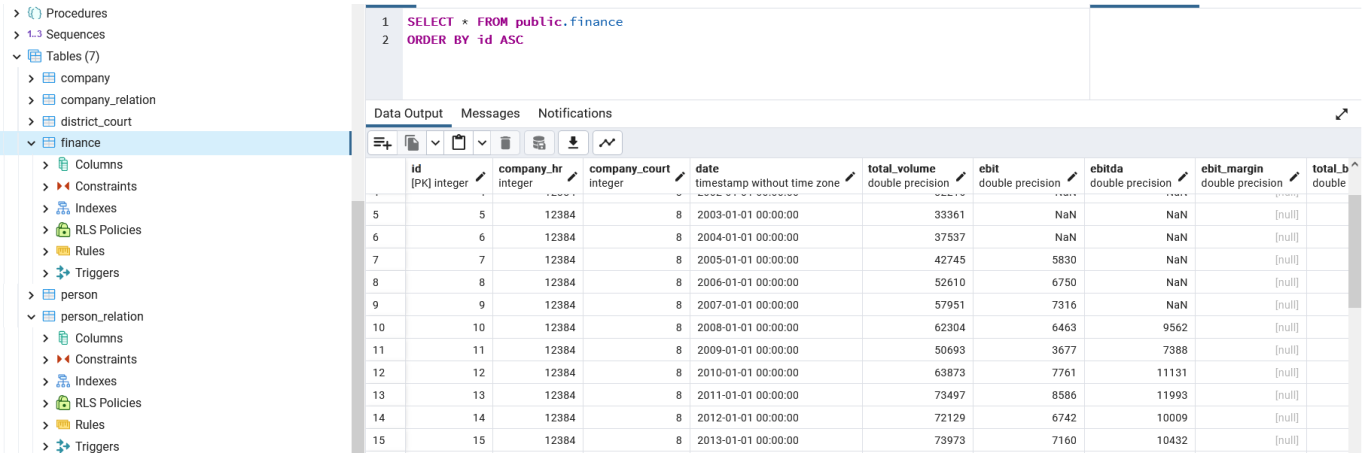
Personen-Unternehmens-Beziehung: Diese Tabelle ist zufällig erzeugt und dient lediglich für weitere Experimente. Hierfür wurde ein Python-Skript erstellt, welches mit der mehreren Random-Funktionen die Beziehungen zufälloig generiert.

Sentiment: keine Mock-Daten vorhanden

Unternehmens-Unternehmens-Beziehung: keine Mock-Daten vorhanden

5.4 Anlegen der relationalen Tabellen

Für das Verbinden zu der Postgre Datenbank und das Anlegen der Tabellen wird ein Jupyter Notebooks verwendet (11_Create_Tables_with_SQL-Alchemy.ipynb).
Die benötigten Bibliotheken werden importiert und das Erstellen von Tabellen als Python-Objekte beschrieben.
Nach dem Anlegen der Tabellen werden die Mock-Daten in die Datenbank geschrieben.
Eine Überprüfung, ob die Daten abgelegt wurden ist sehr einfach mit PGAdmin möglich.



Das grundsätzliche Vorgehen bei der Verwendung von SQLAlchemy ist:

1. Verbindung zur Datenbank herstellen

```
from sqlalchemy import create_engine
 # Connection URL für postgres
 url = URL.create(
 drivername="postgresql",
 username="postgres",
 password="postgres",
 host="localhost",
 database="postgres")
```

```
#Verbindung zur Datenbank
engine = create_engine(database_url)
```

2. Erstellen einer Klasse als Repräsentation der Tabelle.

> Es ist üblich und empfehlenswert die Klassendefinitionen in einer separaten Datei vorzunehmen (model.py), damit diese auch in andere Modulen importiert und verwendet werden können

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class MyClass(Base):
__tablename__ = 'company'

id = Column(Integer, primary_key=True)
name = Column(String)
city = Column(String)
```

3. Starten einer Session/Verbindung, um Daten lesen und schreiben zu können

```
from sqlalchemy.orm import sessionmaker

#starte die Verbindung
Session = sessionmaker(bind=engine)
session = Session()
```

4. Daten abfragen

```
# Alle Daten der Klasse/Tabelle abrufen
data = session.query(MyClass).all()
```

5. Daten speichern, wenn z.B. Datensätze in die Datenbank geschrieben werden, muss dies mit der **commit()**-Funktion ausgeführt werden. Das folgende Snippet iteriert durch einen Dataframe, um jede Zeile in die Datenbank zu schreiben.

```
for i in range(len(df)):
    #get data from dataframe
    myNewData=MyClass(
    name = str(df['Name'].iloc[i]),
    city = str(df['Surname'].iloc[i])
)
```

```
        session.add(myNewData)
        session.commit()
```

5.5 Abfragen der Datenbank

Der folgende Code-Snippet zeigt, wie man eine Abfrage gestaltet.

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

# Erstelle eine SQLite-Datenbankdatei oder gib den Pfad zur vorhandenen Datei an
url = URL.create(
    drivername="postgresql",
    username="postgres",
    password="postgres",
    host="localhost",
    database="postgres"
)

#Erstelle eine Engine zur Verbindung mit der Datenbank
engine = create_engine(url)

#Erstelle eine Klasse, die eine Tabelle repräsentiert
Base = declarative_base()
class Company(Base):
    __tablename__ = 'company'

    hr = Column(Integer(), nullable=False, primary_key=True)
    court_id = Column(Integer, ForeignKey("district_court.id"), nullable=False,
primary_key=True)
    name = Column(String(100), nullable=False)
    street = Column(String(100), nullable=False)
    zip = Column(Integer(), nullable=False)
    city = Column(String(100), nullable=False)
    sector = Column(String(100), nullable=False)

    __table_args__ = (
        PrimaryKeyConstraint('hr', 'court_id', name='pk_company_hr_court'),
    )

#starte die Verbindung zur Datenbank
Session = sessionmaker(bind=engine)
session = Session()

#Abfrage aller Spalten der Tabelle/Klasse Company
Comps = session.query(Company).all()

#Gebe die Spalten name, hr und court_id der Tabelle company aus
```

```
for comp in Comps:
    print(comp.name, comp.hr, comp.court_id)
```

6. Zusammenfassung

Die vorliegende Seminararbeit behandelt das Thema der Datenspeicherung mit Fokus auf dem Projekt Transparenzregister. Es wurde erläutert, warum Daten gespeichert werden und welche Art von Daten es gibt. Für das Projekt sind Daten und die Speicherung eine Kernkomponente, um die geforderten Analysen bezüglich Verflechtungen, unternehmerischen Erfolgs und Aussenwahrnehmung zu ermöglichen.

Es wurden Datencluster definiert und entsprechende Quellen gefunden, welche über geeignete Extraktionspipelines die erforderlichen Informationen extrahieren. Zum Speichern dieser extrahierten Daten wurde ein relationales Modell erarbeitet, um ein Konzept für die folgende Implementierung zu haben.

Um das Konzept zu überprüfen, wurde ein Proof of Concept durchgeführt, um geeignete Werkzeuge zu erproben und das Modell auf seine Tauglichkeit zu überprüfen.
Hierbei wurde ein Dockercontainer eingesetzt, um die Datenbankumgebung bereitzustellen. Mithilfe der SQL-Alchemy-Bibliothek, wurden die Tabellen innerhalb der Datenbank erstellt.
Anschließend wurden die Tabellen mit eigenen Mock-Daten befüllt, um die Funktionalität der Datenbank zu testen.

Insgesamt bietet die Seminararbeit einen umfassenden Überblick über die Bedeutung der Datenspeicherung und die verschiedenen Arten von Datenbanken. Es wurde ein erstes relationales Modell und ein High level design für die Softwarearchitektur erarbeitet. Diese Arbeit hat grundsätzliche Fragen geklärt und Verständnis für die Datenspeicherung im Zusammenhang mit dem Projekt Transparenzregister geschaffen und unterstützt die weitere Entwicklung.

Quellen

Klug, Uwe: SQL-Der Einstieg in die deklarative Programmierung, 2. Auflage, Dortmund, Springer, 2017
Steiner, Rene: Grundkurs relationale Datenbanken, 10. Auflage, Wiesbaden, Springer, 2021
https://backupchain.de/daten-backup-tipps-3-wie-oft-daten-sichern/
https://www.talend.com/de/resources/strukturierte-vs-unstrukturierte-daten/
https://www.sqlservercentral.com/articles/creating-markdown-formatted-text-for-results-from-sql-server-tables
https://www.sqlalchemy.org/
https://medium.com/@arthurapp98/using-sqlalchemy-to-create-and-populate-a-postgresql-database-with-excel-data-eb6049d93402